ANALISIS EFISIENSI ALGORITMA AKUISISI FORENSIK DIGITAL

Disusun Oleh:

N a m a : Sujatmiko Wikantiyoso

NIM : 16523017

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2020

HALAMAN PENGESAHAN DOSEN PEMBIMBING

ANALISIS EFISIENSI ALGORITMA AKUISISI FORENSIK DIGITAL

TUGAS AKHIR

Disusun Oleh:

N a m a : Sujatmiko Wikantiyoso

NIM : 16523017

Yogyakarta, 10 Juli 2020

Pembimbing,

(Fietyata Yudha, S.Kom, M.Kom)

HALAMAN PENGESAHAN DOSEN PENGUJI

ANALISIS EFISIENSI ALGORITMA AKUISISI FORENSIK DIGITAL

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 21 Juli 2020

Tim Penguji

Fietyata Yudha, S.Kom., M.Kom. _____

Anggota 1

Fayruz Rahma, S.T., M.Eng. _____

Anggota 2

Elyza Gustri Wahyuni, S.T., M.Cs. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Sujatmiko Wikantiyoso

NIM : 16523017

Tugas akhir dengan judul:

ANALISIS EFISIENSI ALGORITMA AKUISISI FORENSIK DIGITAL

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apa pun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 07 Juli 2020

(Sujatmiko Wikantiyoso)

HALAMAN PERSEMBAHAN

Alhamdulillah hirabbil'alamin, segala puja bagi Allah saya panjatkan kepada Allah SWT. yang telah memberikan segala nikmat, karunia, kesehatan, kesabaran, petunjuk, kekuatan, dan perlindungan sehingga saya bisa menyelesaikan tugas akhir ini dalam keadaan sehat. Tak lupa shalawat serta salam saya curahkan kepada Rasul Nabi Muhammad SAW. yang karena kesabaran, ketabahan, dan ketulusan hatinya menjadikannya sebagai suri tauladan bagi umat manusia sehingga dapat membimbing manusia menuju jalan yang baik dan lebih terang.

Dengan rasa bangga dan bahagia, alhamdulillah saya telah memenuhi kewajiban sebagai mahasiswa untuk menyelesaikan tugas akhir ini. Dengan rasa syukur saya persembahkan tugas akhir ini kepada:

1. Bapak Sancoyo Setyo Atmanto dan Ibu Tri Suwarni Widayati yang dengan bimbingan, kesabaran, nasihat, kasih sayang, dukungan, serta ridho beliau berdua saya bisa menyelesaikan tugas akhir ini.
2. Kedua adik saya, Arif Wijanarko dan Awang Sulistyantoro yang selalu memberikan semangat, motivasi, dan doa.
3. Bapak Fietyata Yudha S. Kom, M. Kom yang dengan kesabarannya telah membimbing saya dalam menyusun tugas akhir.
4. Keluarga besar saya yang selalu mendoakan dan memberikan saya semangat untuk selalu mengingatkan kewajiban yang harus saya laksanakan sebagai mahasiswa.
5. Teman-teman *hexadecima* yang sama-sama berjuang untuk melaksanakan kewajiban.
6. Teman-teman lainnya yang selalu memberikan dukungan.
7. Semua orang yang telah membantu dan selalu memberikan saya motivasi untuk menyelesaikan tugas akhir ini.

HALAMAN MOTO

"*Dan tidak ada satu pun makhluk bergerak (bernyawa) di muka bumi melainkan semuanya telah dijamin rezekinya oleh Allah. Dia mengetahui tempat kediaman dan tempat penyimpanannya. Semua itu (tertulis) dalam Kitab yang nyata (Lauh Mahfuzh)".*

(QS. Hud ayat 6)

"Dan Katakanlah: "Bekerjalah kamu, maka Allah dan Rasul-Nya serta orang-orang mukmin akan melihat pekerjaanmu itu, dan kamu akan dikembalikan kepada (Allah) Yang Mengetahui akan yang gaib dan yang nyata, lalu diberitakan-Nya kepada kamu apa yang telah kamu kerjakan."

(QS. At-Taubah ayat 105)

"Sesungguhnya Allah tidak akan mengubah nasib suatu kaum kecuali kaum itu sendiri yang mengubah apa yang ada pada diri mereka"

(QS. Ar-Ra'd ayat 11)

"Allah tidak akan memberikan suatu cobaan melainkan sesuai dengan kesanggupannya. Ia mendapat pahala (dari kebajikan) yang diusahakannya dan ia mendapat siksa (dari kejahatan) yang dikerjakannya"

(QS. Al-Baqarah ayat 286)

KATA PENGANTAR

Assalamual'aikum Warahmatullahi Wabarakatuh

Alhamdulillah, pertama puja dan puji syukur ke hadirat Allah SWT. atas segala karunia, rahmat, rizqi, dan kasih sayang-Nya. Selanjutnya shalawat serta salam kepada Nabi Muhammad SAW. yang telah memberikan cahaya bagi umat manusia.

Puji syukur kembali haturkan kepada Allah SWT. yang atas Ridha-Nya penelitian tugas akhir dengan judul *"Analisis Efisiensi Algoritma Akuisisi Forensik Digital"* dapat terselesaikan dengan baik dan lancar. Laporan tugas akhir ini disusun sebagai syarat untuk menyelesaikan jenjang pendidikan dan meraih gelar di Program Studi Strata Satu (S1) Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia. Penyusunan tugas akhir ini juga memberikan manfaat bagi peneliti yaitu mendapatkan ilmu baru yang tidak diajarkan di kampus.

Dalam menyusun laporan peneliti menyadari adanya bantuan, bimbingan, doa serta nasihat dari berbagai pihak. Oleh karena itu dengan kesempatan ini izinkan peneliti dengan rasa syukur dan kerendahan hati mengucapkan terima kasih kepada beberapa pihak:

1. Keluarga besar penulis, Bapak Sancoyo, Ibu Tri Suwarni, Adik Arif Wijanarko, dan Adik Awang Sulistyantoro yang selalu memberikan dorongan, doa, dan bantuan moral maupun materi.
2. Bapak Fietyata Yudha S. Kom, M. Kom selaku dosen pembimbing tugas akhir yang dengan kesabarannya telah memberikan bimbingan kepada peneliti sehingga peneliti dapat menyelesaikan tugas akhir dengan baik.
3. Bapak Dr. Raden Teduh Dirgahayu, S. T, M. Sc selaku Ketua Program Studi Informatika – Program Sarjana Fakultas Teknologi Industri Informatika Universitas Islam Indonesia.
4. Teman-teman *Hexadecima* yang selalu memberikan motivasi, bantuan, serta doa.
5. Semua pihak yang telah membantu dan mendoakan sehingga peneliti mampu menyelesaikan laporan tugas akhir ini.

Semoga semua dukungan, bantuan, dan bimbingan yang diberikan kepada peneliti mendapatkan balasan dari Allah SWT. Dalam menyusun laporan tugas akhir ini, peneliti

menyadari masih banyaknya kekurangan dan jauh dari kata sempurna. Oleh karena itu peneliti mengharapkan masukan berupa kritik mau pun saran yang membangun agar mendapatkan hasil yang baik. Cukup sekian, semoga dengan adanya laporan ini dapat memberikan manfaat bagi kita semua. Terima kasih.

Wassalamu'alaikum Warahmatullahi Wabarakatuh

Yogyakarta, 07 Juli 2020

(Sujatmiko Wikantiyoso)

SARI

Di era industri 4.0 sekarang ini perkembangan teknologi digital sangat pesat. Hampir setiap sektor tidak terlepas dari perkembangan teknologi digital. Mulai naiknya grafik peningkatan teknologi dapat dilihat dari bertambahnya nilai pengguna internet dan alat komunikasi elektronik. Tingginya pengguna internet diikuti dengan bertambahnya kejahatan digital yang muncul. Kejahatan digital juga meninggalkan jejak, maka dibutuhkan orang dengan keahlian khusus untuk menanganinya. Untuk mengungkap kasus kejahatan digital maka diperlukan proses investigasi. Investigasi meliputi *assessment, acquisition, examination,* dan *reporting*.

Proses akuisisi saat ini masih memerlukan waktu yang lama. Sehingga diperlukan adanya analisis pada algoritma untuk mengetahui format akuisisi yang lebih baik. Dengan banyaknya kasus digital yang ada mengharuskan para ahli forensik digital untuk segera menyelesaikan pekerjaan tersebut. Mengetahui format yang tepat ketika melakukan akuisisi akan sangat membantu dalam mempercepat proses investigasi. Penelitian ini dilakukan dengan menganalisis algoritma akuisisi format Raw dan AFF dan menjelaskannya ke dalam notasi *Big-O.* Pada penelitian ini akan dibahas bagaimana format Raw lebih efisien dibandingkan dengan AFF dengan menghitung kompleksitas waktu yang menghasilkan nilai *Big-O.*

Kata kunci: Analisis Algoritma, AFF, Raw, dd.

GLOSARIUM

Compile	proses penyusunan yang dilakukan sebelum menjalankan suatu program agar bisa dieksekusi pada suatu sistem operasi.
Hard drive	media penyimpanan yang menggunakan piringan magnetis untuk menyimpan data.
Hashing	proses yang menghasilkan kombinasi angka unik yang bertujuan untuk menjaga integritas data.
Hoax	berita bohong atau ti dak benar dan tidak terbukti kebenarannya.
Imaging	proses yang dilakukan untuk melakukan akuisisi data pada media penyimpanan.
Investigator	orang yang melakukan investigasi (penyelidikan).
Log	catatan yang merekam semua kegiatan atau aktivitas
Opensource	suatu aplikasi yang dapat kode programnya dapat dikembangkan dengan bebas.
Pen drive	media penyimpanan portabel yang ukurannya sebesar ibu jari.
Source code	sekumpulan kode yang ditulis dengan bahasa pemrograman.
User	pengguna.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di era industri 4.0 sekarang ini perkembangan teknologi digital sangat pesat. Hampir setiap sektor tidak terlepas dari perkembangan teknologi digital. Mulai naiknya grafik peningkatan teknologi dapat dilihat dari bertambahnya nilai pengguna internet dan alat komunikasi elektronik. Menurut data yang dilansir dari situs *websindo.com* terdapat kenaikan persentase pengguna internet dari tahun 2018 ke 2019 sebesar 13%. Pesatnya perkembangan teknologi kini tidak lagi menghasilkan solusi saja, namun juga memunculkan masalah baru. Dampak negatif yang timbul dari pesatnya perkembangan tersebut merupakan sebab dari tidak terbatasnya teknologi untuk diterima siapa pun. Karena tingginya antusiasme yang terjadi terhadap teknologi digital, justru dapat menjadikan masalah baru. Banyaknya penggunaan internet memunculkan jenis kejahatan baru. Yaitu kejahatan siber atau sering disebut dengan *cybercrime*.

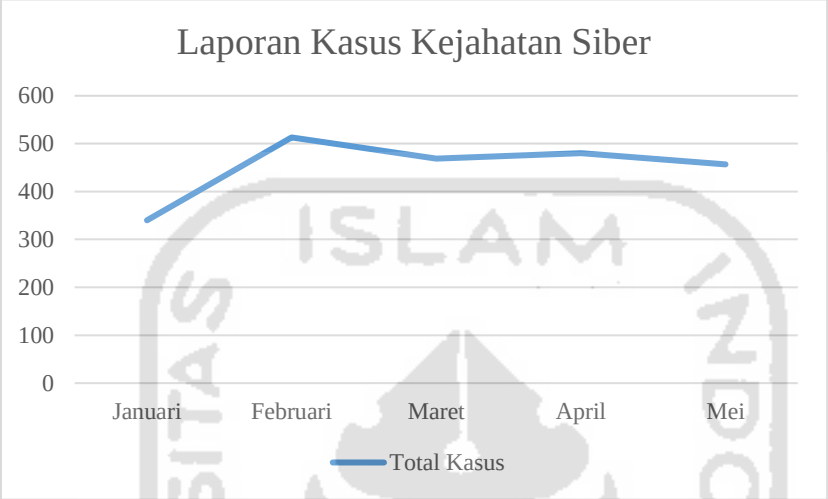
Cybercrime adalah segala bentuk kejahatan yang dilakukan melalui jaringan internet. Termasuk penggunaan perangkat khususnya komputer dan jaringan yang bertujuan untuk melakukan penipuan. Bentuk kejahatan bisa berupa ekstraksi data, masuk tanpa izin ke suatu sistem, menyisipkan virus yang semua bertujuan untuk merusak atau memanipulasi (Selina Kolls, 2016).

Menurut data Direktorat Tindak Pidana Siber (Dittipidsiber) Bareskrim Polri yang dimuat pada situs *patrolisiber.id* terdapat 1.919 kasus yang dilaporkan pada situs tersebut pada bulan Februari 2020 (Direktorat Tindak Pidana Siber Bareskrim Polri, 2020). Pada situs tersebut terdapat beberapa laporan data mengenai kejahatan siber yang terjadi di Indonesia antara lain, yaitu 573 penipuan *online*, 899 penyebaran konten provokatif, 165 pornografi, 111 akses ilegal, 27 perjudian, 15 pemerasan, 20 pencurian data/identitas, 62 manipulasi data.

Tabel 1. 1 Data kejahatan siber bulan Januari – Mei 2020

Bulan	Kasus
Januari	340
Februari	513

Maret	469
April	480
Mei	457
Total	2.259



Gambar 1. 1 Grafik kejahatan siber Januari – Mei 2020

Berdasarkan data pada Tabel 1.1 dan Gambar 1.1 per Januari 2020 - Mei 2020 masih banyak kasus kejahatan siber yang terjadi di Indonesia. Data tersebut hanya mencakup kasus yang dilaporkan saja. Pada data yang didapat dapat diketahui terjadi peningkatan sebesar 33,7% pada bulan Januari ke Februari. Masih banyaknya kasus kejahatan siber di Indonesia memerlukan penanganan khusus untuk mengungkap kejadian tersebut ke pengadilan. Namun banyaknya kasus yang ditangani tidak sebanding dengan jumlah tenaga ahli yang menangani kasus tersebut. Hal tersebut juga menyebabkan para tenaga atau ahli forensik digital kewalahan ketika menyelesaikan kasus yang banyak. Banyak kasus dikerjakan berdasarkan prioritas kerugiannya. Dampaknya banyak kasus kecil yang dikesampingkan terlebih dahulu, namun kasus kecil tersebut bisa menjadikan bom waktu bagi Indonesia karena lambatnya penanganan kasus. Misalnya kasus *hoax* yang sering terjadi di Indonesia. Karena dalam UU bahwa penyebaran berita *hoax* hanya dapat dipidanakan apabila terdapat pihak yang dirugikan sehingga kepolisian lebih berfokus pada pengungkapan kasus lain dari pada kasus *hoax* (Ervina Chintia, et.al., 2018).

Karena kejahatan digital juga meninggalkan jejak, maka dibutuhkan orang dengan keahlian khusus untuk menanganinya. Berbeda dengan kejahatan pada umumnya yang meninggalkan barang bukti fisik di lokasi kejadian, kejahatan digital meninggalkan bukti yang lebih tidak terlihat dan mudah menghilang. Dibutuhkan kemampuan cara me-*recovery* barang bukti yang sudah dihilangkan tersebut agar dapat dijadikan sebagai barang bukti yang sah di pengadilan. Untuk mendapatkan barang bukti digital tersebut diperlukan teknik khusus.

Salah satu cara yang dilakukan untuk mengungkap kejahatan siber adalah proses akuisisi. Proses akuisisi dilakukan saat *investigator* melakukan penyalinan data dari perangkat penyimpanan. Proses akuisisi memerlukan waktu yang lama berdasarkan format akuisisi dan *tool* yang digunakan. Dengan banyaknya kasus kejahatan siber mengharuskan para ahli forensik untuk memilih format akuisisi yang tepat sehingga *investigator* memiliki bukti yang cukup diikuti dengan efisiensi waktu saat melakukan akuisisi forensik digital. Proses akuisisi sangat penting. Pada tahap ini *investigator* harus berhati-hati dengan data yang dikerjakan. Karena sifat data yang rentan, mudah rusak, hilang, atau mudah dilakukan modifikasi.

Forensik digital menggunakan metode ilmiah dan bertujuan untuk pengumpulan, validasi, identifikasi, analisis, interpretasi, dokumentasi, dan pemaparan bukti dari dokumen digital atau merekonstruksi peristiwa yang mengandung unsur pidana, juga untuk tindakan pencegahan (B. Carrier, 2003). Forensik digital di Indonesia dimiliki oleh kepolisian, instansi swasta, dan akademisi. Minimnya ahli forensik digital di Indonesia menjadikan mereka untuk bekerja ekstra ketika mengatasi kasus yang banyak. Akuisisi data memerlukan waktu yang lama dan proses yang rumit. Ada beberapa metode yang digunakan selama akuisisi proses barang bukti digital, *manual acquisition, local acquisition, physical acquisition,* dan *Chipp-off* (A. Khawla dan J. Andy, 2012). Akuisisi adalah proses pengumpulan barang bukti dari perangkat elektronik, misalnya *hardisk* atau *flashdrive.*

Untuk mengetahui efisiensi pada algoritma maka diperlukan proses analisis. Analisis memerlukan data pendukung yaitu *flowchart* dan *pseudocode* untuk mempermudah dalam perhitungan. Kompleksnya suatu algoritma akuisisi forensik digital juga akan berpengaruh dengan waktu akuisisi. Efisiensi dianalisis dengan memperhitungkan 3 kondisi yaitu, *best-case, worst-case,* dan *average-case*. *Worst-case* sangat penting dalam memperkirakan kemungkinan terburuk pada algoritma sedangkan parameter yang lain seperti *best-case* dan *average-case* adalah parameter pendukung. Dari perhitungan analisis *worst-case* akan didapatkan notasi efisiensi algoritma dalam bentuk *Big-O.*

Penelitian ini dilakukan dengan menganalisis algoritma akuisisi format Raw dan AFF. Kemudian setiap algoritma yang sudah ada akan dibandingkan satu sama lain. Yaitu bagaimana efisiensi waktunya ketika algoritma proses akuisisi berjalan. Penelitian ini dirasa penting karena memang proses akuisisi saat ini masih memerlukan waktu yang lama. Diperlukan adanya analisis pada algoritma tersebut supaya proses akuisisi forensik digital bisa lebih efisien. Kemudian dengan banyaknya kasus digital yang ada mengharuskan para ahli forensik digital untuk segera menyelesaikan pekerjaan tersebut.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, penulis merumuskan masalah sebagai berikut:

1. Bagaimana membuat *flowchart* dari kode program akuisisi forensik digital format Raw dan AFF?
2. Bagaimana cara mengetahui kompleksitas waktu pada algoritma akuisisi forensik digital format Raw dan AFF?
3. Bagaimana cara mengetahui algoritma akuisisi forensik digital yang lebih efisien pada format Raw dan AFF?

1.3 Batasan Masalah

Agar penelitian lebih terarah terdapat beberapa batasan masalah, yaitu:

1. Studi kasus pada algoritma akuisisi format .dd dan .aff.
2. Pada format Raw analisis dilakukan ketika fungsi *copy_dd* dijalankan.
3. Pada format AFF menggunakan versi AFF1
4. Analisis efisiensi waktu menggunakan notasi *Big-O*
5. Analisis dilakukan dengan menghitung pada kondisi *worst case*.

‘

1.4 Tujuan Penelitian

Tujuan penelitian yang ingin dicapai adalah:

a. Memahami cara untuk menyusun *flowchart* dari kode program akuisisi.
b. Mengetahui cara menghitung efisiensi waktu algoritma akuisisi forensik digital pada format Raw dan AFF.
c. Mengetahui algoritma mana yang lebih efisien dengan menggunakan notasi *Big-O*.

1.5 Manfaat penelitian

Manfaat yang akan diperoleh dari penelitian ini adalah:

a. Untuk peneliti:
    1. Mengetahui cara untuk melakukan analisis pada suatu algoritma.
    2. Mendapatkan pembelajaran baru mengenai cara menghitung kompleksitas waktu suatu algoritma.
    3. Memiliki pandangan yang lebih mengenai ilmu forensik digital khususnya pada saat proses akuisisi.
b. Untuk pengembang:
    1. Hasil analisis yang didapat bisa dijadikan referensi untuk melakukan pengembangan terhadap metode yang sudah ada.
    2. Hasil analisis yang didapat bisa dijadikan acuan atau referensi untuk melakukan penelitian selanjutnya.

1.6 Metodologi Penelitian

Metodologi penelitian yang digunakan dalam analisis algoritma forensik terbaik, yaitu:

1. Identifikasi masalah

   Pada tahap ini peneliti mengangkat masalah yang muncul. Melihat seberapa penting masalah tersebut untuk diselesaikan dan bagaimana dampak yang ditimbulkan jika penelitian berhasil dilakukan.
2. Pengumpulan data

   Dalam tahapan ini peneliti mengumpulkan data dan mempelajari literatur sesuai dengan masalah yang muncul. Literatur tersebut mengenai karya ilmiah, jurnal, konferensi, diskusi, artikel ilmiah, dan buku sehingga dapat menjadi referensi dan membantu peneliti dalam pengerjaan tugas akhir.
3. Pembuatan diagram alur

   Pada tahap ini peneliti membuat diagram alur atau *flowchart* untuk mempermudah dalam melihat proses yang terjadi di dalam program. Diagram alur akan dibuat dengan membaca *kode* baris per baris untuk melihat operasi apa saja yang berjalan pada algoritma.
4. Membuat pseudocode.

Pseudocode dibuat sebagai alat tambahan untuk mempermudah proses analisis pada algoritma. Pada tahap ini proses yang dilakukan sama ketika akan membuat diagram alur yaitu dengan membaca *kode* baris per baris untuk memudahkan dalam menghitung efisiensi waktu.

5. Menghitung efisiensi waktu dengan notasi *Big-O.*

Cara untuk mengetahui efisiensi pada algorima salah satunya dengan penulisan menggunakan notasi *Big-O.* Penggunaan notasi ini digunakan untuk menyimpulkan hasil

6. Perbandingan

Peneliti melakukan perbandingan dari hasil yang didapat. Kemudian dilakukan analisis terhadap hasil tersebut. Peneliti akan melihat performa tiap metode akuisisi.

1.7 Sistematika Penulisan

Sistematika penulisan disusun untuk mempermudah pemahaman mengenai penelitian yang dilakukan. Secara garis besar sistematika penulisan pada penelitian ini adalah sebagai berikut:

BAB I PENDAHULUAN

Pada bab ini berisi tentang latar belakang masalah, rumusan masalah, Batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan penjelasan mengenai sistematika penulisan laporan penelitian.

BAB II LANDASAN TEORI

Bab ini membahas tentang teori-teori yang digunakan oleh peneliti sebagai landasan maupun rujukan dilakukannya penelitian yang dilakukan. Bab ini berisi tentang teori dasar mengenai analisis algroritma, teori kompleksitas waktu suatu algoritma, definisi diagram alur, teori komputer forensik, bahasa C, dan proses akuisisi forensik digital.

BAB III METODOLOGI PENELITIAN

Bagian ini berisi tentang metodologi penelitian yang digunakan sebagai langkah peneliti dalam mengumpulkan atau mencari data dan informasi guna mempermudah penelitian yang dilakukan. Kemudian dengan terkumpulnya data dan informasi nantinya akan dianalisis dan diolah untuk membantu penelitian.

BAB IV HASIL DAN PEMBAHASAN

Bab ini membahas hasil penelitian yang sudah dicapai oleh peneliti dan bagaimana peneliti melakukan perhitungan untuk menentukan kompleksitas waktu algoritma. Tahap ini juga berisikan grafik perbandingan antar format .dd maupun AFF (Advance forensik Format) untuk mempermudah melihat hasil mengenai algoritma mana yang lebih efisien.

BAB V KESIMPULAN DAN SARAN

Bagian ini berisi mengenai kesimpulan dan capaian yang telah dicapai peneliti dan saran untuk penelitian yang selanjutnya.

BAB II
LANDASAN TEORI

2.1 Penelitian Sebelumnya

Pada penelitian yang dilakukan didasari oleh berbagai penelitian sebelumnya yang sudah membahas mengenai format Raw dan AFF seperti yang tertera pada tabel 2.1.

Tabel 2. 1 Penelitian sebelumnya

No	Judul	Penulis	Pencapaian	Saran
1.	*Extending the Advanced Forensic Format to Accommodate Multiple Data sources, Logical evidence, Arbitrary Information and Forensic Workflow*	Michael Cohen, Garfinkel Simson, Bradley Schatz	Menjelaskan peningkatan yang signifikan untuk format AFF1. Pada AFF4 format *file* bisa lebih baik dalam kerangka kerja untuk manajemen bukti kasus, seperti kemampuan untuk menyimpan berbagai *device* dalam satu arsip. Peningkatan antara mekanisme penyimpanan dan perangkat lunak forensik untuk menyimpan barang bukti menggunakan AFF.	Cara kerja AFF4 sangat sederhana. Dengan hal tersebut diharapkan dapat meningkatkan kualitas AFF4 sebagai *platform* menejemen pada barang bukti.
2.	*Forensic Images: For Your Viewing Pleasure*	Sally Vandeven	Menjelaskan tentang pentingnya analisis forensik. Memaparkan beberapa *tools* yang sering digunakan dalam membuat *image* pada bukti forensik.	Pemilihan metode tergantung pada data yang akan diakuisisi.
3.	*A Literature review on Cyber Forensic and its Analysis tools*	Mandeep Kaur, Navreet Kaur, Suman Khurana.	Memeriksa alat khusus untuk forensik yang digunakan dalam menganalisis kelemahan, keamanan dalam forensik digital. Bukti digital dapat diperoleh dari struktur data yang terletak pada memori dengan alat yang berbeda. Cara baru untuk mengumpulkan barang bukti dengan cepat,	Karena meningkatnya jumlah pengguna internet di seluruh dunia, frekuensi serangan digital juga meningkat. Oleh karena itu

			berguna, dan penting untuk menangani kasus-kasus tersebut dengan cepat.	dibutuhkan metodologi yang efektif dalam mengembangkan alat yang efisien untuk mendeteksi serangan-serangan tersebut dengan tepat waktu.
Berdasarkan penelitian sebelumnya akan dilakukan penelitian menghitung efisiensi waktu pada algoritma akuisisi forensik digital.				

2.2 Forensik Digital

2.1.1 Definisi Forensik Digital

Forensik Digital terdiri dari dua kata, yaitu forensik dan digital. Menurut Kamus Besar Bahasa Indonesia (KBBI) digital memiliki arti yaitu berhubungan dengan angka-angka untuk sistem perhitungan tertentu; berhubungan dengan penomoran. Sedangkan kata forensik menurut KBBI adalah cabang ilmu kedokteran yang berhubungan dengan penerapan fakta medis pada masalah hukum. Adapun definisi keduanya adalah ilmu bedah yang berkaitan dengan penentuan identitas mayat seseorang yang ada kaitannya dengan kehakiman dan peradilan. Jika forensik digital diartikan per kata memang memiliki definisi yang berbeda yang berarti kata forensik berhubungan dengan dunia kedokteran. Namun penggunaan kata forensik mengalami perluasan makna. Forensik Digital memiliki konsep yang sama dengan dunia kedokteran namun memiliki objek yang berbeda.

Menurut Erhan Akbal dan Sengul Dogan forensik digital adalah suatu disiplin ilmu yang berhubungan dengan semua proses termasuk pengumpulan data digital dari barang bukti kejahatan, memeriksa, menganalisis, dan melaporkannya sesuai dengan metode dan standar tertentu (Akbal & Dogan, 2018).

Ada pendapat lain dari Mandeep Kaur, Navret Kaur, dan Suman Khurana mengenai forensik digital, yaitu penerapan teknik investigasi dan analisis untuk mengumpulkan dan menyimpan barang bukti dari perangkat komputasi yang sesuai dengan aturan tertentu untuk dipaparkan di pengadilan (Kaur, Kaur, & Khurana, 2016).

Secara umum pengertian forensik digital dapat disimpulkan sebagai suatu teknik yang yang digunakan untuk melakukan investigasi, pengumpulan bukti digital dari suatu perangkat digital untuk dianalisis dan digunakan sebagai bukti yang sah di pengadilan.

Berdasarkan Undang-Undang Republik Indonesia Nomor 11 Tahun 2008 tentang Informasi dan Transaksi Elektronik (UU ITE) informasi elektronik dan atau dokumen elektronik dan/atau hasil cetaknya merupakan alat bukti hukum yang sah (Republik Indonesia, 2015). Dari penjelasan undang-undang tersebut peran forensik digital sangat penting untuk mengungkap kebenaran dan membuktikan bentuk kejahatan siber sebagai bukti yang sah di pengadilan. Berdasarkan UU ITE pasal 32 ayat 1 Setiap Orang dengan sengaja dan tanpa hak atau melawan hukum dengan cara apa pun mengubah, menambah, mengurangi, melakukan transmisi, merusak, menghilangkan, menyembunyikan suatu Informasi Elektronik dan/atau Dokumen Elektronik milik Orang lain atau milik publik (Republik Indonesia, 2015) merupakan tindak pidana.

Dalam menangani kejahatan digital/siber diperlukan orang dengan keahlian khusus untuk menangani perkara tersebut karena sifat dari data digital yang rentan, mudah rusak, dan mudah dimodifikasi untuk menghapus jejak kejahatan.

2.1.2 Akuisisi Forensik Digital

Akusisi forensik digital merupakan tahapan pada proses investigasi forensik digital sering dikenal dengan *forensic imaging.* Dalam memperoleh data digital diperlukan proses akuisisi forensi. Akuisisi termasuk dalam proses investigasi dalam penanganan kasus kejahatan digital. Investigasi adalah proses yang mengembangkan suatu hipotesis dan mengujinya untuk memberikan jawaban tentang peristiwa termasuk pertanyaan mengenai apa penyebab terjadinya peristiwa, kapan peristiwa terjadi, dan di mana peristiwa terjadi (Fisher, Mandelbaum, & Walker, 2006). Investigasi dilakukan oleh seorang *investigator*. *Investigator* melakukan penyalinan data/informasi dari media penyimpanan digital. Data yang berhasil disalin harus bersifat valid (benar) karena akan digunakan sebagai barang bukti di pengadilan. Proses akuisisi juga harus berdasarkan hukum yang berlaku sebagai jaminan dilakukannya akuisisi forensik digital sebagai alat bukti yang legal dan diperbolehkan. Setelah media penyimpanan berhasil disalin, hasil salinan tersebut harus dipertahankan integritas keasliannya dengan memiliki nomor *hashing* dari bukti tersebut.

Akuisisi adalah proses pengumpulan barang bukti dari media elektronik/digital. Ada empat metode untuk melakukan akuisisi data; *disk-to-disk copy, disk-to-image file, logical disk*

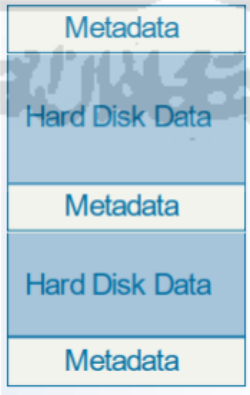
to disk file, dan data yang jarang diakses dari *file* atau folder. Media digital yang dapat dijadikan sebagai bukti digital adalah sistem komputer, media penyimpanan (*flashdisk, pen drive, hard drive*, atau CD-ROM), PDA, *smartcard*, sms, *cookies, source code, web, log chat*, dokumen, log *file*.

Proses *imaging* dapat dilakukan dengan menggunakan *software* forensik untuk melakukan *copy bit by bit* pada media penyimanan dengan tujuan mendapatkan hasil yang sama dengan media penyimpanan yang diakuisisi. Proses *copy bit by bit* digunakan agar memperoleh data seperti *bootloader*, *unallocated space*, *free space*, *slack space*, dan *hidden data* sehingga investigator memiliki data yang cukup untuk dianalisis. Proses ini juga bisa mengumpulkan *file-file* yang sudah terhapus. Perlu diingat untuk sebisa mungkin tidak melakukan analisis pada data aslinya karena dapat menyebabkan rusaknya barang bukti, sehingga proses akuisisi (*imaging*) diperlukan dalam proes investigasi forensik digital.

2.1.3 Image File Formats

Advance Forensic Format (AFF)

AFF merupakan format *image* hasil akuisisi yang dikembangkan oleh Simson Garfinkel yang bersifat *opensource*. AFF dapat dikembangkan misalnya dengan cara menambahkan fitur baru agar lebih kompatibel. Pengembangan tersebut memungkinkan untuk program lama supaya bisa membaca *file* AFF yang dibuat oleh program yang lebih baru, dan memungkinkan program AFF yang dibuat oleh program yang lebih baru untuk membaca *file* AFF yang dibuat oleh program versi sebelumnya. AFF menggunakan ekstensi .aff pada data hasil akuisisinya.

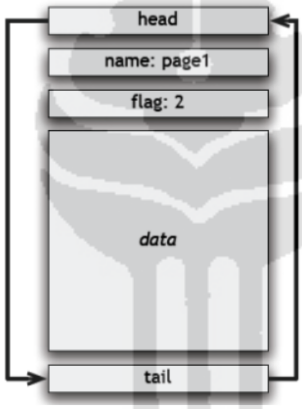


Gambar 2. 1 Format AFF

Sumber: (Basis Technology, 2007)

Struktur format penyimpanan pada AFF seperti pada gambar 2.1 terdiri dari dua bagian utama, yaitu hardisk data dan meta data. Hard disk data merupakan informasi yang terdapat pada media penyimpanan hard disk, sedangkan metadata adalah informasi tentang asal, struktur, dan karakteristik dari suatu perangkat data. Sifatnya lebih mendetail mengenau suatu data atau informasi.

AFF dapat menyimpan dengan hasil kompresi maupun tidak, Bersama dengan metadata yang dapat tersimpan secara bersamaan atau terpisah pada disk. *file* AFF dipartisi menjadi dua lapisan, *disk representation layer* dan *data-storage layer*. *Disk representation layer* menentukan nama segmen yang digunakan untuk menunjukkan semua informasi pada *disk image*. Tiap segmen AFF terdiri dari nama segmen, 32-bit *flag*, dan *payload data*. Nama dan *payload* data. Metadata menyimpan informasi mengenai *disk image*, dan segmen data yang disebut *pages*.



Gambar 2. 2 Struktur format AFF

Sumber: (Basis Technology, 2007)

Pada gambar 2.2 lapisan penyimpanan data pada AFF menyimpan segmen dalam bentuk biner (segmen disimpan berurutan dalam satu atau beberapa *file*). *Data pages* pada AFF dapat dikompres dengan zlib, ataupun tidak dikompresi sama sekali. Format ini mendukung *internal self-consistency checking* yang artinya menjadi alat yang dapat memulihkan *image* jika bagian pada *image* terdapan kerusakan atau *missing*. Format ini juga memiliki integritas untuk menjaga keaslian data dengan fungsi *hash* (MD5 atau SHA-1) dan tanda tangan digital.

Terdapat beberapa hal yang bisa didapat dengan menggunakan format AFF (Garfinkel, Malan, Dubec, Stevens, & Pham, 2006), yaitu:

a) Kemampuan untuk menyimpan *disk image* dengan atau tanpa kompresi.
b) Kemampuan untuk menyimpan berbagai ukuran *disk image.*

c) Kemampuan untuk menyimpan metadata bersamaan dengan *disk image* atau terpisah.
d) Kemampuan untuk menyimpan image pada *single file* atau dibagi menjadi beberapa *file* (*multiple file*).
e) Bisa dikembangkan
f) Lintas *platform, open source.*
g) Kemampuan untuk pengecekan konsistensi *image*. Bagian pada *image* bisa dipulihkan apabila terdapat bagian yang *corrupt* atau hilang.
h) Kemampuan untuk menjamin keaslian *file* barang bukti dengan fungsi *hash* (MD5 dan SHA-1) dan tanda tangan digital.

Raw Images

Hanya mengandung data dari sumber *disk.* Tidak memiliki meta data sehingga tidak menyimpan informasi tentang nomor serial pada drive, nama *investigator*, dan informasi tentang dilakukannya akuisisi. Berdasarkan gambar 2.3 metadata dapat diambil namun berada pada *file* yang berbeda.

Gambar 2. 3 Raw *Image* mengandung meta data
Sumber: (Vandeven, 2014)

Biasanya format ini digunakan pada sistem operasi Linux. Proses *imaging* dilakukan dengan menyalin data secara keseluruhan dengan menerapkan metode *sector-by-sector*. *Raw image* dapat dibuat dengan *tools* akuisisi yang beraneka macam dan biasanya menggunakan ekstensi .dd, .raw, .img. Pada *raw image* tidak dilakukan kompresi, sehingga hasil dari *file* akuisisi tersebut bisa berukuran besar, bahkan ketika drive hanya mengandung sedikit data. Kekurangan pada *file* tipe *raw image* yaitu tidak mengandung informasi mengenai meta data melainkan data asli dari hasil akuisisi sumber media penyimpanan.

Adapun beberapa kelebihan menggunakan *file* tipe Raw (Abdulghani Ali Ahmed Wan Nurulsafawati Wan Manan, 2015):

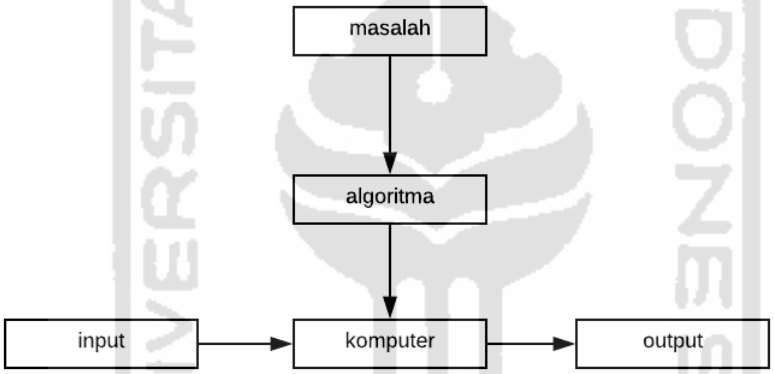
a) Cepat dalam transfer data.
b) Bisa dibaca hampir pada setiap *tool digital forensic*.

c) Dapat menghindari kesalahan kecil (*minor error*) ketika membaca driver yang akan diakuisisi.

2.3 Algoritma

2.2.1 Definisi Algoritma

Algoritma ditemukan oleh Al-Khawarizmi yang berasal dari Persia. Algoritma biasa dikenal dengan langkah-langkah atau prosedur yang harus dilakukan untuk menyelesaikan suatu masalah. Algoritma dapat diimplementasikan menggunakan program komputer. Suatu algoritma adalah urutan instruksi untuk menyelesaikan suatu masalah, yaitu untuk mendapatkan output yang sesuai untuk setiap input yang diberikan dalam jumlah waktu yang terbatas (Greenfield, Goodman, & Hedetniemi, 1979).



Gambar 2. 4 notasi suatu algoritma

Sumber: (Greenfield et al., 1979)

Pada gambar 2.4 tersebut dapat dipahami bahwa algoritma berperan penting untuk mengatasi suatu masalah. Dengan dimasukkannya nilai sebagai input yang kemudian akan dikomputasi dengan algoritma tertentu oleh komputer untuk menghasilkan nilai atau hasil atau output yang diinginkan. Sebelum adanya komputer implementasi dari algoritma dilakukan oleh manusia. Dengan berkembangnya teknologi kini algoritma bisa diimplementasikan dengan komputer yang bisa berisi perhitungan numerik, logika, atau kondisional.

2.2.2 Analisis Algoritma

Menurut KBBI arti kata analisis adalah penyelidikan terhadap suatu peristiwa untuk mengetahui keadaan yang sebenarnya. Untuk mengetahui kualitas baik atau tidaknya suatu algoritma maka diperlukan analisis untuk mendapatkan efisiensi (tidak membuang *source*).

Terdapat dua macam jenis efisiensi algoritma yaitu, *time efficiency* (efisiensi waktu) dan *space efficiency* (efisiensi ruang).

Time efficiency atau efisiensi waktu atau kompleksitas waktu merupakan seberapa cepat suatu algoritma dieksekusi. *Time efficiencsy* diketahui dengan menghitung berapa kali operasi dasar pada algoritma dieksekusi. Sedangkan *space efficiency* atau efisiensi ruang atau kompleksitas ruang adalah seberapa besar memori yang digunakan saat algoritma berjalan. Hal tersebut dapat diketahui. Tujuan dilakukannya analisis algoritma yang meliputi menghitung efisiensi waktu dan ruang adalah untuk melihat dan melakukan evaluasi terhadap algoritma yang sudah dibuat apakah sudah efisien. Jika belum makaa hasil analisis bisa dijadikan sebagai pedoman untuk melakukan perbaikan atau mendesain ulang algoritma yang sudah ada. Namun perlu dipahami bahwa belum ada algoritma yang benar-benar efisien untuk menyelesaikan suatu masalah (Himawan, 2015).

Menurut Anany Levitin terdapa 3 hal dalam analisis algoritma, yaitu;

a) Worst-case efficiency

   Kondisi ketika nilai input dari n merupakan nilai yang diberikan ketika algoritmanya berjalan paling lama di antara semua nilai input yang mungkin. Misalnya tidak ditemukan nilai yang sesuai dengan nilai input yang diberikan pada urutan pertama atau nilai input baru ditemukan pada urutan paling akhir sehingga memerlukan waktu yang panjang. Cara untuk mengetahuinya adalah menganalisis algoritma untuk melihat input jenis apa yang menghasilkan nila terbesar pada operasi dasar. Mengetahui kondisi ini merupakan hal yang penting karena dengan memperoleh kondisi *worst-case* dapat menjadi batasan dengan kondisi yang lain.

b) The Best-case efficiency

   Kondisi ketika nilai input dari n saat nilai tersebut menjadikan algoritma berjalan cepat dari semua kemungkinan nilai input yang ada. Misalnya ketika nilai input dari n ditemukan pada urutan pertama maka algoritma tidak perlu melakukan operasi yang panjang untuk menemukannya. Cara memperoleh *best-case efficiency* yaitu dengan menganalisis algoritma kemudian menentukan nilai input n yang menjadikan operasinya menjadi cepat berlangsung. *The best-case efficiency* bukan berarti memberi input dengan nilai terkecil, namun nilai n yang memungkinkan algoritma berjalan cepat (Greenfield et al., 1979).

c) Average-case efficiency

Untuk mengetahui kondisi ini diperlukan untuk membuat beberapa asumsi tentang kemungkinan nilai input yang ada. Namun dengan asumsi tersebut menjadikannya sulit untuk untuk diuji kebenarannya. Mencari *average-case efficiency* lebih sulit dibandingkan dengan *best-case efficiency* dan *worst-case efficiency.*

2.2.3 Big-O

Untuk menghitung kompleksitas algoritma salah satunya dengan notasi *Big-O*. Notasi *Big-O* merupakan simbol yang digunakan dalam ilmu kompleksitas, ilmu komputer, dan matematika untuk menggambarkan fungsi asimptotik (Black, 2007). Definisi *Big-O* adalah sebagai berikut

$$T(n) \in O(g(n))$$

T(n) berorde di atas atau paling besar dari *g(n)*. Bilai terdapat nilai konstan positif C dan n_0, maka $T(n) \leq cg(n)$ untuk semua $n \geq n_0$.

Dengan notasi ini akan ditunjukkan seberapa cepat suatu fungsi tersebut naik atau turun. Pada tabel 2.2 terdapat beberapa fungsi yang biasanya digunakan dalam notasi *Big-O*.

Tabel 2. 2 tabel kompleksitas notasi *Big-O*

Notasi	Nama
$O(1)$	Konstan
$O(\log\ n)$	Logaritmik
$O(\log(n))^c$	Polylogaritmik
$O(n)$	Linear
$O(n^2)$	Kuadratik
$O(n^c)$	Polinomial
$O(c^n)$	exponensial

.

2.4 Flowchart

Flowchart atau diagram alur merupakan pemaparan urutan proses dalam bentuk bagan yang bertujuan untuk mempermudah dalam melihat atau menganalisis suatu proses lebih mendetail. Adapun penjelasan lain mengenai *flowchart* yaitu penggambaran dengan menggunakan grafik dari beberapa langkah dan beberapa urutan dari suatu program (Ridlo, 2017). Seperti yang dikatakan oleh Dr. W. Edwards Deming, "*draw a flowchart for whatever*

you do" yang artinya "menggambar sebuah diagram alur untuk apa saja yang kamu lakukan" (Ridlo, 2017).

2.5 Pseudocode

Pseudocode digunakan untuk menuliskan atau mendeskripsikan suatu algoritma dengan menggunakan bahasa yang sederhana dan mudah dipahami. Terdapat fakta mengenai *pseudocode* bahwa tidak ada pedoman yang baku atau *standard* dalam penulisannya. Hal tersebut memiali arti bahwa setiap pribadi yang membuat *pseudocode* memiliki tata bahasa dan kosa kata tersendiri kecuali pribadi tersebut bekerja pada suatu organisasi atau perusahaan yang sudah menetapkan *standard* dalam penulisan *pseudocode* (Bennett, 2016).

2.6 Bahasa C

Bahasa C merupakan bahasa tingkat tinggi yang pertama kali dikembangkan oleh Dennis Ritchie dan bekerja dengan sistem operasi UNIX. bahasa C mendekati tingkat bahasa *assembly* yang biasanya digunakan untuk melakukan pemrograman sistem komputer dan jaringan komputer. Walau bahasa ini dirancang untuk memprogram suatu sistem komputer dan jaringan komputer, namun dengan adanya perkembangan dalam penggunaannya, bahasa ini bisa digunakan untuk mengembangkan aplikasi. *file* pemrograman bahasa C disimpan dengan menggunakan ekstensi *file* .c (dot si). Pemrograman ini baik untuk menulis program pada sistem operasi komputer karena ekspresi penulisannya yang sederhana dalam berbagai penerapan (Souli, 2007).

BAB III
METODOLOGI PENELITIAN

Metodologi penelitian adalah langkah-langkah yang harus ditempuh untuk kepentingan penelitian. Langkah-langkah tersebut dibuat supaya menjawab masalah yang muncul secara sistematis dan logis sehingga dilakukan proses ilmiah untuk menyelesaikan masalah yang muncul.

3.1 Identifikasi Masalah

Pada bagian ini kita mencari fakta mengenai kejahatan siber di Indonesia. Proses yang dilakukan dengan meriset data dari berbagai sumber untuk mengetahui seberapa besar kejahatan siber yang muncul dan memungkinkan bertambahnya pula barang bukti yang harus ditangani.

3.2 Pengumpulan Data

Pada tahap ini diperbanyak referensi mengenai teori-teori tentang forensik digital, analisis algoritma, format AFF, dan juga format Raw yang berasal dari jurnal, *website*, artikel, *paper*, *powerpoint*, ataupun skripsi penelitian sebelumnya. Pencarian literatur menggunakan internet.

Dalam tahap ini peneliti juga mencari data mengenai *source-code* tentang aplikasi maupun *master-code* yang berhubungan dengan penelitian. *Source-code* didapat melalui situ *github.com* sehinga mendapatkan 2 *source-code* yang menggunakan bahasa C, yaitu:

- Aff-master (berisi pemrograman akusisi dengan format AFF versi 1)
- dcfldd-1.3.4-1.tar (merupakan aplikasi untuk melakukan akuisisi dengan format Raw/dd)

Penelitian berfokus pada *file* tempat berlangsungnya proses akuisisi dilakukan. Pada format Raw digunakan *file copy.c* yang diambil dari aplikasi dcfldd-1.3.4-1.tar yang diunduh pada situs *dcfldd.sourceforge.net*. Kemudian pada format AFF1 digunakan *file copy.c* yang diambil dari projek aff-master.zip yang diunduh pada situs *github.com.* Alasan mengambil *file copy.c* pada setiap format karena algoritma mengenai proses akuisisi forensik digital terdapat pada *file* tersebut.

3.3 Membuat Diagram Alur

Setelah mendapatkan *source-code* tentang masing-masing format penelitian dilanjutkan dengan membuat diagram alur untuk mempermudah dalam menganalisis algoritma. Pembuatan diagram alur dilakukan dengan membaca kode per baris. *file copy.c* pada setiap format baik Raw mau pun AFF dianalisis proses yang terjadi, kemudian digambarkan ke dalam bentuk diagram alur agar lebih terlihat proses apa saja atau operasi apa saja yang terjadi pada proses akuisisi forensik digital. Hasil dari diagram alur digunakan untuk mempermudah melihat jalannya proses akuisisi forensik digital.

3.4 Membuat psudocode

Tujuan dibuatnya pseudocode agar suatu algoritma dapat dipahami dengan mudah tanpa terikat dengan bahasa pemrograman tertentu. Pseudocode dapat memudahkan untuk melakukan analisis algoritma. Pseudocode biasanya menggunakan bahasa yang baku. Pada gambar 3.1 terdapat format penulisan *pseudocode* yang biasanya digunakan.

Program: (berisi nama program)
Kamus: (berisikan deklarasi variabel yang digunakan)
Deskripsi: (menunjukkan jalannya program, berisi instruksi-instruksi)

Gambar 3. 1 format penulisan *pseudocode*

3.5 Menghitung Efisiensi

Sebelum menghitung efisiensi waktu pada algoritma diperlukan untuk menganalisis proses dan memperhatikan operasi apa saja yang terjadi ketika algoritma berjalan. Kemudian dilanjutkan dengan menghitung nilai waktu atau *time* yang diperlukan pada setiap *cost* yang dibutuhkan algoritma. Nilai *cost* dan *time* mulai dihitung dari baris pertama kode program

dimulai hingga akhir. Pencatatan tersebut dimasukkan ke dalam tabel untuk mempermudah analisis.

Kemudian nilai waktu tersebut akan dijumlahkan menjadi nilai waktu total. Hasil dari nilai total tersebut akan membentuk persamaan aljabar. Efisiensi algoritma dihitung dengan menggunakan notasi *Big-O* berdasarkan hasil persamaan yang didapatkan. Persamaan 3.1 merupakan cara untuk menghitung waktu total pada algoritma.

$$Waktu\ Total = \sum_{n}^{i} Cn \qquad (3.1)$$

3.5.1 Raw

Pada tabel 3.1 format Raw terdapat *cost* sebanyak n dimulai dari C_1 hingga C_n diikuti dengan nilai *time* yang dimulai dari $t_1$sampai t_n. Kemudian nilai *time* yang tercatat akan dipisah masing-masing berdasarkan jenis waktunya.

Tabel 3. 1 Tabel *Cost-time* pada Raw

Cost	Time
C_1	t_1
C_2	t_2
…	…
C_n	t_n

3.5.2 AFF

Kemudian pada format AFF terdapat *cost* sebanyak n dimulai dari C_1 hingga C_n diikuti dengan nilai *time* yang dimulai dari $t_1$sampai t_n seperti pada tabel 3.2. Sama seperti format Raw, nilai yang didapatkan kemudian dipisah berdasarkan jenis waktunya yaitu dari $t_1$sampai t_n.

Tabel 3. 2 Tabel *cost-time* pada AFF

Cost	Time
C_1	t_1
C_2	t_2
…	…
C_n	t_n

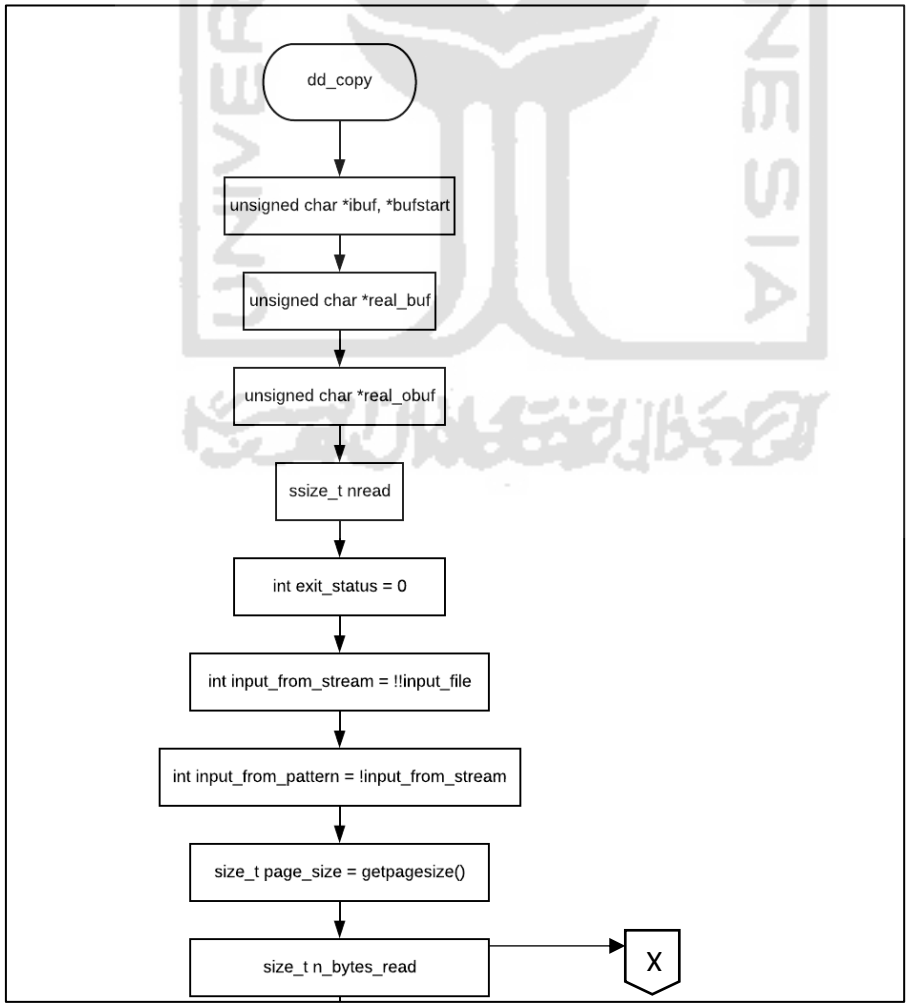
BAB IV
HASIL DAN PEMBAHASAN

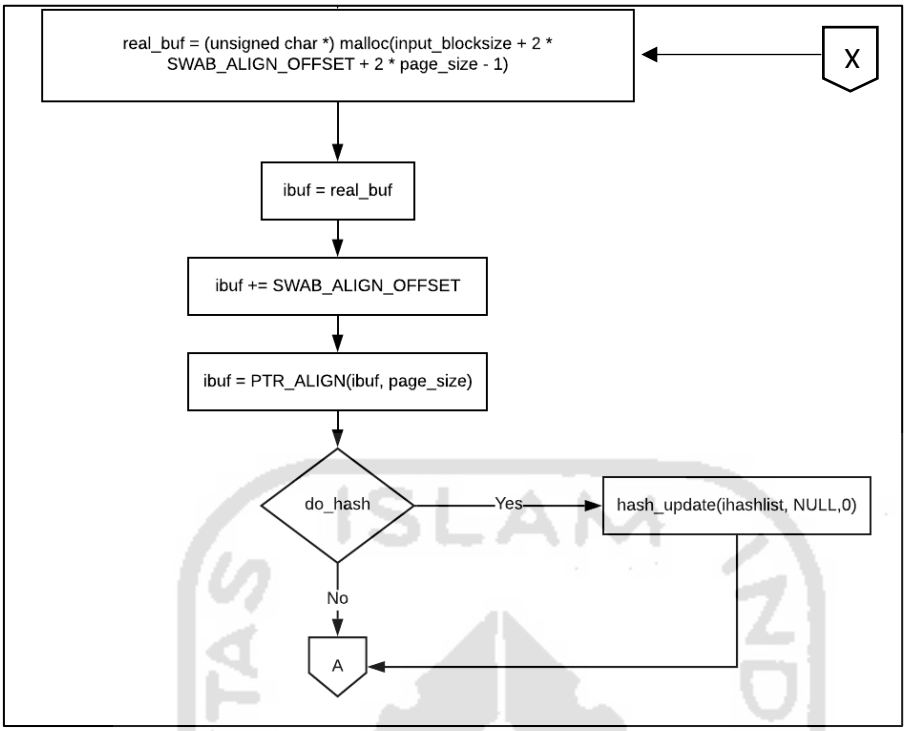
Analisis algoritma dilakukan dengan membuat *flowchart* diikuti dengan *pseudocode* untuk memudahkan dalam melihat proses di dalam algoritma. Kemudian setelah membuat diagram penelitian dilanjutkan dengan menghitung kompleksitas waktu.

4.1 Flowchart

Langkah pertama untuk melakukan analisis dengan membuat diagram alur berdasarkan algoritma yang sudah didapat. Algortima tersebut berbentuk *file* dengan ekstensi .c (menggunakan bahasa C) yang berisikan program untuk melakukan proses penyalinan data. Diagram alur dibuat dengan cara melihat proses pada program baris-per-baris.

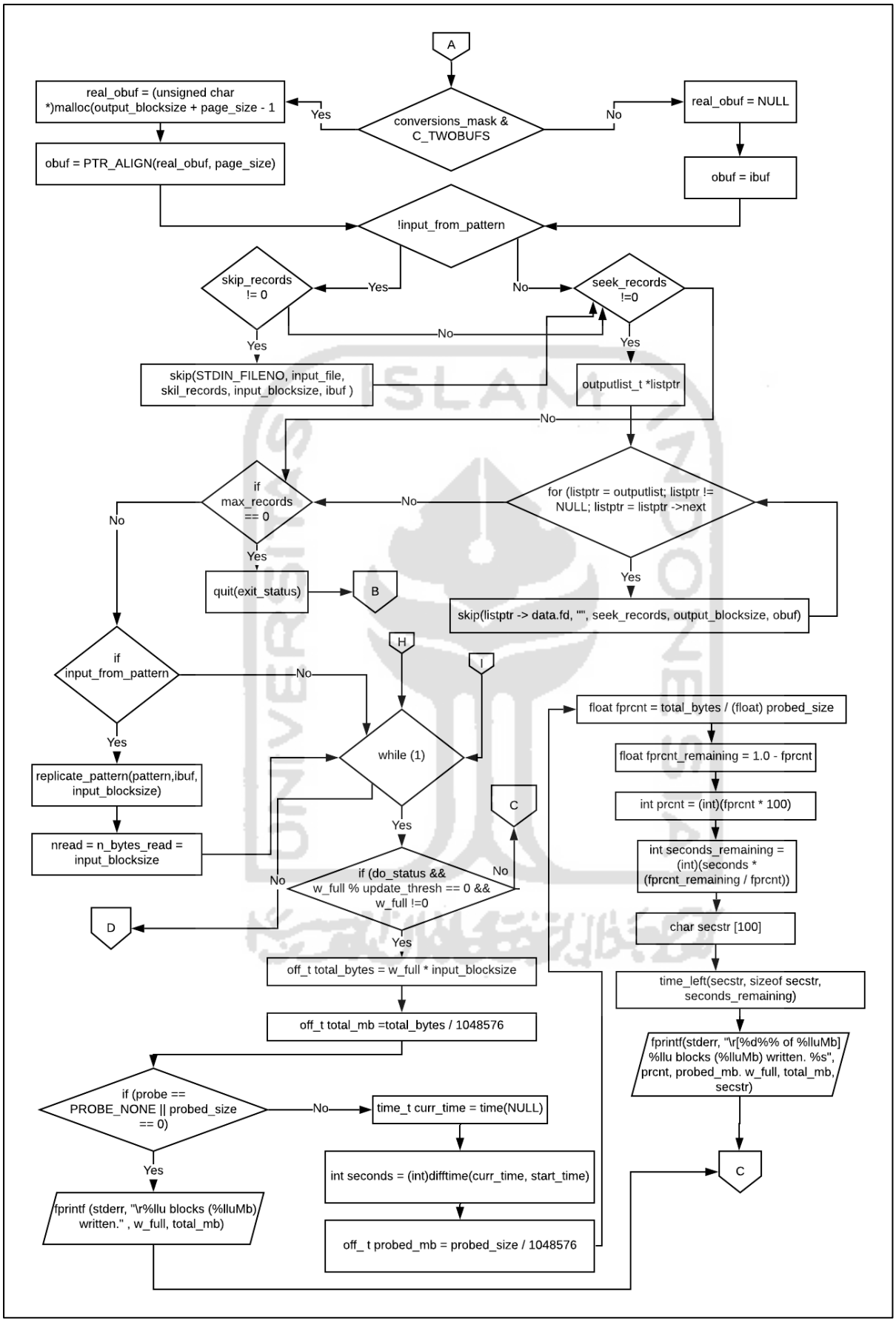
Format Raw



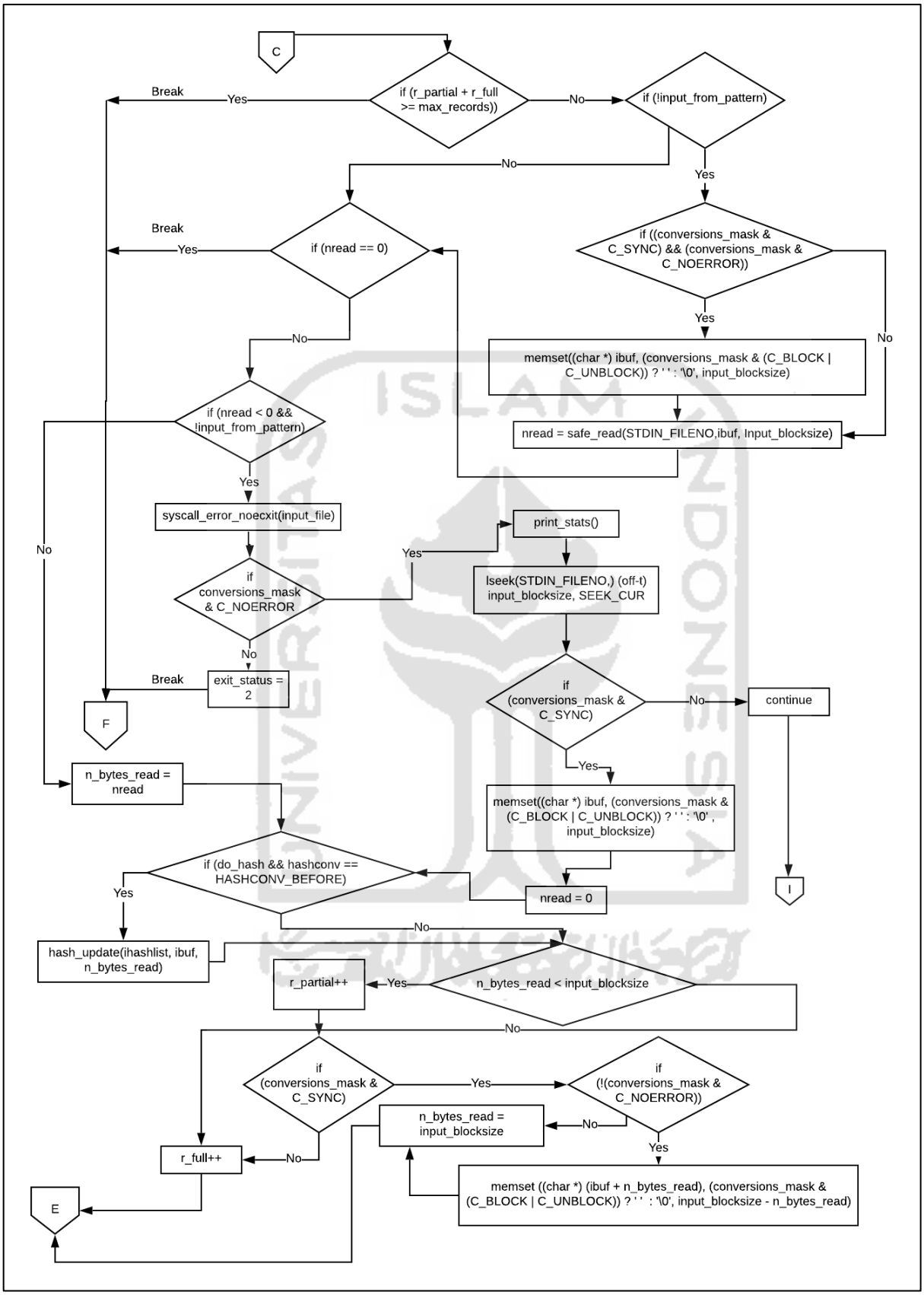


Gambar 4. 1 Diagram alur format Raw bagian 1

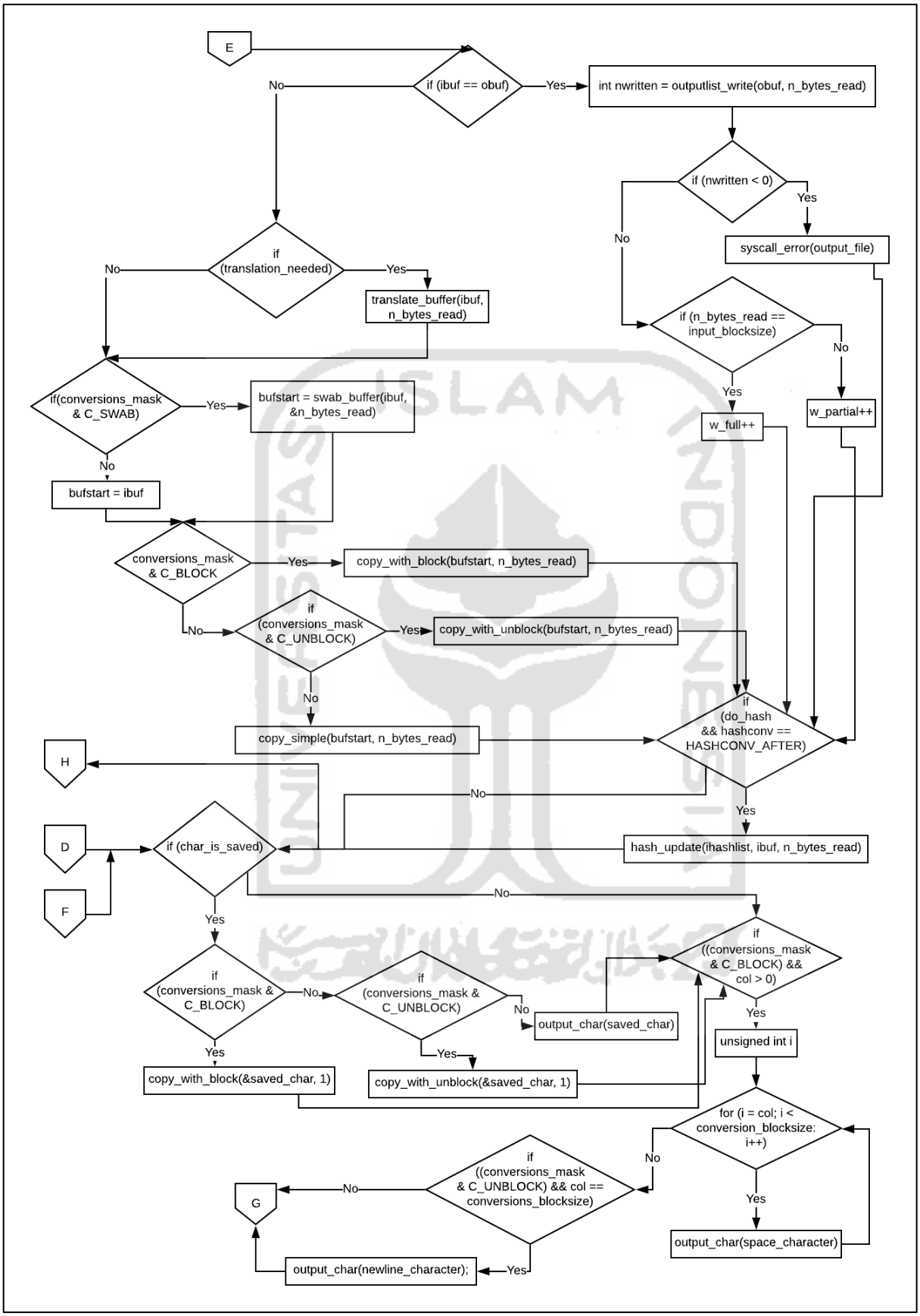
Pada diagram alur tersebut proses dilakukan dengan memanggil fungsi *dd_copy*. Kemudian setelah fungsi dipanggil diikuti dengan pendeklarasian variabel. Pada diagram alur tersebut tidak mengandung semua variabel yang digunakan dalam keseluruhan proses. Karena pada *source-code* terbagi dari beberapa *file* lain yang berisi *library* maupun fungsi pendukung diikuti dengan pendeklarasian variabel lainnya. Variabel *ibuf* dan *obuf* merupakan kepanjangan dari *input buffer* dan *output buffer*. *Buffer* sangat penting dalam proses pertukaran data antar perangkat ke perangkat, maupun dari perangkat ke aplikasi. Kemudian pada diagram di atas terdapat percabangan *do_hash* yang berarti apakah user akan melakukan proses *hashing* pada saat akuisisi atau tidak. Jika melakukan *hashing* maka proses dilakukan dengan memanggil fungsi *hash_update* untuk melakukan pengecekan pada data.



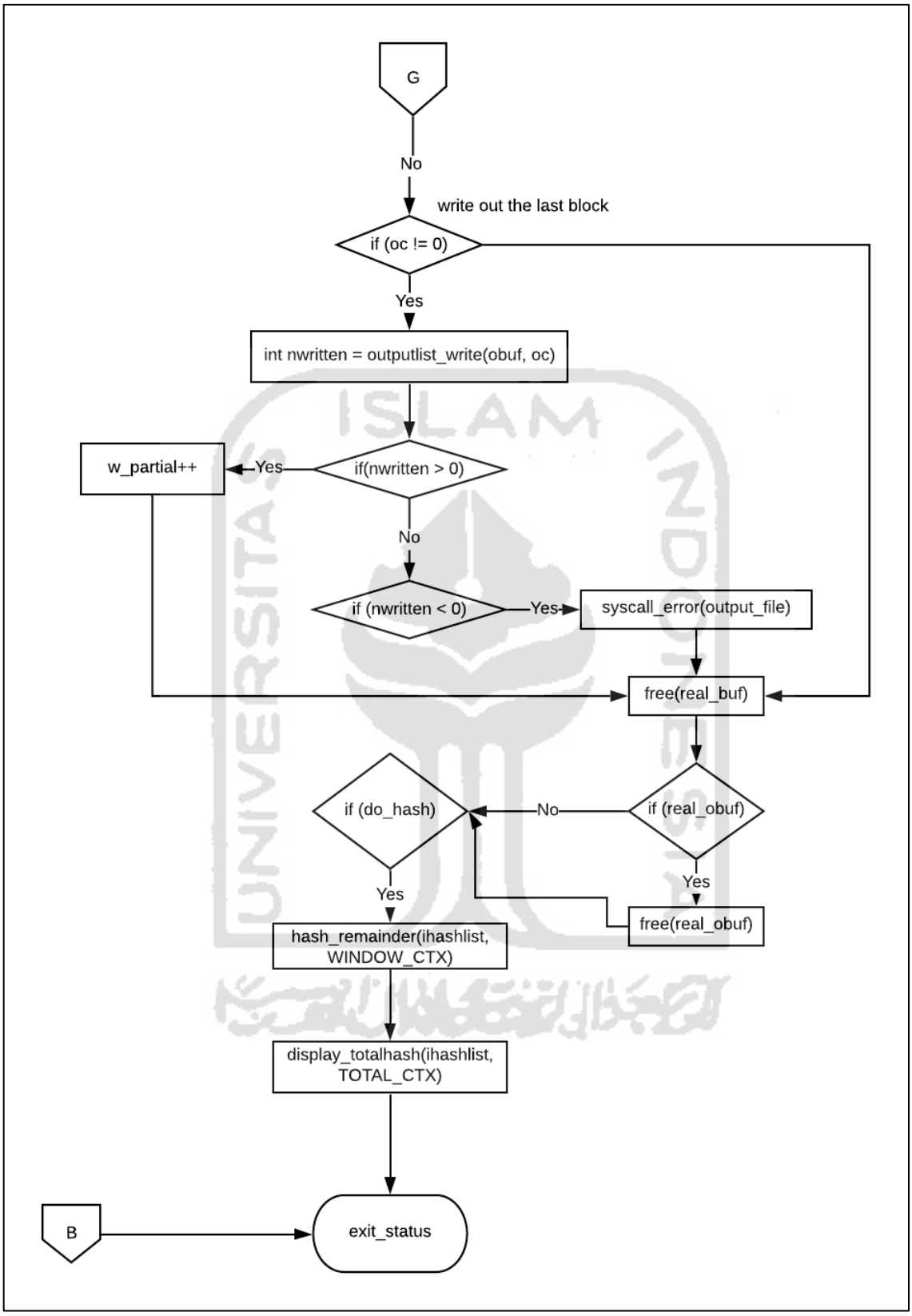
Gambar 4. 2 Diagram alur format Raw bagian 2



Gambar 4. 3 Diagram alur format Raw bagian 3



Gambar 4. 4 Diagram alur format Raw bagian 4



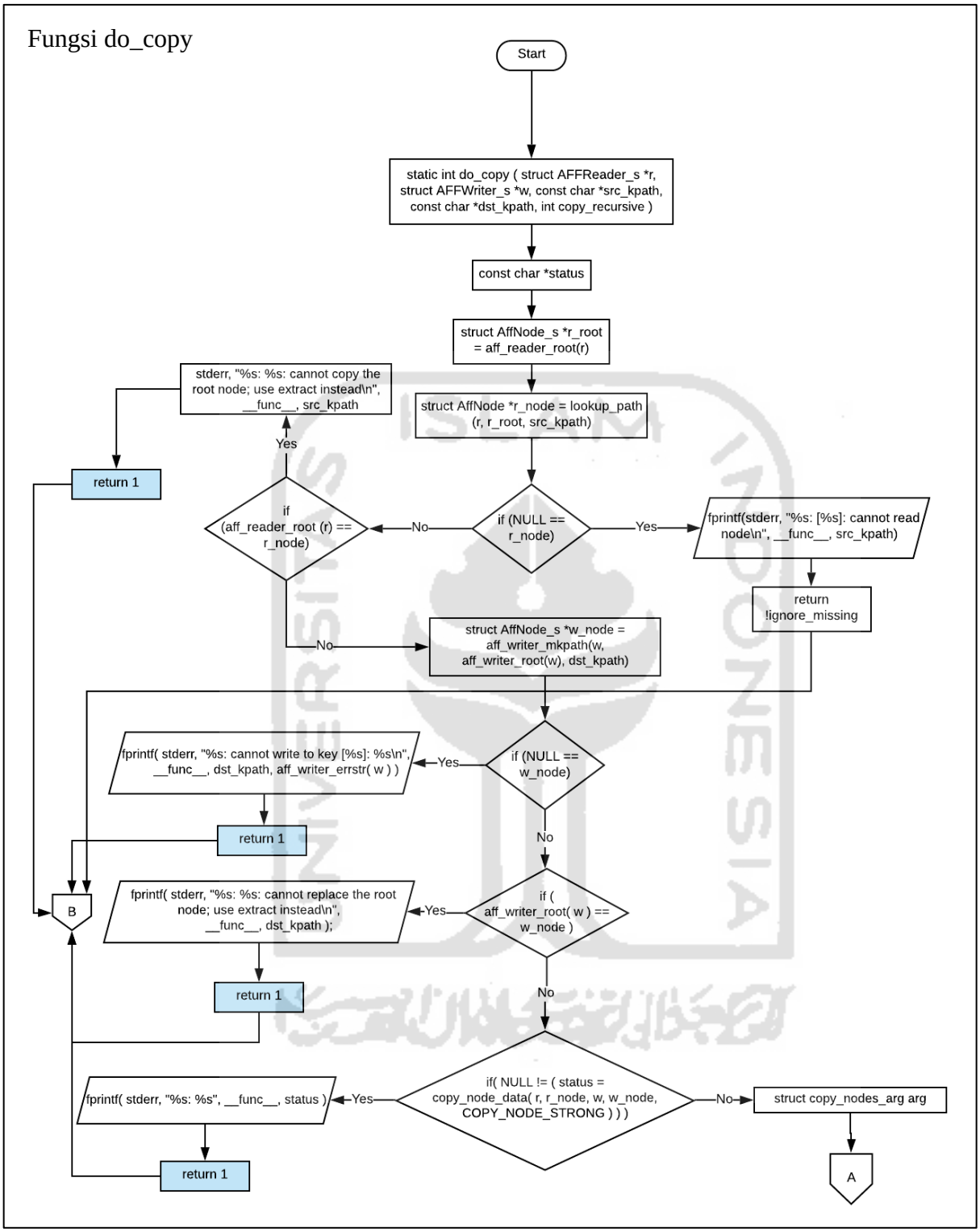
Gambar 4. 5 Diagram alur format Raw bagian 5

Pada Gambar 4.5 terdapat percabangan *do_hash* lagi. Di sini alur berjalan sebagai pengingat kembali pada dimulainya proses akuisisi. Proses *hashing* dilakukan di tahap akhir sebagai bentuk integritas pada data. Nilai *hashing* berupa nilai unik yang dibuat secara acak. Jika pada awal proses akuisisi dilakukan *hashing* maka proses selanjutnya adalah memanggil fungsi *hash_remainder* untuk men-*generate* nilai *hash*. Kemudian nilai *hash* akan ditampilkan pada *user* dengan memanggil fungsi *display_hash*. *Hashing* yang dilakukan pada format ini dapat berupa MD5 ataupun SHA1. Kemudian nilai *hash* yang didapat disimpan dalam file terpisah dalam ekstensi *file* .txt.

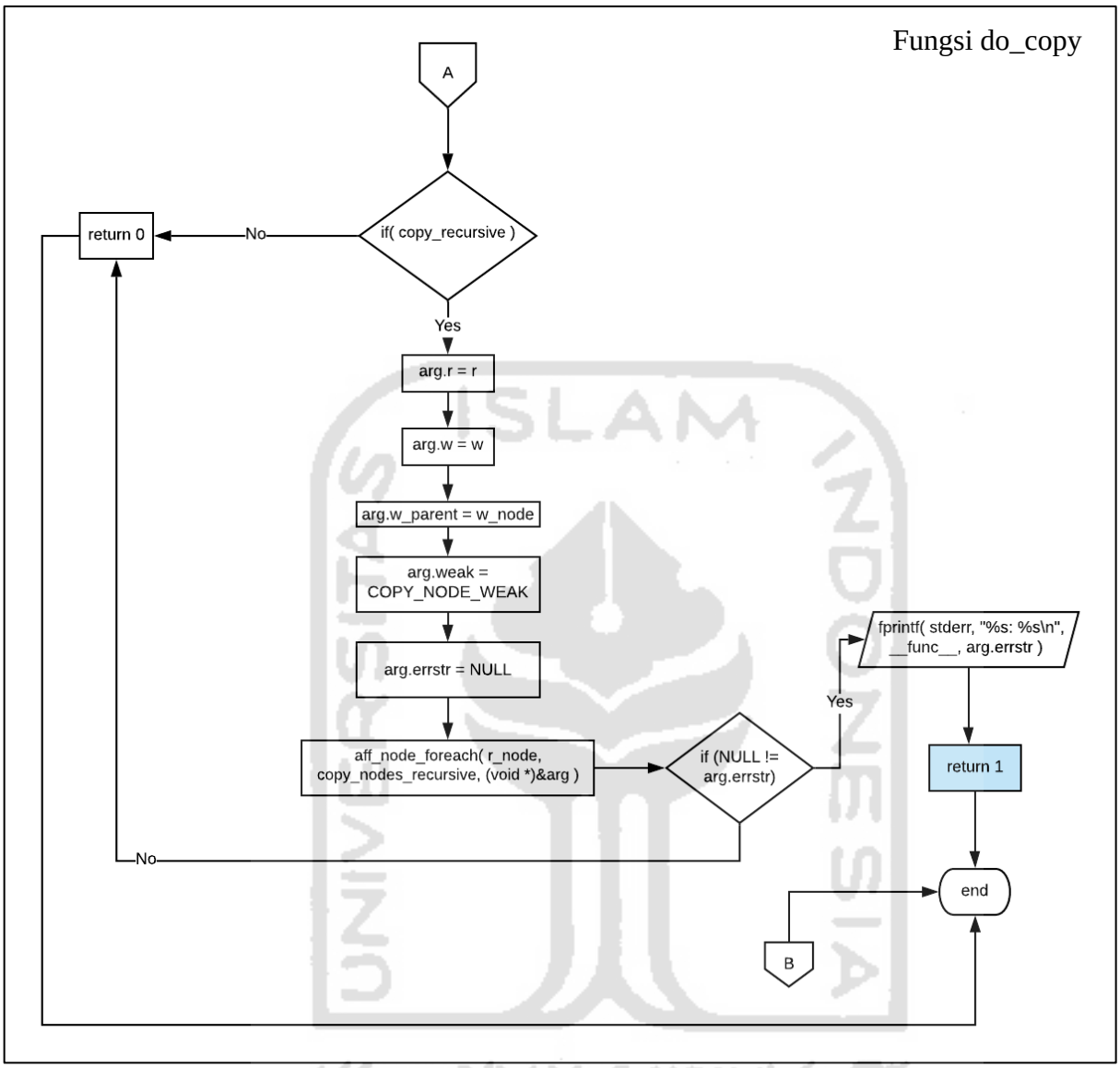
Jika pada proses akuisisi tidak dilakukan proses *hashing* pada data, maka proses akan berakhir dengan menunjukkan status bahwa proses telah selesai. Maka pada *user* akan ditampilkan mengenai waktu yang dibutuhkan selama proses akuisisi berlangsung.

Format AFF (Advance forensik Format)

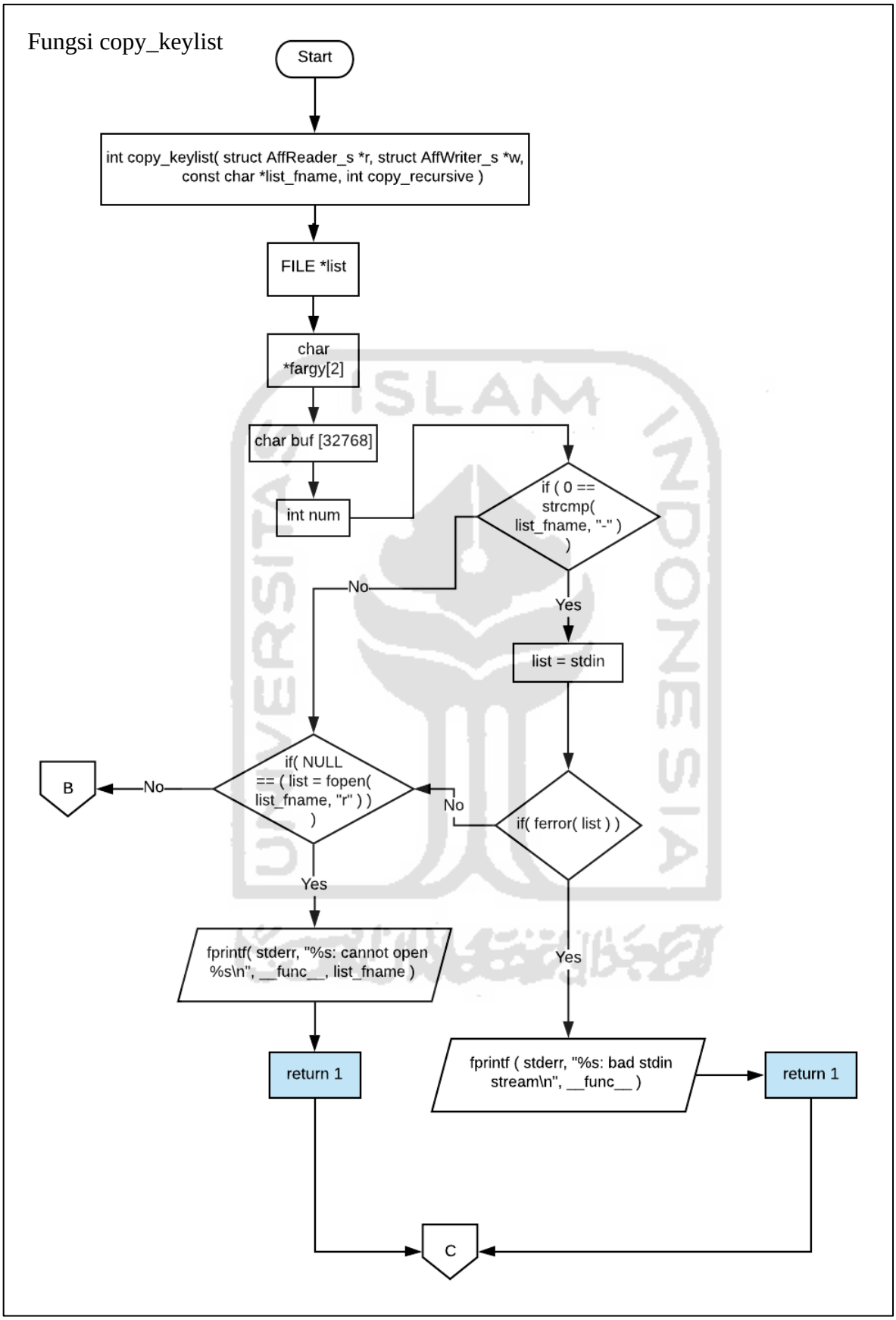
Pada AFF terdapat 3 fungsi yaitu do_copy, copy_keylist, dan x_copy oleh karena itu terdapat 3 jenis *flowchart*. Pembuatan *flowchart* dilakukan dengan membaca kode dari awal hingga akhir. Pembuatan *flowchart* akan mempermudah dalam memahami arah alur atau eksekusi yang terjadi pada proses akuisisi forensik digital.



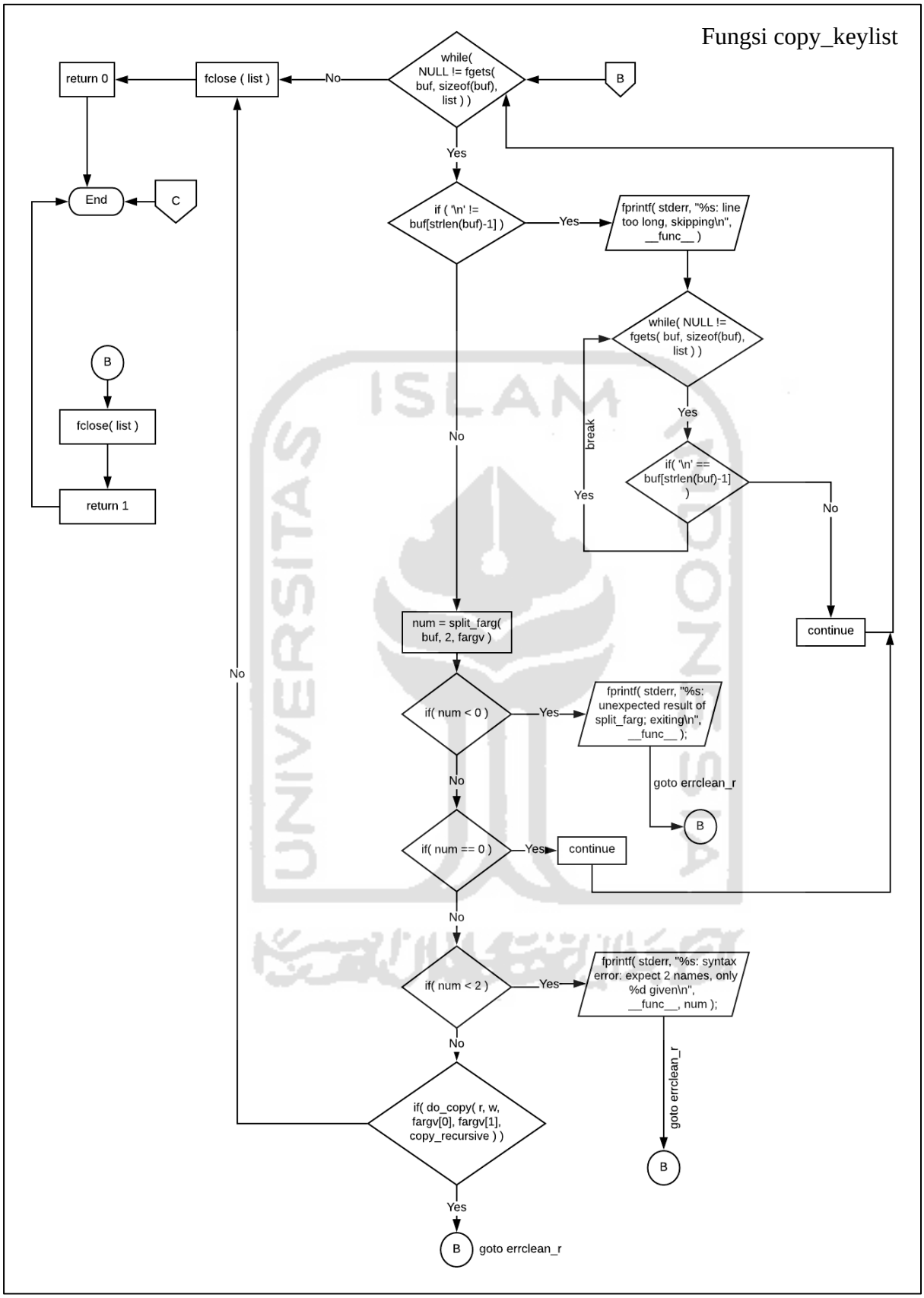
Gambar 4. 6 Diagram alur fungsi do_copy bagian 1



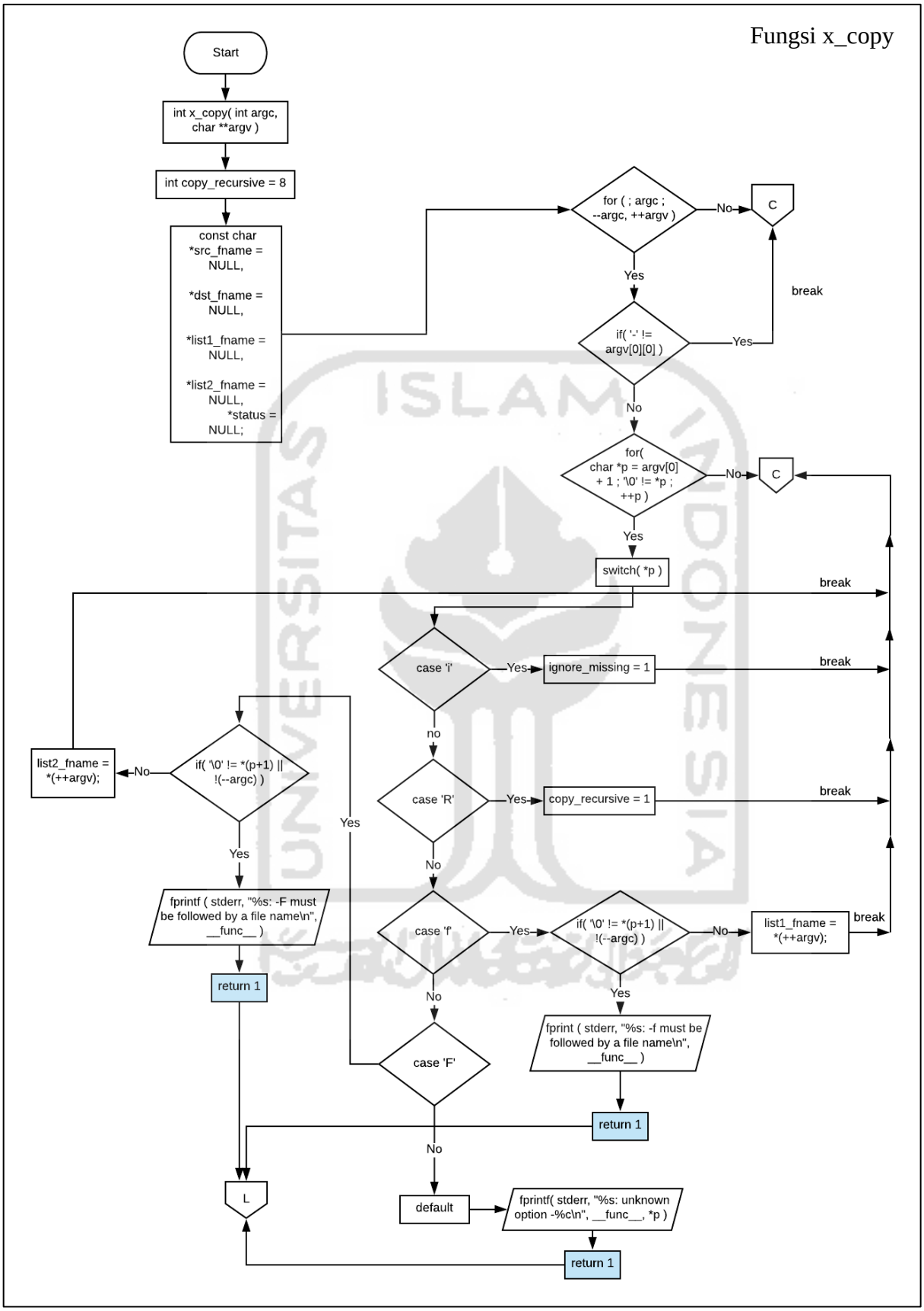
Gambar 4. 7 Diagram alur fungsi do_copy bagian 2



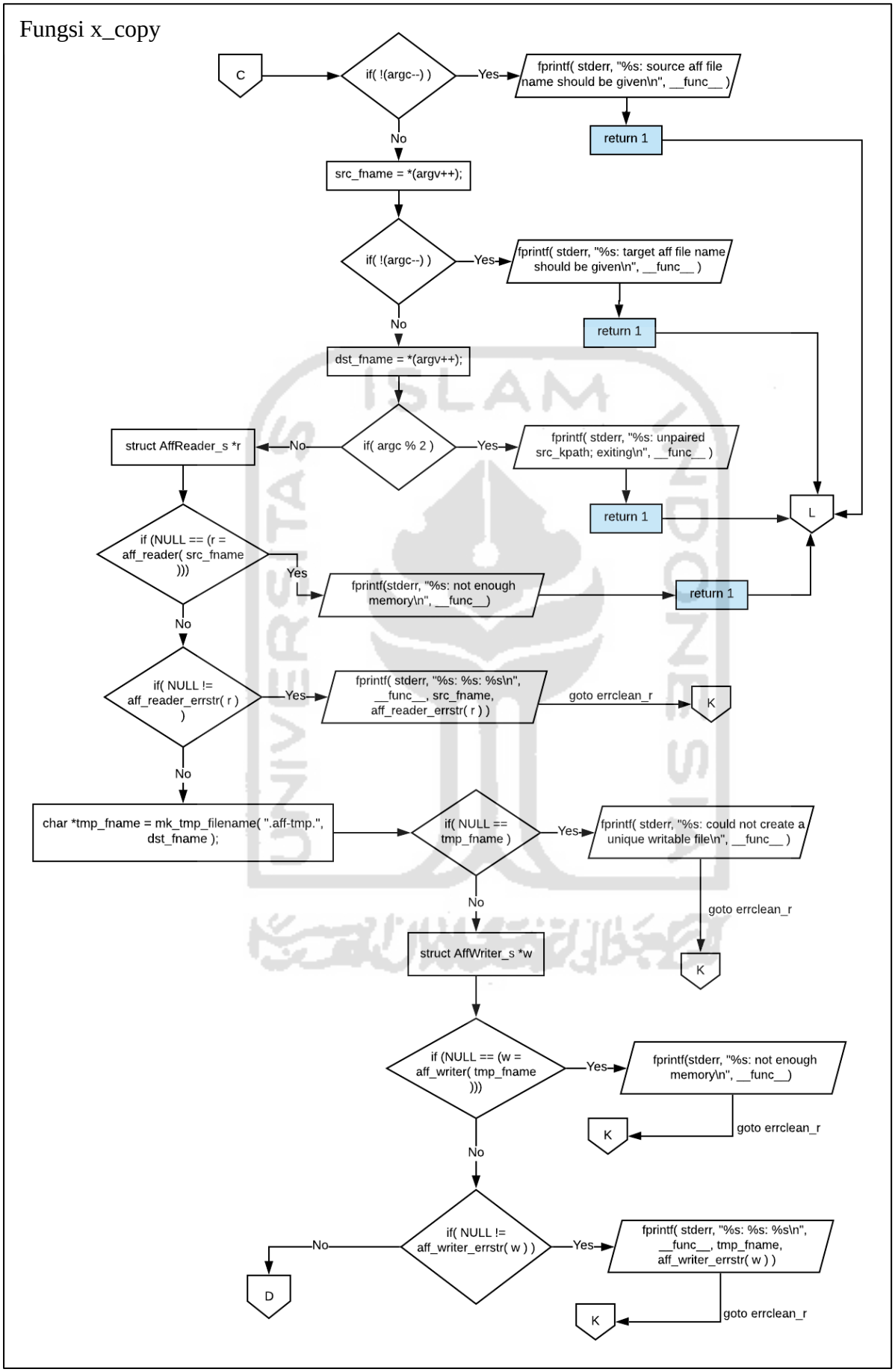
Gambar 4. 8 Diagram alur fungsi copy_keylist bagian 1



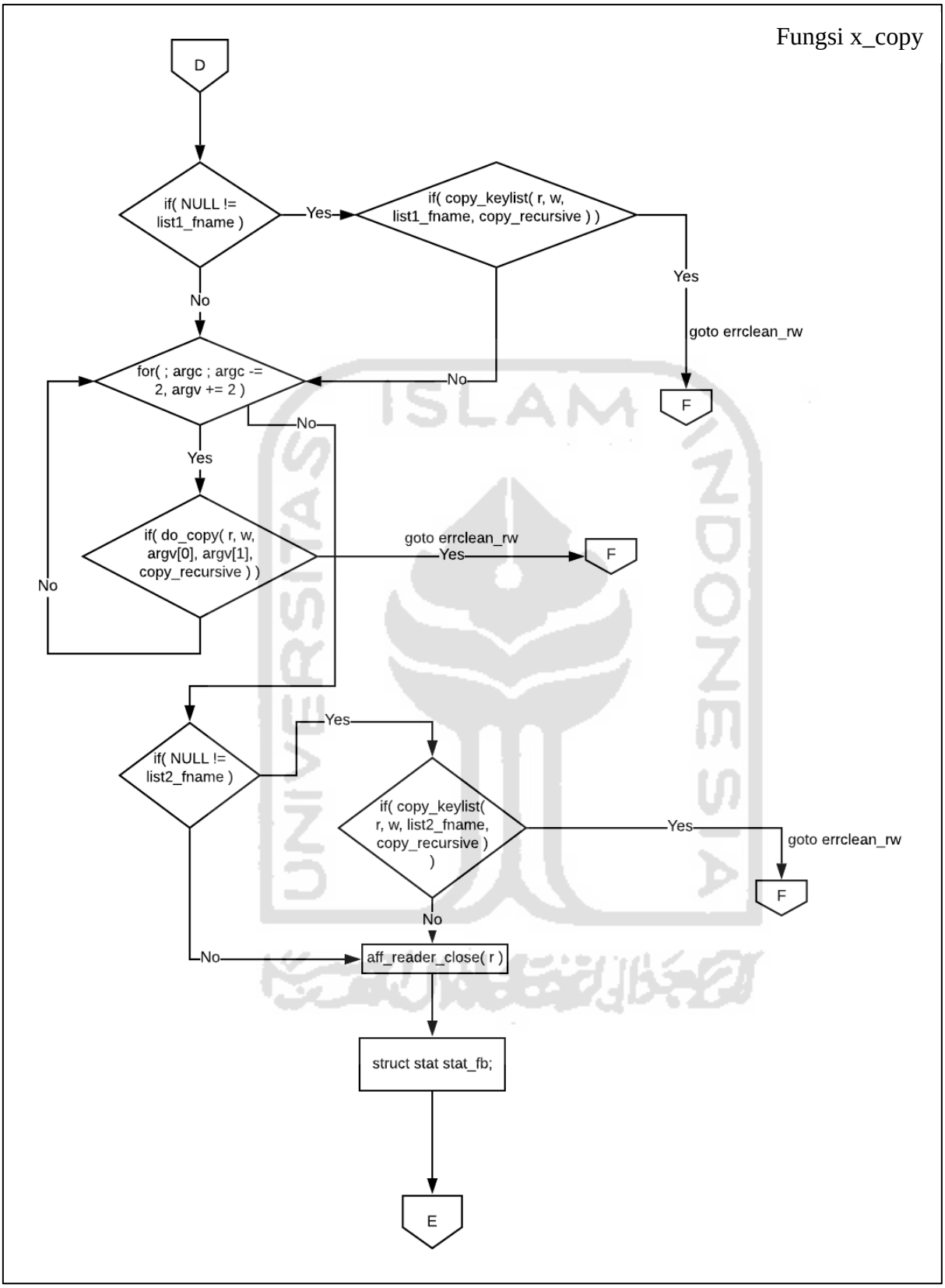
Gambar 4. 9 Diagram alur fungsi copy_keylist bagian 2



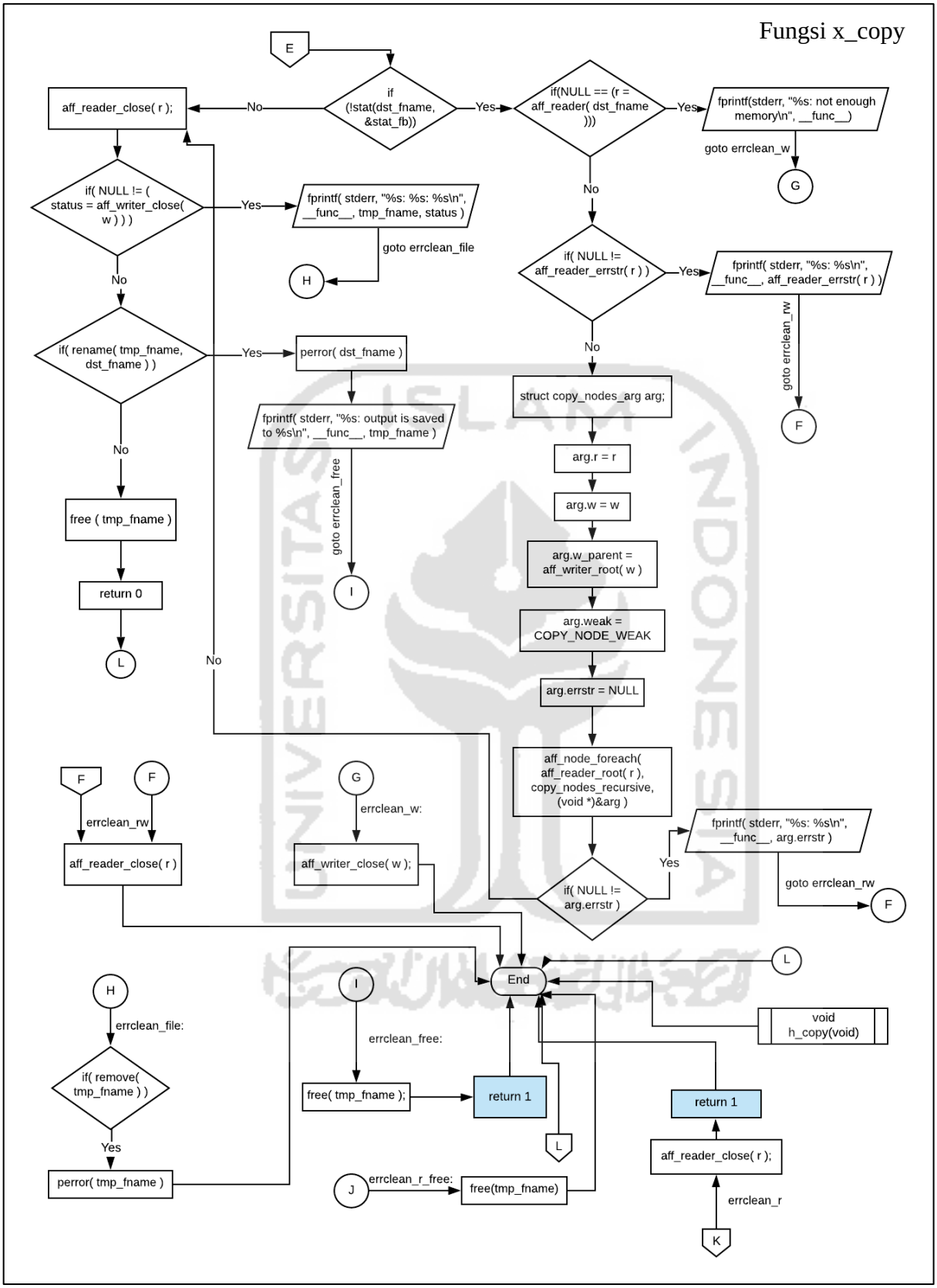
Gambar 4. 10 Diagram alur fungsi x_copy bagian 1



Gambar 4. 11 Diagram alur fungsi x_copy bagian 2



Gambar 4. 12 Diagram alur fungsi x_copy bagian 3



Gambar 4. 13 Diagram alur fungsi x_copy bagian 5

Pada format AFF1 proses yang terjadi lebih kompleks dibandingkan dengan format Raw. Pada proses akuisisi dengan tipe *file* AFF1 dimulai dengan mendeklarasikan variabel *ignore_missing = 0.* Ini berarti pada proses akuisisi dengan tipe *file* AFF1 dilakukan pengecekan sektor data apakah terdapat data yang hilang, *corrupt*¸ atau *missing.* Dengan memberikan nilai 0, menunjukkan pada metode ini terdapat pengecekan kualitas data.

4.2 Pseudocode

Membuat *pseudocode* akan memudahkan dalam melakukan analisis maupun ketika menghitung efisiensi waktu pada algoritma. *Pseudocode* dibuat dengan membaca kode baris per baris kemudian mendefinisikannya ke dalam bahasa yang lebih mudah dimengerti. Dalam menulis *pseudocode* ketika program memanggil suatu fungsi, maka parameter akan disertakan sesuai dengan bentuk pada bahasa C. Ini dikarenakan pada saat membuat *pseudocode* peneliti menemui fungsi dengan banyak parameter. Peneliti sudah mencoba untuk mengganti pendefinisian ke dalam bentuk lain justru membuat peneliti susah dalam membaca prosesnya. Dalam penulisan tipe data dibedakan antar tipe data supaya lebih rapi dan mudah dibedakan saat membacanya.

Pseudocode Format Raw

```
Program: Pemrograman Akuisisi Format Raw
```

```
Kamus:

*ibuf: unsigned char
*bufstart: unsigned char
*real_obuf: unsigned char
secstr[100]: char


exit_status: int
input_from_stream: int
input_from_pattern: int
seconds: int
difftime: int
nwritten: int
prcnt: int
i: unsigned int

fprcnt: float
```

```
    probed_size: float
```

Deskripsi

```
    Call function ssize_t and nread to read bytes in the current block
    exit_status ←0
    input_from_stream ← !!input_file
    input_from_pattern ← !input_from_stream
    size_t page_t ← Call getpagesize()
    Read the size and bytes in the current block
    real_buf ← (unsigned char *)malloc(input_blocksize + 2 * SWAB_ALIGN_OFFSET +
2 * page_size - 1)
    ibuf ←real_buf
    allow space for swab by compute ibuf = ibuf + SWAB_ALIGN_OFFSET
    ibuf ← PTR_ALIGN(ibuf, page_size)

    if(do_hash) then
        Call hash_upadate(ihashlist,NULL, 0)
    endif

    if (conversions_mask & C_TWOBUFS)
        real_obuf ← (unsigned char *) malloc(output_blocksize + page_size - 1)
        obuf ← PTR_ALIGN(real_obuf, page_size)

    else
        real_obuf ← NULL
       obuf ← ibuf
    endif

    if(!input_from_pattern)
        if( skip_record != 0)
            Call skip(STDIN_fileNO, input_file, skip_records, input_blocksize, ibuf)
        endif
    endif

    if(seek_records !=0)
        outputlist_t *listptr;
        for (listptr = outputlist; listptr != NULL; listptr = listptr->next)
            Call skip(listptr->data.fd, "", seek_records, output_blocksize, obuf)
        endfor
    endif

    if(max_records == 0)
```

```
        Call quit(exit_status);
    endif

    if(input_from_pattern) then
        Call replicate_pattern(pattern, ibuf, input_blocksize)
        nread ← n_bytes_read ← input_blocksize
    endif

    while (1)
        if(do_status && w_full % update_thresh == 0 && w_full != 0)
            off_t total_bytes ← w_full * input_blocksize
            off_t total_mb ← total_bytes / 1048576
            if (probe == PROBE_NONE OR probe_soze == 0)
                output(stderr, "\r%llu blocks (%lluMb) written.",w_full, total_mb)
            else
                time_t curr_time ← time(NULL)
                seconds ← Read by compute call difftime(curr_time, start_time)
                off_t probed_mb ← probed_size/1048576
                fprcnt ← total_bytes/probed_size
                fprcnt_remainging ← 1.0 - fprcnt
                prcnt ← fprcnt * 100
                seconds_remaining ← seconds * (fprcnt_remaining / fprnct)

                Call time_left(secstr, sizeof secstr, seconds_remaining)
                        output(stderr, "\r[%d%% of %lluMb] %llu blocks (%lluMb)
                    written.   %s",prcnt, probed_mb, w_full, total_mb, secstr)
            endif
        endif

        if(r_partial + r_full >= max_records)
            break
         endif

        if(!input_from_pattern)
            if ((conversions_mask & C_SYNC) && (conversions_mask & C_NOERROR))
                        Call memset(ibuf,(conversions_mask & (C_BLOCK | C_UNBLOCK))
                    ? ' ' : '\0',input_blocksize)
            nread ← safe_read(STDIN_fileNO, ibuf, input_blocksize))
            endif
        endif
        if(nread==0)
              break
        endif
```

```
if(nread < 0 && !input_from_pattern)
    Call syscall_error_noexit(input_file)
    if(conversions_mask & C_NOERROR)
        Call print_stats()

        Call lseek(STDIN_fileNO, (off_t) input_blocksize, SEEK_CUR)
        if(conversions_mask & C_SYNC)
                    Call  memset (ibuf,(conversions_mask  &  (C_BLOCK  |
                C_UNBLOCK)) ? ' ' : '\0',input_blocksize)
                nread ← 0
             else
            continue
             endif
    else
             * Write any partial block. */
             exit_status=2
             break
    endif
endif

n_bytes_read ← nread


if (do_hash && hashconv == HASHCONV_BEFORE)
    hash_upadte (ihashlist, ibuf, n_bytes_read)
endif

if(n_bytes_read < input_blocksize)
    rpartial←npartial + npartial
    if(coversions_mask & C_SYNC)
        if(!(conversions_mask & C_NOERROR))
        /* If C_NOERROR, we zeroed the block before reading. */
                    Call memset ((ibuf + n_bytes_read),(conversions_mask
                & (C_BLOCK | C_UNBLOCK)) ? ' ' : '\0', input_blocksize
                - n_bytes_read)
        endif
      n_bytes_read ← input_blocksize
    endif
else
    r_full ← r_full + r_full
endif

if(ibuf == obuf) /* If not C_TWOBUFS. */
      /* int nwritten = full_write(STDOUT_fileNO, obuf, n_bytes_read); */
```

```
            nwritten ← Call nwritten = outputlist_write(obuf, n_bytes_read)
            if(nwritten < 0)
                Call syscall_error(output_file)
            else if (n_bytes_read == input_blocksize)
                w_full ← w_full + w_full
                endif
            else
                w_partial ← w_partial + w_partial
            endif
        else /* If C_TWOBUFS */
            /* Do any translations on the whole buffer at once.  */
            if (translation_needed)
                Call translate_buffer(ibuf,n_bytes_read)
            endif
            if (conversions_mask & C_SWAB)
                bufstart ← Call swab_buffer(ibuf, &n_bytes_read)
            else
                bufstart ← ibuf
            endif
            if(conversions_mask & C_BLOCK)
                Call copy_with_block(bufstart, n_byter_ read)
            else if (conversions_mask & C_UNBLOCK)
                call copy_with_unblock(bufstart, n_bytes_read)
                endif
            else
                Call copy_simple(bufstart, n_bytes_read)
            Endif
        if (do_hash && hashconv == HASHCONV_AFTER)
            hash_update(ihashlist, ibuf, n_bytes_read);
         endif
    endwhile

    if(char_is_saved)
        if(conversions_mask & C_BLOCK)
              Call copy_with_block(&saved_char, 1)
         endif
    else if (conversions_mask & C_UNBLOCK)
        copy_with_unblock(&saved_char, 1)
        endif
    else
        Call output_char(saved_char)
    endif

    if ((conversions_mask & C_BLOCK) && col > 0)
```

```
        for (i = col; i < conversion_blocksize; i++)
         Call output_char(space_character)
        endfor
    endif
    if ((conversions_mask & C_UNBLOCK) && col == conversion_blocksize)
        /* Add a final '\n' if there are exactly `conversion_blocksize'
        characters in the final record. */
        Call output_char(newline_character)
    Endif

    if (oc != 0)
        nwritten ← Call outputlist_write(obuf, oc)
        if(nwritten > 0)
         w_partial← w_partial + w_partial
        endif
        if(nwritten < 0 )
         Call syscall_error(output_file)
        endif
    endif

    Call free(real_buf)

    if(real_obuf)
        Call free(real_obuf)
    endif

    if(do_hash)
        Call hash_remainder(ihashlist, WINDOW_CTX)
        Call display_totalhash(ihash,list, TOTAL_CTX)
    endif
    Return (exit_status)
```

Gambar 4. 14 *Pseudocode* Format Raw

Pseudocode Format AFF

```
Aplikasi: Pemrograman Akuisisi Format AFF

Kamus:
ignore_missing: static int //static berarti variabel tersimpan pada memori meski
fungsi sudah terlewati, nilai variabel biasanya dihapus setelah fungsi berjalan.
```

```
    do_copy: static int
    copy_recursive: int
    num: int
    argc: int
    x_copy: int

    AffReader_s *r: struct
    AffWriter_s *w: struct
    AffNode_s *r_root: struct
    AffNode_s *r_node: struct
    copy_nodes_arg arg: struct
    stat stat_fb: struct


    *src_kpath: const char // * berarti variabel src_kpath hanya menyimpan alamat
(sebagai pointer) tidak menyimpan data
    *dst_kpath: const char
    *status: const char
    *list_fname: const char
    *src_fname: const char
    *dst_fname: const char
    *list1_fname: const char
    *list2_fname: const char

    *fargv[2]: char
    buf[32768]: char
    **argv: char
    *p: char
    *tmp_fname: char
```

Deskripsi:

```
    Ignore_missing ← 0
    Call do_copy( pointer to AffReader_s *r, pointer to AffWriter_s *w,
             *src_kpath, *dst_kpath, copy_recursive

    AffNode_s *r_root ← aff_reader_root(r)

    AffNode_s *r_node ← lookup_path(r, r_root, src_kpath)

    if(NULL == r_node)
        Output (stderr, "%s: [%s]: cannot read node\n", __func__, src_kpath)
        Return (!ignore_missing)
    endif
```

```
    if(aff_reader_root(r)== r_node)
           Output (stderr, "%s: %s: cannot copy the root node; use extract
       instead\n",    func__, src_kpath ))
           Return (1)
    endif

    AffNode_s *w_node ← aff_writer_mkpath(w, aff_writer_root(w),dst_kpath)

    if (NULL == w_node)
           Output ( stderr, "%s: cannot write to key [%s]: %s\n", __func__,
       dst_kpath, aff_writer_errstr( w ) ))
           Return (1)
    endif

    if (aff_writer_root(w) == w_node)
        Output(stderr, "%s: %s: cannot replace the root node; use extract
instead\n",
           __func__, dst_kpath ))
        Return (1)
    endif

    if (NULL != ( status = copy_node_data( r, r_node, w, w_node, COPY_NODE_STRONG
) ) )
        Output ( stderr, "%s: %s", __func__, status )
        Return(1)
    endif

    if (copy_recursive)
        arg.r ← r
        arg.w ← w
        arg.w_parent ← w_node
        arg.weak ← COPY_NODE_WEAK
        arg.errstr ← NULL
           Call    aff_node_foreach    (    pointer    to    r_node,    pointer    to
       copy_nodes_recursive,   pointer to(void *)&arg )

       if (NULL != arg.errstr)
           Output ( stderr, "%s: %s\n", __func__, arg.errstr )
           Return (1)
        endif
    Return (0)
    endif
```

```
    Call copy_keylist( pointer to AffReader_s *r, pointer to AffWriter_s *w,
    pointer to *list_fname, pointer to copy_recursive )

       file pointer to list

       if(0 == strcmp( list_fname, "-" ))
            list ← stdin
            if(ferror( list ))
                Output (( stderr, "%s: bad stdin stream\n", __func__ ))
                Return (1)
            endif
       else
            if (NULL == ( list = fopen( list_fname, "r" ) ))
                output ( stderr, "%s: cannot open %s\n", __func__, list_fname )
                return (1)
            endif
        endif

       while (NULL != fgets( buf, length(buf), list )) do
            if ( '\n' != buf[strlen(buf)-1] )
                output( stderr, "%s: line too long, skipping\n", __func__ )
                while( NULL != fgets( buf, length(buf), list ) )
                    if( '\n' == buf[strlen(buf)-1] )
                        break
                    continue
            endif

            num ← split_farg(buf, 2, fargv)

            if (num<0)
                output( stderr, "%s: unexpected result of split_farg;   exiting\n",
__func__ )
                go to errclean_r
            endif

            if (num == 0)
                continue
            endif

            if (num < 2)
                        output( stderr, "%s: syntax error: expect 2 names, only %d
                     given\n", __func__, num );
                go to errclean_r
            endif
```

```
        if ( do_copy( r, w, fargv[0], fargv[1], copy_recursive ) )
            go to errclean_r
        endif

    fclose (list)
    return (0)
errclean_r
    fclose(list)
    return(1)
endwhile

Call x_copy (pointer to argc, pointer to **aargv)
    copy_recursive ← 0
    src_fname ← NULL
    dst_fname ← NULL
    list1_fname ← NULL
    list2_fname ← NULL
    status ← NULL

    for (; argc ; --argc, ++argv)
        if ('-' != argv[0][0])
            break
        endif

        for (p = argv[0] + 1 ; '\0' != *p ; ++p)
            switch (*p)
            case value of
                i : ignore_missing ← 1
                    break
                R : copy_recursive ← 1
                    break
                f : in condition if (( '\0' != *(p+1) || !(--argc))
                                        output (stderr, "%s: -f must be
                                    followed by a file name\n", __func__
                                    )
                                    return (1)
                            list1_fname ← *(++argv)
                    break
                F : in condition if ('\0' != *(p+1) || !(--argc))
                                        output ( stderr, "%s: -F must be
                                    followed by a file name\n", __func__
                                    )
                            return (1)
```

```
                                list2_name = *(++aarv)
                    break
                default:
                                    output(tderr, "%s: unknown option -%c\n",
                                __func__, *p)
                    return(1)
            endcase
        endfor

        if (!(argc--))
                output ( stderr, "%s: target aff file name should be given\n",
            __func__ )
            return(1)
        else
            dst_fname ← *(argv++)
        endif

        if (argc % 2)
            output( stderr, "%s: unpaired src_kpath; exiting\n", __func__ )
            return(1)
        endif

        if (NULL == (r = aff_reader( src_fname )))
            output((stderr, "%s: not enough memory\n", __func__))
            return(1)
        endif

        if ( NULL != aff_reader_errstr( r ) )
                output fprintf( stderr, "%s: %s: %s\n", __func__, src_fname,
            aff_reader_errstr( r ) )
                goto errclean_r
        endif
        *tmp_fname ← mk_tmp_filename( ".aff-tmp.", dst_fname )

        if(NULL == tmp_fname)
                output printf( stderr, "%s: could not create a unique writable
            file\n", __func__ )
                goto errclean_r
        endif

        if (NULL == (w = aff_writer( tmp_fname )))
            output(stderr, "%s: not enough memory\n", __func__)
            goto errclean_r_free
        endif
```

```
if ( NULL != aff_writer_errstr( w ) )
          output fprintf(  stderr,  "%s:  %s:  %s\n",  __func__,  tmp_fname,
      aff_writer_errstr( w ) )
          goto errclean_rw
endif

if(NULL != list1_fname)
    goto errclean_rw
    if ( copy_keylist( r, w, list1_fname, copy_recursive ) )
        goto errclean_rw
endif

for ( ; argc ; argc -= 2, argv += 2 )
    if ( do_copy( r, w, argv[0], argv[1], copy_recursive ) )
        goto errclean_rw
        endif
endfor

if ( NULL != list2_fname )
    if ( copy_keylist( r, w, list2_fname, copy_recursive ) )
        goto errclean_rw
    endif
aff_reader_close(r)

if (!stat(dst_fname, &stat_fb))
    if (NULL == (r = aff_reader( dst_fname )))
        output (stderr, "%s: not enough memory\n", __func__)
        goto errclean_w
    endif

    if(NULL != aff_reader_errstr(r))
                output   fprintf(    stderr,    "%s:    %s\n",    __func__,
            aff_reader_errstr(r))
                goto errclean_rw
    endif

    arg.r ← r
    arg.w ← w
    arg.w_parent ← aff_writer_root(w)
    arg.w ← COPY_NODE_WEAK
    arg.errstr ← NULL
```

```
                CALL     aff_node_foreach     (     aff_reader_root(     r     ),
            copy_nodes_recursive, (void *)&arg )

            if (NULL != arg.errstr)
                output ( stderr, "%s: %s\n", __func__, arg.errstr )
                goto errclean_rw
            endif
            CALL aff_reader_close(r)
        endif

        if ( NULL != ( status = aff_writer_close(w)))
            output ( stderr, "%s: %s: %s\n", __func__, tmp_fname, status )
            goto errclean_file
        endif
        if ( rename( tmp_fname, dst_fname ) )
            CALL perror(dst_fname)
            output( stderr, "%s: output is saved to %s\n", __func__, tmp_fname )
            goto errclean_free
        endif

        CALL free(tmp_fname)
        return(0)

    errclean_rw:
        CALL aff_reader_close(r)
    Errclean_w:
        CALL aff_writer_close(w)
    Errclean_file:
        If(remove(tmp_fname)
            CALL(perror(tmp_fname)
    Errclean_r_free:
        CALL free(tmp_fname)
    Errclean_r:
        Aff_reader_close(r)
        Return(1)

    in the h_copy function
        Output ("Usage:\n"
                "lhpc-aff cp [-iR] [-f <list>] [-F <list>] <src-file> <dst-file>\n"
                "\t\t[<src-kpath> <dst-kpath>] ...\n"
                "Copy data from <src-kpath> of <src-file> to <dst-kpath> of <dst-
file>:\n"
                "<dst-file>[<dst-kpath>]  <--  <src-file>[<src-kpath>].\n"
```

```
                "Keys <src-kpath> and <dst-kpath> cannot be root nodes, use
'extract' instead.\n"
                "New data replaces old data. If <dst-file> does not exist, it will
be created.\n"
                "If <src-file> and <dst-file> are the same, nodes to copy will be put
first,\n"
                "and then the rest of file will be reinserted.\n"
                "Options:\n"
                "\t-i\tignore error if <src-keypath> do not exist\n"
                "\t-R\tcopy data sub-entries recursively.\n"
                "\t-f <pre-list>\n\t\texecute copy instructions in file <pre-list>
BEFORE\n"
                "\t\tprocessing command line list\n"
                "\t-F <post-list>\n\t\texecute copy instructions in file <pre-list>
AFTER\n"
                "\t\tprocessing command line list\n"
                "\t\teach line of <list> is a separate insert instruction,\n"
                "\t\tand it must have at least two names, in order:\n"
                "\t\t<src-kpath> <dst-kpath>\n"
                "\t\tparameter to only one of -f, -F options can be '-' (stdin).\n"
```

Gambar 4. 15 *Pseudocode* Format AFF

Pada penelitian ini pembuatan *pseudocode* sangat penting. Karena untuk menghitung nilai-nilai yang nantinya akan digunakan dalam menghitung efisiensi waktu pada algoritma.

4.3 Menghitung Efisiensi Waktu

Hasil dari *pseudocode* kembali dibaca baris per baris untuk mengetahui nilai pada setiap *statement* yang digunakan. Kemudian setiap *statement* pada kode diberikan *cost* dan *time* untuk mengetahui waktu yang dibutuhkan pada setiap baris kode. Setelah nilai tiap baris dihitung, analisis dilakukan dengan menghitung per blok (sekumpulan perintah) untuk memperjelas perhitungan. Untuk penjelasan lebih detail mengenai analisis algoritma tercantum pada bagian lampiran. Berdasarkan analisis dari setiap *pseudocode* didapatkan hasil pada tabel berikut.

4.3.1 Membuat tabel *cost-time*

Pada tabel menggunakan dua variabel yaitu *cost* dan *time*. Pada kolom *cost* mencatat biaya yang digunakan pada proses C_n. Sedangkan kolom *time* mencatat waktu yang dibutuhkan pada saat algoritma berjalan dengan satuan *n*.

a. Tabel Format Raw

Setelah dilakukan analisis terdapat 116 *cost* pada algortima. Proses tersebut terdapat operasi dasar, *if-else, for*, pengisian nilai, pemanggilan fungsi dan deklarasi variabel. C_1didapatkan dari awal baris pertama kode hingga C_{116} pada akhir kode. C_1hingga C_{116} menunjukkan *cost* yang dibutuhkan ketika kode dieksekusi pada setiap baris.

Tabel 4. 1 Keseluruhan *cost-time* format Raw

Cost	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*
C1	1	C21	1	C41	1	C61	1	C81	1	C101	1
C2	1	C22	1	C42	1	C62	1	C82	1	C102	1
C3	1	C23	n	C43	1	C63	1	C83	1	C103	1
C4	1	C24	1	C44	1	C64	1	C84	1	C104	1
C5	1	C25	1	C45	1	C65	1	C85	1	C105	1
C6	1	C26	1	C46	1	C66	1	C86	1	C106	1
C7	1	C27	1	C47	1	C67	1	C87	1	C107	1
C8	1	C28	1	C48	1	C68	1	C88	1	C108	1
C9	1	C29	1	C49	1	C69	1	C89	1	C109	1
C10	1	C30	1	C50	1	C70	1	C90	1	C110	1
C11	1	C31	1	C51	1	C71	1	C91	1	C111	1
C12	1	C32	1	C52	1	C72	1	C92	1	C112	1
C13	1	C33	1	C53	1	C73	1	C93	1	C113	1
C14	1	C34	1	C54	1	C74	1	C94	1	C114	1
C15	1	C35	1	C55	1	C75	1	C95	1	C115	1
C16	1	C36	1	C56	1	C76	1	C96	1	C116	1
C17	1	C37	1	C57	1	C77	1	C97	1		
C18	1	C38	1	C58	1	C78	1	C98	1		
C19	1	C39	1	C59	1	C79	1	C99	1		
C20	1	C40	1	C60	1	C80	1	C100	n		

Tabel 4.1 merupakan hasil yang diperoleh dari penrhitungan *time* yang diperlukan pada setiap *cost*. Nilai 1 menunjukkan proses berjalan konstan. Berapa pun nilai *n*, waktu yang digunakan sama, yaitu 1. Proses tersebut juga meliputi operasi dasar, deklarasi variabel, dan operasi pengisian nilai. Kemudian *n* menunjukkan bahwa algoritma dijalankan membutuhkan waktu sebanyak *n*.

Setelah nilai proses secara keseluruhan sudah dicatat, maka akan dipecah lagi menjadi beberapa bagian yang lebih kecil berdasarkan variabel dan konstantanya. Tujuannya adalah untuk memudahkan melihat nilai berdasarkan variabel nya.

Tabel 4. 2 *Cost* Raw dengan nilai waktu konstan

C1	1	C18	1	C97	1	C113	1
C2	1	C19	1	C98	1	C114	1
C3	1	C20	1	C100	1		
C4	1	C21	1	C101	1		
C5	1	C22	1	C102	1		
C6	1			C103	1		
C7	1	C25	1	C104	1		
C8	1	C26	1	C105	1		
C9	1	C27	1	C106	1		
C10	1	C28	1	C107	1		
C11	1	C29	1	C108	1		
C12	1	C30	1	C109	1		
C13	1	C91	1	C110	1		
C14	1	C92	1	C111	1		
C15	1	C93	1	C112	1		

Pada Tabel 4.2 tersebut terlihat *cost* yang memiliki waktu konstan.

Tabel 4. 3 *Cost* Raw dengan waktu *n*

C23	n
C100	n

Tabel 4.3 menunjukkan nilai waktu sebesar *n*. Untuk proses dengan waktu sebesar *n* terjadi pada operasi *looping* yang terjadi di luar kalang bersarang.

Tabel Format AFF

Analisis yang dilakukan pada algoritma akuisisi dengan format AFF menghasilkan 162 nilai *cost* yang digunakan.

Tabel 4. 4 Keseluruhan c*ost-time* pada AFF keseluruhan waktu

cost	time	cost	time	cost	time	cost	time	cost	time	cost	time
C1	1	C31	1	C61	1	C91	1	C121	n	C151	1
C2	1	C32	1	C62	1	C92	1	C122	1	C152	1
C3	1	C33	1	C63	1	C93	1	C123	1	C153	1
C4	1	C34	1	C64	1	C94	1	C124	1	C154	1
C5	1	C35	1	C65	1	C95	1	C125	1	C155	1
C6	1	C36	1	C66	1	C96	1	C126	1	C156	1
C7	1	C37	1	C67	1	C97	1	C127	1	C157	1
C8	1	C38	1	C68	1	C98	1	C128	1	C158	1
C9	1	C39	1	C69	1	C99	1	C129	1	C159	1
C10	1	C40	1	C70	1	C100	1	C130	1	C160	1
C11	1	C41	1	C71	n	C101	1	C131	1	C161	1
C12	1	C42	1	C72	1	C102	1	C132	1	C162	1
C13	1	C43	n	C73	1	C103	1	C133	1		
C14	1	C44	1	C74	n^2	C104	1	C134	1		
C15	1	C45	1	C75	1	C105	1	C135	1		
C16	1	C46	n^2	C76	1	C106	1	C136	1		
C17	1	C47	1	C77	1	C107	1	C137	1		
C18	1	C48	1	C78	1	C108	1	C138	1		
C19	1	C49	1	C79	1	C109	1	C139	1		
C20	1	C50	1	C80	1	C110	1	C140	1		
C21	1	C51	1	C81	1	C111	1	C141	1		
C22	1	C52	1	C82	1	C112	1	C142	1		
C23	1	C53	1	C83	1	C113	1	C143	1		
C24	1	C54	1	C84	1	C114	1	C144	1		
C25	1	C55	1	C85	1	C115	1	C145	1		
C26	1	C56	1	C86	1	C116	1	C146	1		
C27	1	C57	1	C87	1	C117	1	C147	1		
C28	1	C58	1	C88	1	C118	1	C148	1		
C29	1	C59	1	C89	1	C119	n^2	C149	1		
C30	1	C60	1	C90	1	C120	1	C150	1		

Tak berbeda dengan penjelasan format sebelumnya pada Tabel 4.5 tersebut terdapat waktu yang digunakan ketika menjalankan algoritma. Pada Algoritma tersebut terdapat bermacam operasi, yaitu matematika, pengisian data, pendeklarasian variabel, *if-else*, ataupun *while.* Algoritma diawali dengan pengisian nilai pada variabel *Ignore*_missing = 0. *Ignore*_missing menunjukkan bahwa metode ini dilakukan pengecekan pada data. Terlihat berdasarkan hasil analisis terdapat waktu dominan dengan nilai *n*. Hal ini dikarenakan pada

algoritma jarang ditemui menggunakan kalang bersarang ketika ketika menjalankan suatu operasi. Pangkat tertinggi pada hasil analisis adalah n^2.

Selanjutnya adalah memecah tabel tersebut menjadi bagian yang lebih kecil untuk memisahkan variabel waktunya.

Tabel 4. 5 *Cost* AFF dengan waktu konstan

Cost	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*
C1	1	C31	1	C63	1	C95	1	C126	1	C156	1
C2	1	C32	1	C64	1	C96	1	C127	1	C157	1
C3	1	C33	1	C65	1	C97	1	C128	1	C158	1
C4	1	C34	1	C66	1	C98	1	C129	1	C159	1
C5	1	C35	1	C67	1	C99	1	C130	1	C160	1
C6	1	C36	1	C68	1	C100	1	C131	1	C161	1
C7	1	C37	1	C69	1	C101	1	C132	1	C162	1
C8	1	C38	1	C70	1	C102	1	C133	1		
C9	1	C39	1	C72	1	C103	1	C134	1		
C10	1	C40	1	C73	1	C104	1	C135	1		
C11	1	C41	1	C75	1	C105	1	C136	1		
C12	1	C42	1	C76	1	C106	1	C137	1		
C13	1	C44	1	C77	1	C107	1	C138	1		
C14	1	C45	1	C78	1	C108	1	C139	1		
C15	1	C47	1	C79	1	C109	1	C140	1		
C16	1	C48	1	C80	1	C110	1	C141	1		
C17	1	C49	1	C81	1	C111	1	C142	1		
C18	1	C50	1	C82	1	C112	1	C143	1		
C19	1	C51	1	C83	1	C113	1	C144	1		
C20	1	C52	1	C84	1	C114	1	C145	1		
C21	1	C53	1	C85	1	C115	1	C146	1		
C22	1	C54	1	C86	1	C116	1	C147	1		
C23	1	C55	1	C87	1	C117	1	C148	1		
C24	1	C56	1	C88	1			C149	1		
C25	1	C57	1	C89	1	C118	1	C150	1		
C26	1	C58	1	C90	1	C120	1	C151	1		
C27	1	C59	1	C91	1	C122	1	C152	1		
C28	1	C60	1	C92	1	C123	1	C153	1		
C29	1	C61	1	C93	1	C124	1	C154	1		
C30	1	C62	1	C94	1	C125	1	C155	1		

Pada Tabel 4.9 tersebut menunjukkan nilai waktu konstan. Waktu konstan berarti tidak terpengaruh dengan banyaknya *n* data, Artinya pada bagian tersebut walaupun nilai *n* bertambah dua kali ataupun empat kali dari sebelumnya, maka waktu yang digunakan tetap sama.

Tabel 4. 6 *Cost* AFF dengan waktu *n*

C43	n
C71	n
C121	n

Proses yang menggunakan waktu sebesar *n*. Pada Tabel 4.7 proses dengan waktu sebesar *n* tidak sebanyak seperti waktu pada tabel konstan. Arti dari *n* adalah algoritma akan berjalan secara linier. Sehingga jika nilai input dinaikkan sebanyak *y* kali maka akan berpengaruh dengan waktu yang akan dibutuhkan bertambah sebesar *y* kali.

Tabel 4. 7 *Cost* AFF dengan waktu n^2

C46	n^2
C74	n^2
C119	n^2

Tabel 4.11 tersebut menunjukkan waktu sebesar n^2. Yang berarti pada proses tersebut jika input dinaikkan sebesar dua kali, maka waktu yang dibutuhkan oleh algoritma untuk berjalan empat kali lebih besar dari waktu semula.

4.3.2 Notasi Big O

Setelah mencari *cost* dan *time* yaitu menghitung total waktu yang digunakan selama algoritma berjalan. Caranya dengan menjumlah semua nilai *time* pada masing-masing format dengan persamaan (4.1).

$$Waktu\ Total = \sum_{n}^{i} C_n \qquad (4.1)$$

Pada format raw terdapat satu fungsi yairu, *dd_copy*. Pada format AFF terdapat tiga fungsi yaitu, *do_copy*, *copy_keylist*, dan *x_copy*. Kemudian fungsi tersebut dikelompokkan ke dalam tabel. Tabel terdiri dari blok (kumpulan nilai *cost*), *time, Big-O,* dan keterangan. Penggunaan blok digunakan untuk melihat nilai *Big-O* pada setiap perintah yang digunakan pada perintah *if-else*, atau *looping*.

a. Format Raw fungsi dd_copy

Tabel 4. 8 Analisis *Big-O* pada fungsi *dd_copy*

	Cost	Time	Big-O	Keterangan
	C1	1	O(1)	Konstan
	C2	1	O(1)	
	C3	1	O(1)	
	C4	1	O(1)	
	C5	1	O(1)	
	C6	1	O(1)	
	C7	1	O(1)	
	C8	1	O(1)	
	C9	1	O(1)	
	C10	1	O(1)	
	C11	1	O(1)	
	C12	1	O(1)	
	C13	1	O(1)	
Blok 1	sub blokC14-C17	1	O(1)	
Blok 2	sub blok C18-C20	1	O(1)	
Blok 3	C21	n	O(n)	Linier
	C22	1		
	C23	n		
	C24	1		
Blok4	sub blok C25-C26	1	O(1)	Konstan
Blok 5	sub blok C27-C29	1	O(1)	
Blok 6	C30	n	O(n)	Linier
	sub blok C31 - C44	1		
	sub blok C45-C46	1		
	sub blok C47-C50	1		
	sub blok C51-C52	1		
	sub blok C53-C63	1		
	C64	1		
	sub blok C65-C66	1		
	sub blok C67-C73	1		
	sub blok C74-C90	1		
	sub blok C91-C92	1		
Blok 7	C93-C98	1	O(1)	Konstan
Blok 8	C99	1	O(n)	Linier
	sub blok C100-C101	n		
Blok 9	C102-C103	1	O(1)	Konstan
Blok 10	C105	1	O(1)	
	sub blok C106-107	1		
	C108-C109	1		
	C110	1	O(1)	
Blok 11	C111-C112	1	O(1)	
Blok 12	C113-C115	1	O(1)	
	C116	1	O(1)	

Pada blok 3, blok 6, dan blok 8 memiliki nilai *Big-O* yaitu O(n) karena pada blok tersebut terdapat nilai n atau linier pada perintah di dalamnya. Walaupun terdapat nilai 1 pada blok tersebut namun *Big-O* melihat pada nilai dengan derajat n terbesar sehingga O(n).

- Misal pada waktu konstan

$$C1 + C2 + \ \ldots + C20 + C22 + C24 + \ \ldots + C29 + C31 + \ldots + C99$$
$$+C102 + \ldots + C116 = A$$

maka

$$\Rightarrow 1(A) = A \tag{4.2}$$

- Misal pada waktu n

$$C21 + C23 + C30 + C100 + C101 = B$$

maka

$$\Rightarrow n(B) = nB \tag{4.3}$$

Setelah mengasumsikan ke variabel lain dilanjutkan dengan menjumlahkan keseluruhan waktu

$$Waktu\ Total = \sum_{n=1}^{116} C_n = A + nB \tag{4.5}$$

$$Waktu\ Total = nB + A \tag{4.6}$$

Untuk mengubahnya ke dalam notasi Big O dengan memperhatikan n dengan pangkat tertinggi dan mengabaikan koefisiennya. Big O untuk algoritma akuisisi dengan format Raw adalah $O(n)$. Notasi asimtotik $O(n)$ termasuk ke dalam notasi linier.

b. Format AFF

AFF memiliki 3 fungsi yaitu *do_copy*, *copy_keylist,* dan *x_copy* sehingga masing-masing fungsi dihitung kompleksitas waktunya. Fungsi utama pada AFF adalah *x_copy*.

Fungsi *do_copy*

Tabel 4. 9 Analisis *Big-O* pada fungsi *do_copy*

	Cost	Time	Big-O	Keterangan
	C1	1	O(1)	Konstan
	C2	1	O(1)	
	C3	1	O(1)	
	C4	1	O(1)	
Blok 1	C5-C7	1	O(1)	
Blok 2	C8-C10	1	O(1)	
Blok 3	C12-C14	1	O(1)	
Blok 4	C15-C17	1	O(1)	
Blok 5	C18-C20	1	O(1)	
Blok 6	C21-C32	1	O(1)	

- Berdasarkan tabel 4.9, misal pada waktu konstan

$$C1 + C2 + C3 + C4 + C5 + C6 + \ldots + C32 = C$$

maka

$$\Rightarrow 1(C) = C \tag{4.7}$$

Sehingga waktu pada fungsi *do_copy*

$$Waktu = \sum_{n=1}^{32} C_n = 1 \tag{4.8}$$

Jadi *Big-O* pada fungsi tersebut adalah O(1) atau konstan.

Fungsi *copy_keylist*

Tabel 4. 10 Analisis *Big-O* pada fungsi *copy_keylist*

	Cost	Time	Big-O	Keterangan
	C33	1	O(1)	Konstan
	C34	1	O(1)	
Blok 7	C35-C42	1	O(1)	Konstan
Blok 8	sub blok C43-C48	n^2	$O(n^2)$	Kuadratik
	C49	1		
	sub blok C50-C52	1		
	sub blokC53-C54	1		
	sub blok C55-C57	1		
	sub C58-59	1		
	C60	1		
	C62	1		
	C63	1		

Pada Blok 8 mengambil *time* maksimal dari blok 8 yang notasinya kuadratik karena derajat tertingginya adalah n^2. n^2 terdapat pada C46 karena menggunakan perintah *loop* dan berada dalam satu kalang bersarang yang merupakan perintah *loop*.

- Berdasarkan tabel 4.10 misal pada waktu konstan

$$C33 + C34 + C35 + \ldots + C42 + C49 + \ldots + C63 = D$$

maka

$$\Rightarrow 1(D) = D \tag{4.9}$$

- Karena tidak terdapat waktu linear atau O(n), maka langsung menghitung ke waktu kuadratik n^2. Karena hanya terdapat satu notasi kuadratik misal,

$$C46 = E$$

maka

$$\Rightarrow n^2(E) = n^2E \tag{4.10}$$

Sehingga waktu pada fungsi *copy_keylist* dengan menjumlahkan semua persamaan.

$$Waktu = \sum_{n=33}^{63} C_n = D + n^2\mathrm{E} \tag{4.11}$$

Setelah dijumlahkan akan menjadi persamaan (4.11). Karena derajat tertinggi adalah n^2 maka *Big-O* pada fungsi tersebut adalah $O(n^2)$ atau kuadratik.

Fungsi x_copy

Tabel 4. 11 Analisis *Big-O* pada fungsi *x_copy*

	Cost	Time	Big-O	Keterangan
	C64-C70	1	O(1)	Konstan
Blok 9	C71	n	$O(n^2)$	Kuadratik
	sub blok C72-C73	1		
	C74	n^2		
	sub blok C75-C92	1		
Blok 10	sub blok 93-96	1	O(1)	Konstan
Blok 11	sub blok C97 - C99	1	O(1)	
Blok 12	sub blok C100-C102	1	O(1)	
Blok 13	sub blok 103-105	1	O(1)	
	C106	1	O(1)	
Blok 14	sub blok 107-C109	1	O(1)	
Blok 15	sub blok C110-C112	1	O(1)	
Blok 16	sub blok C113-C116	1	O(1)	
Blok 17	sub blok C117-C120	n^2	$O(n^2)$	Kuadratik

Blok 18	C121	n	n	Linier
	C122	1		
	C123	1		
Blok 19	sub blok C124-C126	n^2	$O(n^2)$	Kuadratik
	C127	1	O(1)	Konstan
	C128	1	O(1)	
Blok 20	sub blok C129-C131	1	O(1)	
Blok 21	sub blok C132-C134	1	O(1)	
	C140	1	O(1)	
Blok 22	sub blok C141-C143	1	O(1)	
	C144	1	O(1)	
Blok 23	sub blok C145-C147	1	O(1)	
Blok 24	sub blok C148-C151	1	O(1)	
	C152	1	O(1)	
	C153	1	O(1)	
	C154	1	O(1)	
	C155	1	O(1)	
Blok 25	sub blok C156-C157	1	O(1)	
	C158	1	O(1)	
	C159	1	O(1)	
	C160	1	O(1)	

Pada blok 9 mengambil nilai *time* maksimal yang berasal dari C74. Pada C74 terdapat perintah *loop* di dalam satu kalang bersarang yang merupakan notasi *loop* juga. Sehingga pada C74 *time* nya adalah n^2.

Pada blok 17 terdapat pemanggilan fungsi *copy_keylist* pada C119 sehingga menggunakan nilai *time* dari kompleksitas waktu fungsi *copy_keylist* yaitu (n^2).

Pada blok 18 terdapat pemanggilan fungsi *do_copy* pada C122 sehingga menggunakan nilai *time* dari kompleksitas waktu fungsi *do_copy* yaitu O(1).

- Berdasarkan tabel 4.11 misal pada waktu konstan

$$C64 + \cdots C67 + C72 + C73 + C75 + \cdots + C116 + C122 + C123 + C127 + \ldots + C160 = P$$

maka

$$\Rightarrow 1(P) = P \qquad (4.12)$$

- Misal pada waktu konstan

$C1 + \ldots + C36 + C40 + C63 + \ldots + C70 + C92 + C93 + C94 + C96 + \ldots C119 + C122 + \ldots + C160 = P$

maka

$$\Rightarrow 1(P) = P \quad (4.13)$$

- Misal pada waktu n

$C71 + C121 = Q$

maka

$$\Rightarrow n(Q) = nQ \quad (4.14)$$

- Misal pada waktu n^2

$C74 + C119 + C125 = R$

(4.15)

maka

$$\Rightarrow n^2(R) = n^2R$$

Setelah mengasumsikan ke variabel lain dilanjutkan dengan menjumlahkan keseluruhan waktu.

$$Waktu\ Total = \sum_{n=1}^{160} C_n = P + nQ + n^2R \quad (4.16)$$

$$Waktu\ Total = n^2R + nQ + P \quad (4.17)$$

Untuk mengubahnya ke dalam notasi *Big-O* dengan memperhatikan pangkat tertinggi dan mengabaikan koefisien nya. *x_copy* merupakan fungsi utama sehingga *Big-O* untuk algoritma

akuisisi dengan format AFF adalah $O(n^2)$. Notasi asimtotik $O(n^2)$ termasuk ke dalam notasi kuadratik.

Setelah selesai melakukan perhitungan efisiensi waktu pada algoritma akuisisi format AFF dan Raw maka didapatkan hasil bahwa:

- Format Raw memiliki notasi *Big-O* yaitu $O(n)$.
- Format AFF memiliki notasi *Big-O* yaitu $O(n^2)$

Berdasarkan hasil penelitian diketahui bahwa format AFF memiliki derajat n yang lebih besar dari format Raw yaitu dengan notasi $O(n^2)$. Notasi $O(n^2)$ termasuk dalam notasi kuadratik pada *Big-O* yang berarti misal nilai n = 3, maka pada eksekusinya membutuhkan waktu 3^2

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari proses analisis yang telah dilakukan mendapatkan kesimpulan yang didapat dari tugas akhir dengan judul "*Analisis Efisiensi Algoritma Forensik Digital* yaitu:

a. Dengan membaca kode program baris per baris dan memperhatikan operasi yang dilakukan sehingga *flowchart* dapat dibuat dengan baik.
b. Dengan melakukan analisis kompleksitas waktu maka dapat diketahui bahwa format Raw memiliki notasi *Big-O* yaitu $O(n)$ dan format AFF memiliki notasi *Big-O* yaitu $O(n^2)$
c. Format Raw lebih efisien diabandingkan dengan AFF karena memiliki efisiensi dengan nilai kompleksitas waktu $O(n)$.

5.2 Saran

Penelitian yang telah dilakukan masih memiliki kekurangan. Peneliti memberikan saran pada penelitian berikutnya berupa:

a. Menambahkan format yang lain untuk dilakukan analisis efisiensi waktu.
b. Membuat analisis efisiensi ruang pada algoritma forensik digital.

DAFTAR PUSTAKA

Abdulghani Ali Ahmed Wan Nurulsafawati Wan Manan, E. (2015). Storage Formats for Digital Evidence. *Source*. Retrieved from http://ocw.ump.edu.my/mod/resource/view.php?id=8573

Akbal, E., & Dogan, S. (2018). Forensics Image Acquisition Process of Digital Evidence. *International Journal of Computer Network and Information Security*, *10*(5), 1–8. https://doi.org/10.5815/ijcnis.2018.05.01

Basis Technology. (2007). *AFF : The Advanced Forensics Format* (p. 23). p. 23.

Bennett, N. (2016). Introduction to Algorithms and Pseudocode Introduction to Algorithms and Pseudocode. *Researchgate*, (October). https://doi.org/10.13140/RG.2.2.28657.28008

Black, P. (2007). Big O Notation(MIT Lecture). *Dictionary of Algorithms and Data Structures*, 360–363. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Big+O+notation#1

Ervina Chintia, Rofiqoh Nadiah, Humayyun Nabila Ramadhani, Zulfikar Fahmi Haedar, Adam Febriansyah, Nur Aini Rakhmawati S.Kom., M. S. E. (2018). Kasus Kejahatan Siber yang Paling Banyak Terjadi di Indonesia dan Penangannya. *Journal Information Engineering and Educational Technology*, *02 Nomor 0*, 69.

Fisher, K., Mandelbaum, Y., & Walker, D. (2006). The next 700 data description languages. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, 2–15. https://doi.org/10.1145/1111037.1111039

Garfinkel, S., Malan, D., Dubec, K.-A., Stevens, C., & Pham, C. (2006). Advances in Digital Forensics II: IFIP international Conference on Digital Forensics, National Center for Forensic Science, Orlando, Florida, January 29-- February 1, 2006. *IFIP International Federation for Information Processing*, *222*, 13–27. https://doi.org/10.1007/0-387-36891-4_2

Greenfield, T., Goodman, S. E., & Hedetniemi, S. T. (1979). Introduction to the Design and Analysis of Algorithms. In *Applied Statistics* (3rd ed., Vol. 28). https://doi.org/10.2307/2346822

Himawan, H. (2015). Analisa Dan Perancangan Sistem Pembelajaran Online Menggunakan Metode Parsing. *Telematika*, *7*(2), 10. https://doi.org/10.31315/telematika.v7i2.421

Kaur, M., Kaur, N., & Khurana, S. (2016). A Literature review on Cyber Forensic and its Analysis tools. *Ijarcce*, *5*(1), 23–28. https://doi.org/10.17148/ijarcce.2016.5106

Republik Indonesia. (2015). *Undang - Undang Republik Indonesia No. 11 Tahun 2008 Tentang Ite*. *3*(2), 54–67. Retrieved from http://repositorio.unan.edu.ni/2986/1/5624.pdf

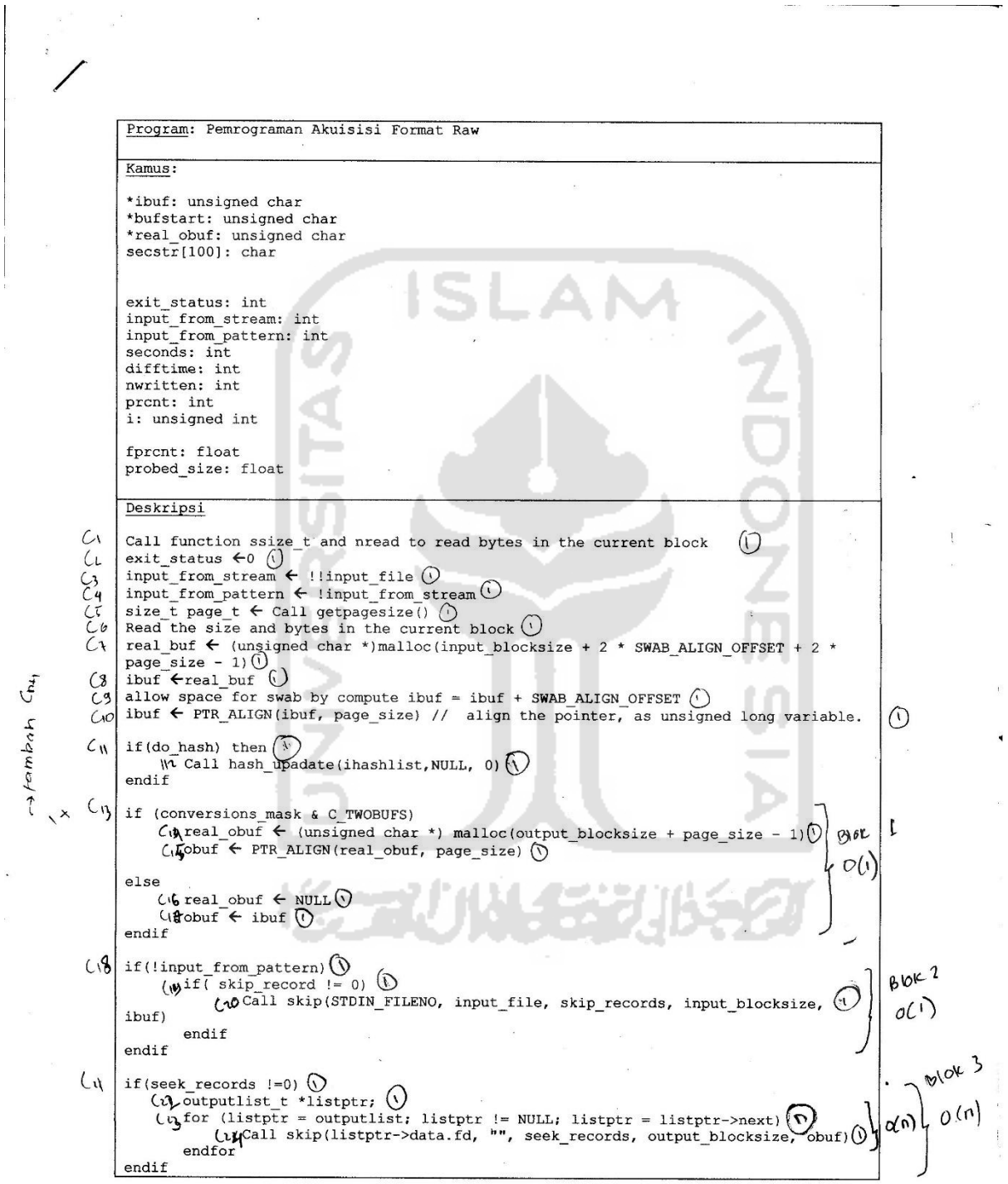
Ridlo, I. A. (2017). Panduan pembuatan flowchart. *Fakultas Kesehatan Masyarakat*, *11*(1), 1–27.

Souli, J. (2007). *C++ Language Tutorial - C++ Documentation*. Retrieved from http://www.cplusplus.com/doc/tutorial/

Vandeven, S. (2014). Forensic Images: For Your Viewing Pleasure. *SANS Institute*, *3*(2), 38. Retrieved from https://www.sans.org/reading-room/whitepapers/forensics/forensic-images-viewing-pleasure-35447

LAMPIRAN

Lampiran A



Program: Pemrograman Akuisisi Format Raw

Kamus:

```
*ibuf: unsigned char
*bufstart: unsigned char
*real_obuf: unsigned char
secstr[100]: char

exit_status: int
input_from_stream: int
input_from_pattern: int
seconds: int
difftime: int
nwritten: int
prcnt: int
i: unsigned int

fprcnt: float
probed_size: float
```

Deskripsi

```
C1  Call function ssize_t and nread to read bytes in the current block
C2  exit_status ←0
C3  input_from_stream ← !!input_file
C4  input_from_pattern ← !input_from_stream
C5  size_t page_t ← Call getpagesize()
C6  Read the size and bytes in the current block
C7  real_buf ← (unsigned char *)malloc(input_blocksize + 2 * SWAB_ALIGN_OFFSET + 2 *
    page_size - 1)
C8  ibuf ←real_buf
C9  allow space for swab by compute ibuf = ibuf + SWAB_ALIGN_OFFSET
C10 ibuf ← PTR_ALIGN(ibuf, page_size) //  align the pointer, as unsigned long variable.
C11 if(do_hash) then
        Call hash_upadate(ihashlist,NULL, 0)
    endif
C13 if (conversions_mask & C_TWOBUFS)
        real_obuf ← (unsigned char *) malloc(output_blocksize + page_size - 1)
        obuf ← PTR_ALIGN(real_obuf, page_size)
    else
        real_obuf ← NULL
        obuf ← ibuf
    endif
C18 if(!input_from_pattern)
        if( skip_record != 0)
            Call skip(STDIN_FILENO, input_file, skip_records, input_blocksize,
    ibuf)
        endif
    endif
C14 if(seek_records !=0)
        outputlist_t *listptr;
        for (listptr = outputlist; listptr != NULL; listptr = listptr->next)
            Call skip(listptr->data.fd, "", seek_records, output_blocksize, obuf)
        endfor
    endif
```

tambah C12

1x

BLOK 1 O(1)

BLOK 2 O(1)

BLOK 3 O(n) O(n)

```
if(max_records == 0)
    Call quit(exit_status);
endif

if(input_from_pattern) then
    Call replicate_pattern(pattern, ibuf, input_blocksize)
    nread ← n_bytes_read ← input_blocksize
endif

while (1)
    if(do_status && w_full % update_thresh == 0 && w_full != 0)
        off_t total_bytes ← w_full * input_blocksize
        off_t total_mb ← total_bytes / 1048576
        if (probe == PROBE_NONE OR probe_soze == 0)
            output(stderr, "\r%llu blocks (%lluMb) written.",w_full,
total_mb)
        else
            time_t curr_time ← time(NULL)
            seconds ← Read by compute call difftime(curr_time, start_time)
            off_t probed_mb ← probed_size/1048576
            fprcnt ← total_bytes/probed_size
            fprcnt_remainging ← 1.0 - fprcnt
            prcnt ← fprcnt * 100
            seconds_remaining ← seconds * (fprcnt_remaining / fprnct)

            Call time_left(secstr, sizeof secstr, seconds_remaining)
            output(stderr, "\r[%d%% of %lluMb] %llu blocks (%lluMb) written.
              %s",prcnt, probed_mb, w_full, total_mb, secstr)
        endif
    endif

    if(r_partial + r_full >= max_records)
        break
    endif

    if(!input_from_pattern)
        if ((conversions_mask & C_SYNC) && (conversions_mask & C_NOERROR))
            Call memset(ibuf,(conversions_mask & (C_BLOCK | C_UNBLOCK)) ? '
              ' : '\0',input_blocksize)
        nread ← safe_read(STDIN_FILENO, ibuf, input_blocksize))
        endif
    endif
    if(nread==0)
        break
    endif

    if(nread < 0 && !input_from_pattern)
        Call syscall_error_noexit(input_file)
        if(conversions_mask & C_NOERROR)
            Call print_stats()
            /* Seek past the bad block if possible. */
            Call lseek(STDIN_FILENO, (off_t) input_blocksize, SEEK_CUR)
            if(conversions_mask & C_SYNC)
                /* Replace the missing input with null bytes and
                proceed normally.  */
                // EXPERIMENTAL: let's try re-zeroing this buffer
                Call memset (ibuf,(conversions_mask & (C_BLOCK |
                C_UNBLOCK)) ? ' ' : '\0',input_blocksize)
                nread ← 0
            else
                continue
```

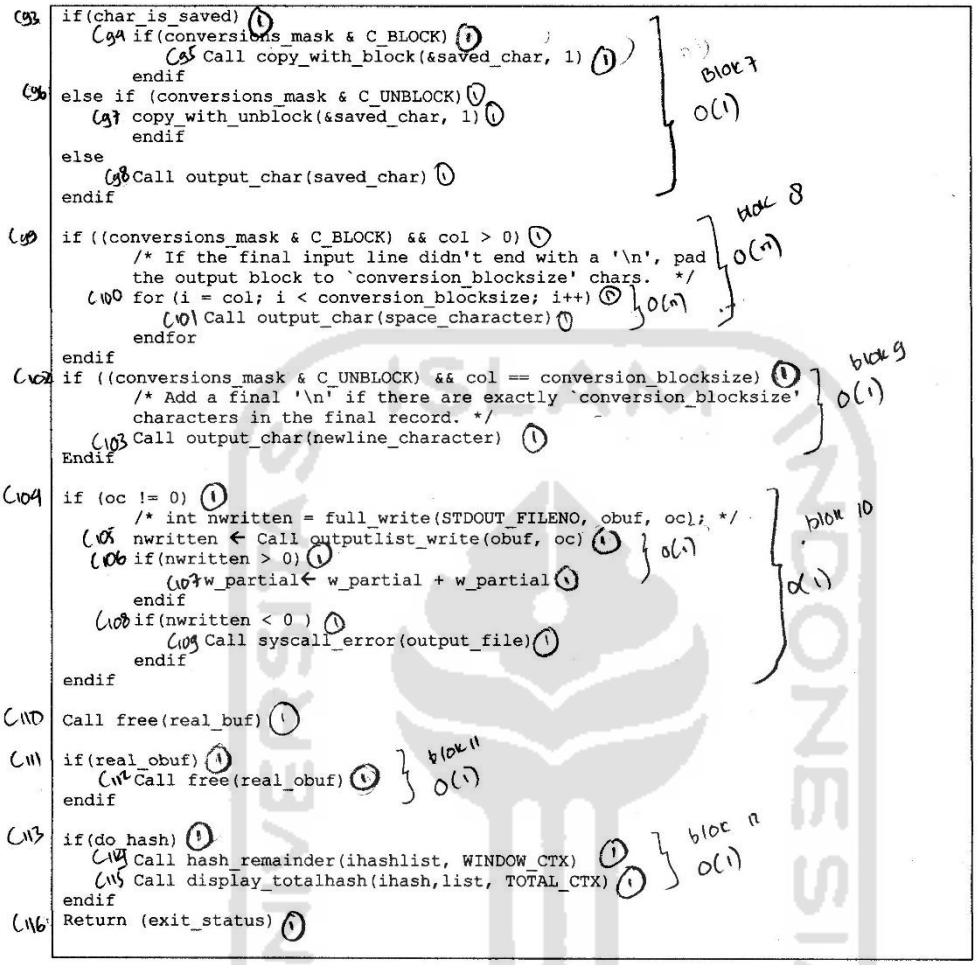
if A = 1

if B = 0

if C =

Block 6 = C30 - C92

= O(n)


C93 if(char_is_saved) 1
C94 if(conversions_mask & C_BLOCK) 1
C95 Call copy_with_block(&saved_char, 1) 1
endif
C96 else if (conversions_mask & C_UNBLOCK) 1
C97 copy_with_unblock(&saved_char, 1) 1
endif
else
C98 Call output_char(saved_char) 1
endif
Blok 7
O(1)
C99 if ((conversions_mask & C_BLOCK) && col > 0) 1
/* If the final input line didn't end with a '\n', pad
the output block to `conversion_blocksize' chars. */
C100 for (i = col; i < conversion_blocksize; i++) n
C101 Call output_char(space_character) 1
endfor
endif
Blok 8
O(n)
O(n)
C102 if ((conversions_mask & C_UNBLOCK) && col == conversion_blocksize) 1
/* Add a final '\n' if there are exactly `conversion_blocksize'
characters in the final record. */
C103 Call output_char(newline_character) 1
Endif
blok 9
O(1)
C104 if (oc != 0) 1
/* int nwritten = full_write(STDOUT_FILENO, obuf, oc); */
C105 nwritten ← Call outputlist_write(obuf, oc) 1
C106 if(nwritten > 0) 1
C107 w_partial← w_partial + w_partial 1
endif
C108 if(nwritten < 0) 1
C109 Call syscall_error(output_file) 1
endif
endif
O(n)
blok 10
O(1)
C110 Call free(real_buf) 1
C111 if(real_obuf) 1
C112 Call free(real_obuf) 1
endif
blok 11
O(1)
C113 if(do_hash) 1
C114 Call hash_remainder(ihashlist, WINDOW_CTX) 1
C115 Call display_totalhash(ihash,list, TOTAL_CTX) 1
endif
blok n
O(1)
C116 Return (exit_status) 1

Lampiran B

Aplikasi: Pemrograman Akuisisi Format AFF

Kamus:

```
ignore_missing: static int //static berarti variabel tersimpan pada memori meski
fungsi sudah terlewati, nilai variabel biasanya dihapus setelah fungsi berjalan.

do_copy: static int
copy_recursive: int
num: int
argc: int
x_copy: int

AffReader_s *r: struct
AffWriter_s *w: struct
AffNode_s *r_root: struct
AffNode_s *r_node: struct
copy_nodes_arg arg: struct
stat stat_fb: struct


*src_kpath: const char // * berarti variabel src_kpath hanya menyimpan alamat
(sebagai pointer) tidak menyimpan data
*dst_kpath: const char
*status: const char
*list_fname: const char
*src_fname: const char
*dst_fname: const char
*list1_fname: const char
*list2_fname: const char

*fargv[2]: char
buf[32768]: char
**argv: char
*p: char
*tmp_fname: char
```

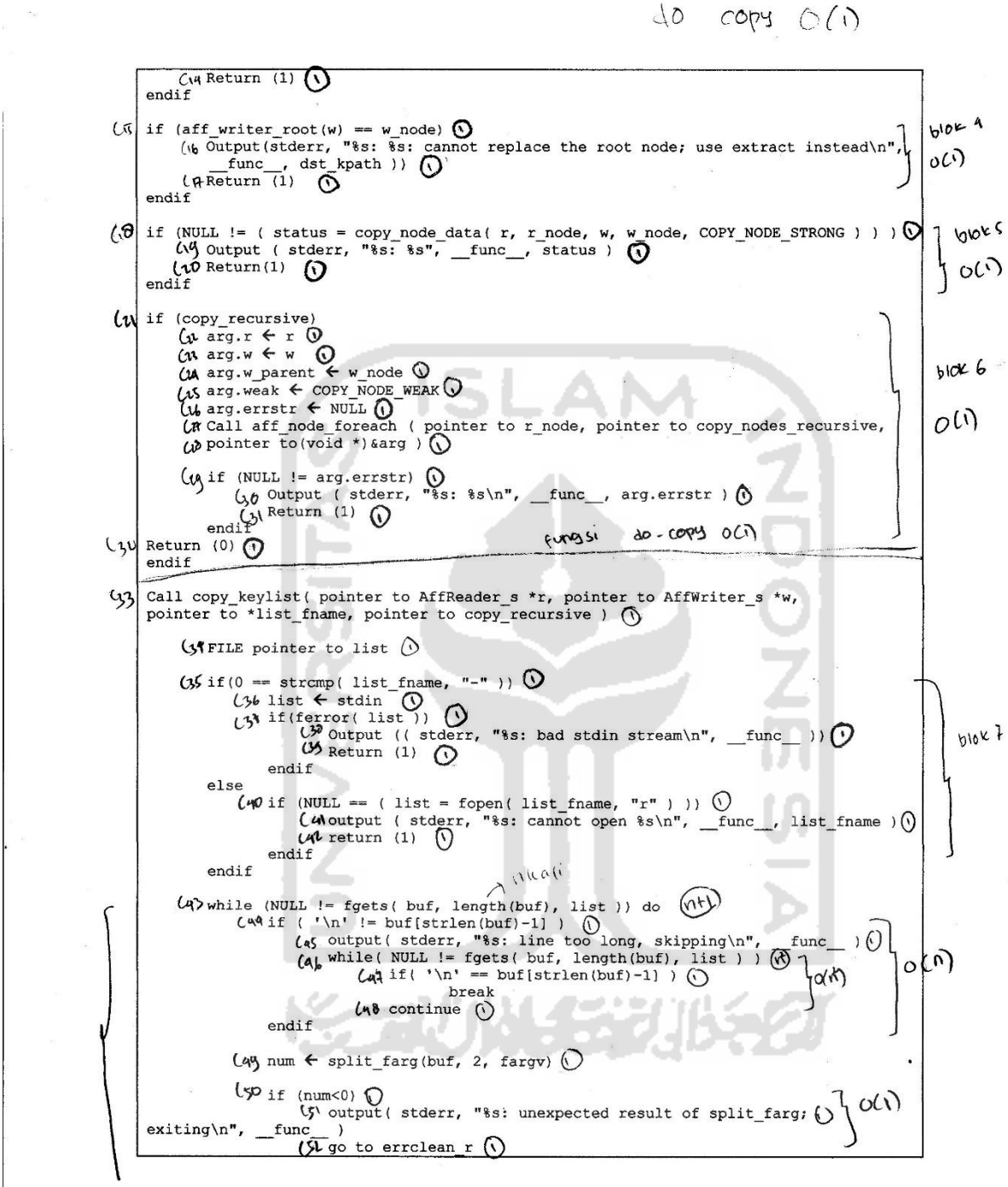
Deskripsi:

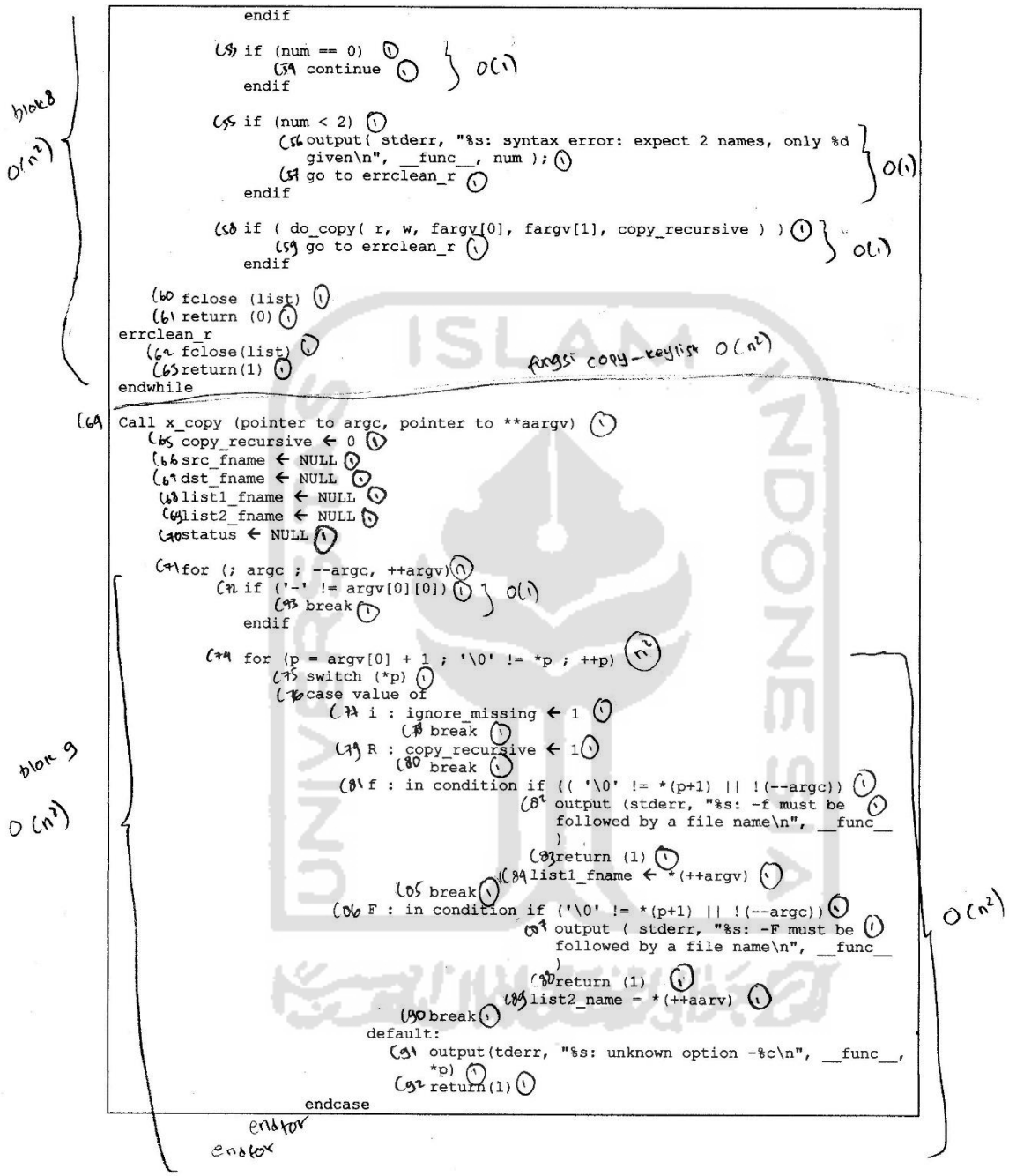
```
C1  Ignore_missing ← 0 (1)
C2  Call do_copy( pointer to AffReader_s *r, pointer to AffWriter_s *w,
            *src_kpath, *dst_kpath, copy_recursive (1)

C3  AffNode_s *r_root ← aff_reader_root(r) (1)

C4  AffNode_s *r_node ← lookup_path(r, r_root, src_kpath) (1)

C5  if(NULL == r_node) (1)
        C6 Output (stderr, "%s: [%s]: cannot read node\n", __func__, src_kpath) (1)
        C7 Return (!ignore_missing) (1)
    endif

C8  if(aff_reader_root(r)== r_node) (1)
        C9 Output (stderr, "%s: %s: cannot copy the root node; use extract instead\n",
            func__, src_kpath )) (1)
        C10 Return (1) (1)
    endif

C11 AffNode_s *w_node ← aff_writer_mkpath(w, aff_writer_root(w),dst_kpath) (1)

C12 if (NULL == w_node) (1)
        C13 Output ( stderr, "%s: cannot write to key [%s]: %s\n", __func__, dst_kpath,
            aff_writer_errstr( w ) )) (1)
```

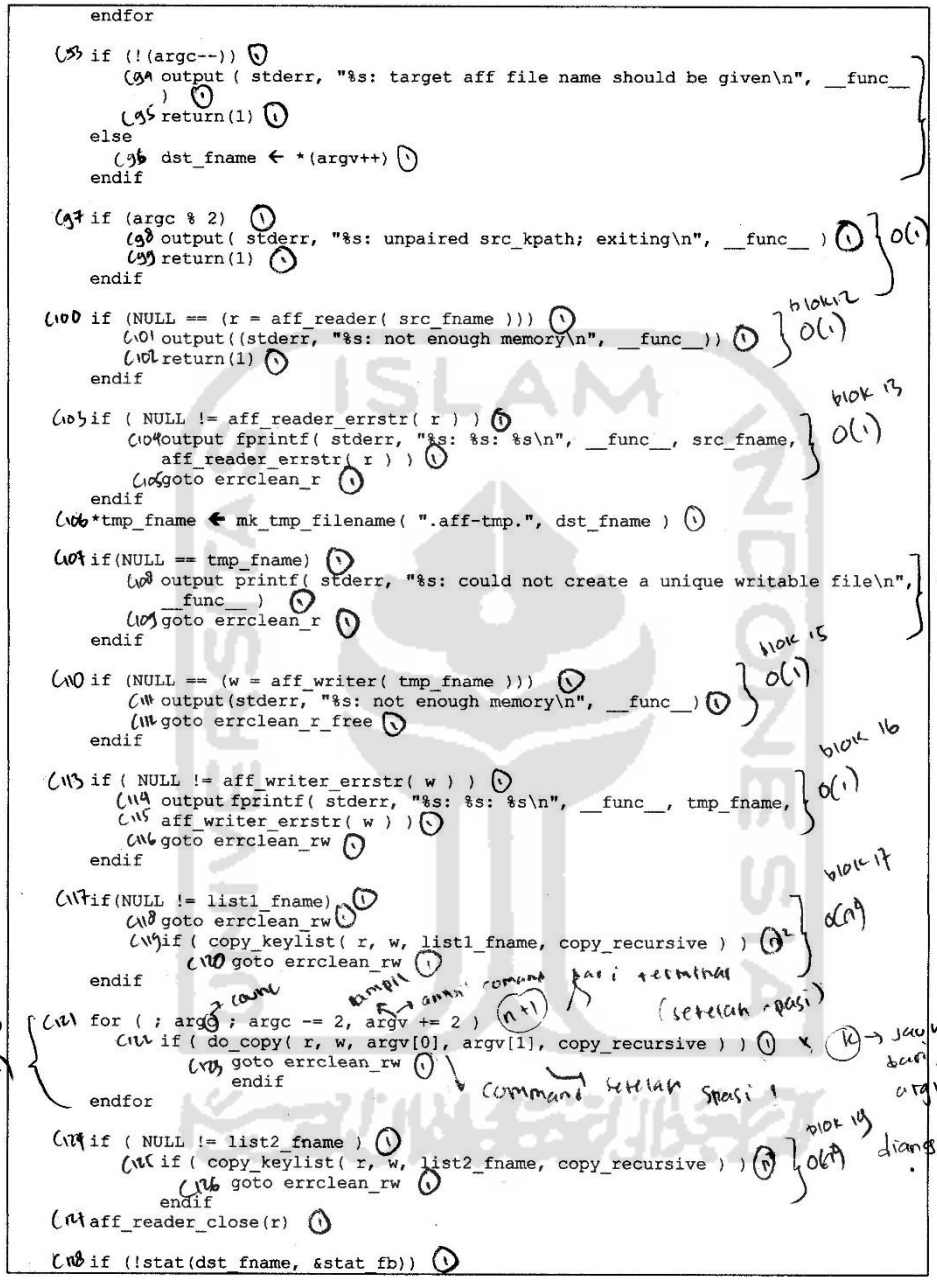
blok 1 O(1)

blok 2 O(1)

blok 3 O(1)


do copy O(1)
Return (1)
endif
if (aff_writer_root(w) == w_node)
Output(stderr, "%s: %s: cannot replace the root node; use extract instead\n", __func__, dst_kpath))
Return (1)
endif
blok 4
O(1)
if (NULL != (status = copy_node_data(r, r_node, w, w_node, COPY_NODE_STRONG)))
Output (stderr, "%s: %s", __func__, status)
Return(1)
endif
blok 5
O(1)
if (copy_recursive)
arg.r ← r
arg.w ← w
arg.w_parent ← w_node
arg.weak ← COPY_NODE_WEAK
arg.errstr ← NULL
Call aff_node_foreach (pointer to r_node, pointer to copy_nodes_recursive,
pointer to(void *)&arg)
if (NULL != arg.errstr)
Output (stderr, "%s: %s\n", __func__, arg.errstr)
Return (1)
endif
blok 6
O(1)
Return (0)
endif
fungsi do-copy O(1)
Call copy_keylist(pointer to AffReader_s *r, pointer to AffWriter_s *w,
pointer to *list_fname, pointer to copy_recursive)
FILE pointer to list
if(0 == strcmp(list_fname, "-"))
list ← stdin
if(ferror(list))
Output ((stderr, "%s: bad stdin stream\n", __func__))
Return (1)
endif
else
if (NULL == (list = fopen(list_fname, "r")))
output (stderr, "%s: cannot open %s\n", __func__, list_fname)
return (1)
endif
endif
blok 7
while (NULL != fgets(buf, length(buf), list)) do
n+1
if ('\n' != buf[strlen(buf)-1])
output(stderr, "%s: line too long, skipping\n", __func__)
while(NULL != fgets(buf, length(buf), list))
n
if('\n' == buf[strlen(buf)-1])
break
continue
endif
n
O(n)
num ← split_farg(buf, 2, fargv)
if (num<0)
output(stderr, "%s: unexpected result of split_farg; exiting\n", __func__)
go to errclean_r
O(1)


endif
if (num == 0)
continue
endif
O(1)
if (num < 2)
output(stderr, "%s: syntax error: expect 2 names, only %d given\n", __func__, num);
go to errclean_r
endif
O(1)
if (do_copy(r, w, fargv[0], fargv[1], copy_recursive))
go to errclean_r
endif
O(1)
fclose (list)
return (0)
errclean_r
fclose(list)
return(1)
endwhile
blok8
O(n²)
fungsi copy_keyfile O(n²)
Call x_copy (pointer to argc, pointer to **aargv)
copy_recursive ← 0
src_fname ← NULL
dst_fname ← NULL
list1_fname ← NULL
list2_fname ← NULL
status ← NULL
for (; argc ; --argc, ++argv)
if ('-' != argv[0][0])
break
endif
O(1)
for (p = argv[0] + 1 ; '\0' != *p ; ++p)
n²
switch (*p)
case value of
i : ignore_missing ← 1
break
R : copy_recursive ← 1
break
f : in condition if (('\0' != *(p+1) || !(--argc))
output (stderr, "%s: -f must be followed by a file name\n", __func__)
return (1)
list1_fname ← *(++argv)
break
F : in condition if ('\0' != *(p+1) || !(--argc))
output (stderr, "%s: -F must be followed by a file name\n", __func__)
return (1)
list2_name = *(++aarv)
break
default:
output(tderr, "%s: unknown option -%c\n", __func__, *p)
return(1)
endcase
endfor
endfor
blok 9
O(n²)
O(n²)



the number of argument

argv → vector of argument

```
C129  if (NULL == (r = aff_reader( dst_fname ))) ①
C130    output (stderr, "%s: not enough memory\n", __func__) ①
C131    goto errclean_w ①
      endif

C132  if(NULL != aff_reader_errstr(r)) ①
C133    output fprintf( stderr, "%s: %s\n", __func__,
          aff_reader_errstr(r))
C134    goto errclean_rw ①
      endif

C135  arg.r ← r ①
C136  arg.w ← w ①
C137  arg.w_parent ← aff_writer_root(w) ①
C138  arg.w ← COPY_NODE_WEAK ①
C139  arg.errstr ← NULL ①

C140  CALL aff_node_foreach ( aff_reader_root( r ), copy_nodes_recursive,
        (void *)&arg ) ①

C141  if (NULL != arg.errstr) ①
C142    output ( stderr, "%s: %s\n", __func__, arg.errstr ) ①
C143    goto errclean_rw ①
      endif
C144  CALL aff_reader_close(r) ①
    endif

C145  if ( NULL != ( status = aff_writer_close(w))) ①
C146    output ( stderr, "%s: %s: %s\n", __func__, tmp_fname, status ) ①
C147    goto errclean_file ①
    endif
C148  if ( rename( tmp_fname, dst_fname ) ) ①
C149    CALL perror(dst_fname) ①
C150    output( stderr, "%s: output is saved to %s\n", __func__, tmp_fname ) ①
C151    goto errclean_free ①
    endif

C152  CALL free(tmp_fname) ①
C153  return(0) ①

errclean_rw:
C154  CALL aff_reader_close(r) ①
Errclean_w:
C155  CALL aff_writer_close(w)
Errclean_file:
C156  If(remove(tmp_fname) ①
C157    CALL(perror(tmp_fname) ①
Errclean_r_free:
C158  CALL free(tmp_fname) ①
Errclean_r:
C159  Aff_reader_close(r) ①
C160  Return(1) ①
```

C161 in the h_copy function ①

```
C162  Output ("Usage:\n"
            "lhpc-aff cp [-iR] [-f <list>] [-F <list>] <src-file> <dst-file>\n"
            "\t\t[<src-kpath> <dst-kpath>] ...\n"
            "Copy data from <src-kpath> of <src-file> to <dst-kpath> of <dst-
file>:\n"
            "<dst-file>[<dst-kpath>]  <--  <src-file>[<src-kpath>].\n"
            "Keys <src-kpath> and <dst-kpath> cannot be root nodes, use 'extract'
instead.\n"
```

blok 20 O(1)

blok 21 O(1)

blok 22 O(1)

blok 23 O(1)

blok 24 O(1)

blok 27 O(1)

Angsi x_copy $O(n^2)$

```
            "New data replaces old data. If <dst-file> does not exist, it will be
created.\n"
            "If <src-file> and <dst-file> are the same, nodes to copy will be put
first,\n"
            "and then the rest of file will be reinserted.\n"
            "Options:\n"
            "\t-i\tignore error if <src-keypath> do not exist\n"
            "\t-R\tcopy data sub-entries recursively.\n"
            "\t-f <pre-list>\n\t\texecute copy instructions in file <pre-list>
BEFORE\n"
            "\t\tprocessing command line list\n"
            "\t-F <post-list>\n\t\texecute copy instructions in file <pre-list>
AFTER\n"
            "\t\tprocessing command line list\n"
            "\t\teach line of <list> is a separate insert instruction,\n"
            "\t\tand it must have at least two names, in order:\n"
            "\t\t<src-kpath> <dst-kpath>\n"
            "\t\tparameter to only one of -f, -F options can be '-' (stdin).\n"
```