BAB 1

PENDAHULUAN ALGORITMA DAN PEMROGRAMAN

Leon A. Abdillah

A. Definisi dan Ruang Lingkup Algoritma dan Pemrograman

Algoritma dan pemrograman merupakan landasan ilmu komputer dan teknologi informasi. Algoritma mewakili urutan langkah-langkah yang terbatas dan terdefinisi dengan baik yang dirancang untuk menyelesaikan masalah tertentu atau melakukan tugas tertentu (Cormen et al., 2022). Langkah-langkah ini harus jelas, tidak ambigu, dan dapat dieksekusi dalam waktu yang wajar, seringkali mematuhi kriteria seperti kepastian, spesifikasi input/output, efektivitas, dan keterbatasan (Knuth, 1997). Algoritma ada secara independen dari bahasa pemrograman apa pun, mewujudkan solusi logis yang dapat diimplementasikan di berbagai paradigma komputasi (Sedgewick & Wayne, 2011).

Sebaliknya, pemrograman melibatkan penerjemahan algoritma menjadi instruksi yang dapat dieksekusi untuk komputer (Cormen et al., 2022). Penerjemahan ini terjadi melalui bahasa pemrograman, yang menyediakan sintaksis, semantik, dan aturan yang memungkinkan manusia untuk menyampaikan

instruksi yang tepat kepada mesin (Aho et al., 2012). Dalam praktiknya, pemrograman mencakup penulisan kode, pengujian kebenaran, debugging kesalahan, dan pemeliharaannya untuk efisiensi dan skalabilitas (Brooks, 1996).

Cakupan algoritma dan pemrograman meluas jauh melampaui komputasi. Algoritma mendasari aktivitas sehari-hari seperti resep masakan, aturan manajemen lalu lintas, dan strategi pemecahan masalah manual (Polya, 2014). Dalam komputasi, algoritma mendorong fungsi-fungsi penting dalam pemrosesan data, kecerdasan buatan, keamanan siber, rekayasa perangkat lunak, dan banyak lagi (Russell & Norvig, 2010). Pemrograman mengotomatiskan algoritma ini, memungkinkan solusi yang terukur untuk masalah-masalah kompleks.

Dalam komputasi modern, algoritma dan pemrograman mendorong inovasi di berbagai aplikasi seluler, layanan web, simulasi ilmiah, dan sistem cerdas (Goodfellow et al., 2016). Hampir setiap kemajuan teknologi bergantung pada algoritma yang efisien dan program yang tangguh. Menguasai dasar-dasar ini tetap penting bagi mahasiswa dan profesional di bidang seperti AI, ilmu data, dan pengembangan perangkat lunak.

B. Dasar-dasar Pemrograman

Pemrograman melibatkan penulisan instruksi dalam bahasa seperti Python agar komputer dapat menjalankan tugas. Pemrograman bergantung pada algoritma, prosedur pemecahan masalah langkah demi langkah, di mana sebuah program mengimplementasikan algoritma menggunakan sintaks, struktur data, dan alur kontrol untuk eksekusi yang efisien. Dengan demikian, program menerjemahkan logika algoritmik abstrak

menjadi kode yang dapat dijalankan. Paradigma pemrograman utama membentuk proses ini: imperatif (urutan perintah, misalnya, C), deklaratif (berorientasi tujuan, misalnya, SQL), berorientasi objek (berbasis kelas, misalnya, Java), dan fungsional (fungsi seperti matematika, misalnya, Haskell). Setiap paradigma cocok untuk aplikasi tertentu, mulai dari perangkat lunak sistem hingga analisis data.

1. Definisi Pemrograman

Pemrograman adalah proses merancang, menulis, menguji, dan memelihara instruksi yang dikenal sebagai kode dalam bahasa pemrograman untuk memungkinkan komputer melakukan tugas-tugas tertentu atau memecahkan masalah (Abdillah, 2009b, 2009c; Scherer & Siddiq, 2020). Proses ini mengubah logika yang dapat dibaca manusia menjadi urutan yang dapat dieksekusi mesin, seringkali dimulai dengan perancangan algoritma sebelum implementasi (Abdillah, 2013). Aktivitas mendasar ini mendukung pengembangan perangkat lunak di berbagai bidang, dari analisis data hingga inovasi digital.

2. Hubungan Algoritma dan Pemrograman

Algoritma dan program secara intrinsik terkait dalam komputasi (Grigas, 1994): algoritma adalah prosedur abstrak langkah demi langkah untuk memecahkan masalah, yang independen dari mesin apa pun, sedangkan program adalah realisasi konkretnya dalam bahasa pemrograman tertentu, yang dapat dieksekusi oleh computer (Abdillah, 2016). Algoritma menyediakan cetak biru logis; program mengimplementasikannya dengan sintaks, variabel, dan struktur kontrol, menjembatani teori dan praktik. Hubungan ini mendasar,

karena program yang efektif berasal dari algoritma yang dirancang dengan baik, yang memungkinkan komputasi efisien di berbagai aplikasi.

3. Program sebagai Implementasi dari Algoritma

Sebuah program mewakili implementasi praktis dari sebuah algoritma, mengubah langkah-langkah abstraknya yang tidak bergantung pada bahasa pemrograman menjadi kode yang dapat dieksekusi menggunakan sintaks, tipe data, dan konstruksi bahasa pemrograman tertentu. Penerjemahan ini melibatkan penyempurnaan algoritma dengan variabel, perulangan, kondisi, dan fungsi untuk memastikan efisiensi komputasi dan kebenaran pada perangkat keras. Implementasi yang efektif menjembatani desain teoretis dan aplikasi dunia nyata, memungkinkan pemecahan masalah dalam sistem perangkat lunak.

4. Gambaran Umum Paradigma Pemrograman

Paradigma pemrograman mewakili gaya fundamental untuk menyusun kode dan memecahkan masalah, masing-masing menekankan prinsip yang berbeda. Paradigma imperatif berfokus pada instruksi langkah demi langkah yang eksplisit dan perubahan status, seperti dalam pemrograman prosedural (misalnya, C). Paradigma berorientasi objek (Abdillah, 2009a) mengatur kode di sekitar objek dan kelas untuk modularitas dan pewarisan (misalnya, Java). Paradigma fungsional memperlakukan komputasi sebagai fungsi matematika, menghindari status yang dapat diubah (misalnya, Haskell). Paradigma deklaratif menentukan hasil yang diinginkan tanpa detail alur kontrol (misalnya, SQL). Paradigma-paradigma ini

memandu implementasi algoritma, meningkatkan daya ekspresi dan kemudahan pemeliharaan.

C. Algoritma dan Pemrograman dalam Ilmu Komputer

1. Peran Algoritma dalam Ilmu Komputer

Algoritma merupakan landasan ilmu komputer, menyediakan prosedur langkah demi langkah yang tepat untuk memecahkan masalah komputasi secara efisien dan sistematis. Algoritma mendukung aktivitas inti seperti pemrosesan data, optimasi, pencarian, dan pengambilan keputusan, memungkinkan solusi yang dapat diskalakan di berbagai bidang seperti AI, pembelajaran mesin, kriptografi, dan perutean jaringan. Dengan menganalisis kompleksitas waktu dan ruang, algoritma memastikan kinerja yang optimal dalam penggunaan sumber daya, membedakan perangkat lunak yang efektif dari yang tidak efisien. Landasan teoritisnya mendorong inovasi, mengubah pemecahan masalah abstrak menjadi kemajuan komputasi praktis.

2. Keterampilan Pemrograman dalam Pengembangan Perangkat Lunak

Keterampilan pemrograman merupakan landasan pengembangan perangkat lunak, memungkinkan para profesional untuk menerjemahkan algoritma ke dalam kode fungsional, mengotomatiskan proses kompleks, dan berinovasi dalam solusi di bidang ilmu komputer. Keterampilan ini sangat penting untuk peran yang mencakup desain, pengujian, dan pemeliharaan, yang secara langsung memengaruhi hasil proyek dan kualitas perangkat lunak. Karena pengembangan perangkat lunak tetap

merupakan upaya yang padat karya manusia, pemrograman yang mahir memastikan implementasi yang efisien, mengurangi kesalahan, dan memenuhi harapan pemangku kepentingan di seluruh siklus hidup proyek. Studi menekankan pemrograman sebagai kompetensi inti yang penting dalam kerangka kerja terpadu seperti UComGSP (Assyne et al., 2022), yang vital untuk perangkat lunak berkualitas tinggi di masyarakat modern.

3. Algoritma sebagai Dasar Sistem Komputasi

Algoritma berfungsi sebagai tulang punggung fundamental sistem komputasi, menyediakan instruksi langkah demi langkah yang tepat yang memungkinkan mesin untuk memproses data, memecahkan masalah, dan menjalankan tugas secara efisien. Algoritma mendasari setiap aspek ilmu komputer, dari perhitungan sederhana hingga simulasi kompleks dalam kecerdasan buatan dan sistem terdistribusi. Tanpa algoritma yang tangguh, komputasi modern, dari sistem operasi hingga infrastruktur cloud, akan kekurangan efisiensi dan skalabilitas yang penting untuk aplikasi dunia nyata. Penelitian menyoroti peran algoritma dalam menentukan kompleksitas komputasi dan kinerja di berbagai domain. Algoritma mencakup beragam aplikasi, termasuk analisis sentimen untuk penambangan opini dan jarak Levenshtein untuk pencocokan string dalam tugas pengarsipan, pengambilan, dan pengikisan data.

Algoritma analisis sentimen (Guspita & Abdillah, 2021; Wijaya & Abdillah, 2023) seperti Naïve Bayes, SVM, dan Random Forest mengklasifikasikan ulasan pengguna. Algoritma Jarak Levenshtein (Vidyarsih et al., 2016) meningkatkan pengambilan dalam sistem pengarsipan dengan menghitung

operasi pengeditan minimum (penyisipan, penghapusan, penggantian) untuk memperbaiki kesalahan ketik, meningkatkan efisiensi pencarian dalam manajemen dokumen. Dalam konteks pengikisan (scraping) data, algoritma ini mendukung mesin pencari artikel (Josi et al., 2014) dengan mencocokkan kueri dokumen di berbagai situs, mengintegrasikan pengikisan web untuk pengumpulan data secara real-time.

D. Berpikir Komputasional dan Pemecahan Masalah

1. Konsep Berpikir Komputasional

Berpikir komputasional atau computational thinking (CT) mewakili pendekatan pemecahan masalah mendasar dalam ilmu komputer, yang melibatkan dekomposisi, pengenalan pola, abstraksi, dan desain algoritma untuk merumuskan solusi yang dapat dieksekusi oleh komputer. Dipelopori oleh Jeannette Wing pada tahun 2006 (Wing, 2006), CT melampaui pemrograman, menumbuhkan keterampilan yang penting untuk mengatasi tantangan dunia nyata yang kompleks di berbagai disiplin ilmu. CT membekali individu untuk berpikir logis dan efisien, seperti halnya membaca, menulis, dan berhitung di era digital.

2. Dekomposisi, Abstraksi, Pengenalan Pola, dan Berpikir Algoritmik

Dekomposisi, abstraksi, pengenalan pola, dan pemikiran algoritmik membentuk pilar inti dari pemikiran komputasional, yang memungkinkan pemecahan masalah secara sistematis dalam ilmu komputer. Dekomposisi memecah masalah kompleks menjadi sub-masalah yang dapat dikelola, sementara abstraksi menyaring detail yang tidak relevan untuk fokus pada elemen-

elemen penting. Pengenalan pola mengidentifikasi kesamaan di antara sub-masalah untuk solusi yang dapat digunakan kembali, dan pemikiran algoritmik menciptakan instruksi langkah demi langkah yang tepat untuk dieksekusi. Studi (Korte, 2025) mengkonfirmasi bahwa keterampilan ini meningkatkan efisiensi kognitif dan inovasi di berbagai konteks pendidikan dan profesional.

3. Menerjemahkan Masalah Dunia Nyata ke dalam Solusi Algoritmik

Menerjemahkan masalah dunia nyata ke dalam solusi algoritmik adalah landasan pemikiran komputasional, di mana skenario kompleks dimodelkan sebagai rangkaian langkah terstruktur untuk eksekusi komputasi. Proses ini melibatkan identifikasi input, pendefinisian operasi, dan penentuan output untuk memastikan efisiensi dan skalabilitas. Misalnya, mengoptimalkan rute pengiriman (Dijkstra, 1959) atau memprediksi kegagalan peralatan bergantung pada algoritma grafik dan model pembelajaran mesin yang berasal dari kendala praktis. Proses penerjemahan ini berperan dalam menjembatani teori dan aplikasi, meningkatkan pemecahan masalah di berbagai bidang teknik dan ilmu data.

E. Evolusi Algoritma dan Bahasa Pemrograman

1. Perkembangan Historis Algoritma

Perkembangan historis algoritma memperoleh bentuk sistematis melalui matematikawan Persia Muhammad ibn Musa al-Khwarizmi (sekitar 780–850 M). Beliau hidup pada Zaman Keemasan Peradaban Islam (dari abad ke-8 hingga ke-13)

(Elamin, 2024), yang risalahnya pada abad ke-9, Al-Kitab al-Mukhtasar fi Hisab al-Jabr wal-Muqabala, memperkenalkan metode aljabar sebagai prosedur langkah demi langkah untuk menyelesaikan persamaan. Karyanya tentang angka memformalkan algoritma aritmatika, menyebar melalui terjemahan Latin sebagai Algoritmi de numero Indorum, melahirkan istilah "*algoritma*". Para sarjana abad pertengahan membangun fondasi ini, yang mengarah pada kemajuan abad ke-17 seperti interpolasi Newton dan teori graf Euler, yang berpuncak pada mesin Turing abad ke-20 yang mendigitalisasi komputasi algoritmik.

Saat ini algoritma telah memasuki era artificial intelligence (AI). AI adalah gagasan umum bahwa mesin sedang dikembangkan untuk melakukan tugas-tugas seperti pemecahan masalah dan pengambilan keputusan yang biasanya membutuhkan kecerdasan manusia (Abdillah, 2026). Linimasa ringkas perkembangan algoritma mulai dari era Al-Khwarizmi sampai dengan AI dapat dilihat pada Tabel 1.

Tabel 1. Linimasa Perkembangan Algoritma dari Al-Khwarizmi sampai Era AI

Periode	Tokoh/Instansi Utama	Keterangan
825	Al-Khwarizmi publikasikan "Al-Kitab al-Mukhtasar fi Hisab al-Jabr wal-Muqabala"	Dari transliterasi namanya lahir kata 'algorithm'.
1842	Ada Lovelace tulis algoritma pertama untuk Analytical Engine Babbage.	Wanita pertama yang menulis koding.
1936	Alan Turing perkenalkan Universal Turing Machine.	Dasar komputasi modern.
1956	Penyelenggaraan Dartmouth Work-shop berlokasi di	Istilah "Artificial Intelligence" resmi

	Dartmouth College, Hanover, New Hampshire, USA. John McCarthy ciptakan Lisp.	diperkenalkan sebagai bidang ilmu baru. Ide Lisp, bahasa pemrograman AI pertama, muncul dari diskusi AI di workshop ini.
1997	IBM Deep Blue kalahkan Kasparov dalam catur.	Tonggak AI.
2012	AlexNet revolusi deep learning (DL).	Ledakan AI.
2020-an	Large Language Model (LLM) large language seperti GPT.	Transformasi aplikasi AI di dunia nyata secara masif.

2. Evolusi Bahasa Pemrograman

Evolusi bahasa pemrograman mencakup lima generasi, yang mencerminkan kemajuan dalam abstraksi, kemampuan perangkat keras, dan paradigma pemrograman. Bahasa generasi pertama (1GL) menggunakan kode mesin biner pada tahun 1940-an untuk kontrol perangkat keras langsung. Bahasa assembly generasi kedua (2GL) (tahun 1950-an) memperkenalkan mnemonik seperti ADD dan MOV untuk instruksi simbolik. Bahasa tingkat tinggi generasi ketiga (3GL) seperti Fortran (1957), COBOL (1959), dan C (1972) memungkinkan sintaks yang dapat dibaca manusia yang dikompilasi ke kode mesin. Alat generasi keempat (4GL) seperti SQL dan MATLAB (tahun 1970-an ke atas) berfokus pada "apa" deklaratif daripada "bagaimana" prosedural, meningkatkan produktivitas spesifik domain. Bahasa generasi kelima (5GL) termasuk Prolog dan kerangka kerja AI modern menekankan logika, batasan, dan pemrosesan bahasa

alami. Perkembangan ini dapat dilihat pada tabel 2, mendorong ekosistem perangkat lunak modular dan terukur saat ini.

Tabel 2. Generasi Bahasa Pemrograman

Generasi	Periode	Karakteristik	Contoh Bahasa
1GL	1940-an	Kode biner langsung prosesor	Machine Code
2GL	1950-an	Mnemonik simbolis, assembler	Assembly (x86)
3GL	1957+	High-level, prosedural / fungsional / objek	Fortran, Lisp, COBOL, BASIC, Pascal, C, Java
4GL	1970-an+	Deklaratif, domain-specific, RAD tools	Visual Basic (1991), Borland Delphi (1995), SQL, MATLAB
5GL	1980-an+	Logika, AI, natural language	Prolog, Mercury

3. Paradigma Pemrograman Modern

Paradigma modern saat ini mendominasi lanskap pemrograman tahun 2026, memadukan pendekatan fungsional, reaktif, konkuren, dan terintegrasi AI untuk sistem yang skalabel dan tangguh. Pemrograman fungsional (Haskell, Scala, Elixir) menekankan immutabilitas, fungsi murni, dan operasi tingkat tinggi untuk menghilangkan efek samping dan memungkinkan paralelisme dalam aplikasi cloud-native. Pemrograman reaktif (RxJava, Spring WebFlux) menangani aliran data asinkron untuk UI real-time dan microservice berbasis event. Paradigma konkuren di Go (goroutine) dan Rust (async/await dengan kepemilikan) mengoptimalkan pemrosesan multi-core untuk sistem terdistribusi. Pengembangan berbasis AI

mengintegrasikan asisten AI langsung ke dalam IDE, mengotomatiskan pembuatan kode sambil mempertahankan maksud pengembang. Bahasa multi-paradigma hibrida seperti Python, TypeScript, dan Mojo mendukung transisi yang mulus antar gaya. Penelitian yang terindeks Scopus memvalidasi keunggulan paradigma ini dalam menangani kompleksitas, keamanan, dan tuntutan kinerja ekosistem berbasis AI.

F. Representasi Algoritma

1. Naratif/Deskriptif

Dalam representasi "Narasi/Deskriptif", sebuah algoritma dijelaskan dalam bahasa yang sederhana dan langkah demi langkah yang menyerupai cerita pendek atau serangkaian instruksi, sehingga mudah diakses oleh pemula yang belum terbiasa dengan diagram formal atau pseudocode. Gaya ini menekankan kejelasan dan alur logis, menggunakan bahasa alami untuk menguraikan urutan tindakan, kondisi, dan hasil, yang mendukung pemikiran algoritmik awal dalam kursus pemrograman pengantar. Penelitian pedagogis tentang pembelajaran dan visualisasi berbasis narasi menunjukkan bahwa penjelasan deskriptif yang menyerupai cerita dapat meningkatkan pemahaman dan keterlibatan, terutama ketika siswa pertama kali dihadapkan pada pemecahan masalah dan struktur kontrol seperti seleksi dan pengulangan.

> The program starts by asking the user to enter three exam scores. It adds the three scores together and divides the total by 3 to get the average. If the average is 60 or higher, the program prints "PASS"; otherwise it prints "FAIL". Then the program ends.

Gambar 1. Contoh Narasi Algoritma

2. Pseudocode

Pseudocode adalah cara ringkas dan independen bahasa untuk mendeskripsikan algoritma menggunakan pernyataan terstruktur seperti kode dalam bahasa Inggris sederhana, yang membantu pelajar fokus pada logika daripada sintaksis. Biasanya menggunakan kata kunci seperti START, INPUT, IF–THEN–ELSE, dan PRINT untuk menguraikan keputusan, perulangan, dan manipulasi data, menjadikannya alat standar dalam buku teks dan kurikulum pemrograman pengantar. Studi pendidikan tentang metode representasi algoritma menyoroti bahwa pseudocode meningkatkan perencanaan, mengurangi kesalahan terkait sintaksis, dan mendukung kolaborasi dengan menyediakan cetak biru yang jelas dan bersama sebelum kode sebenarnya ditulis.

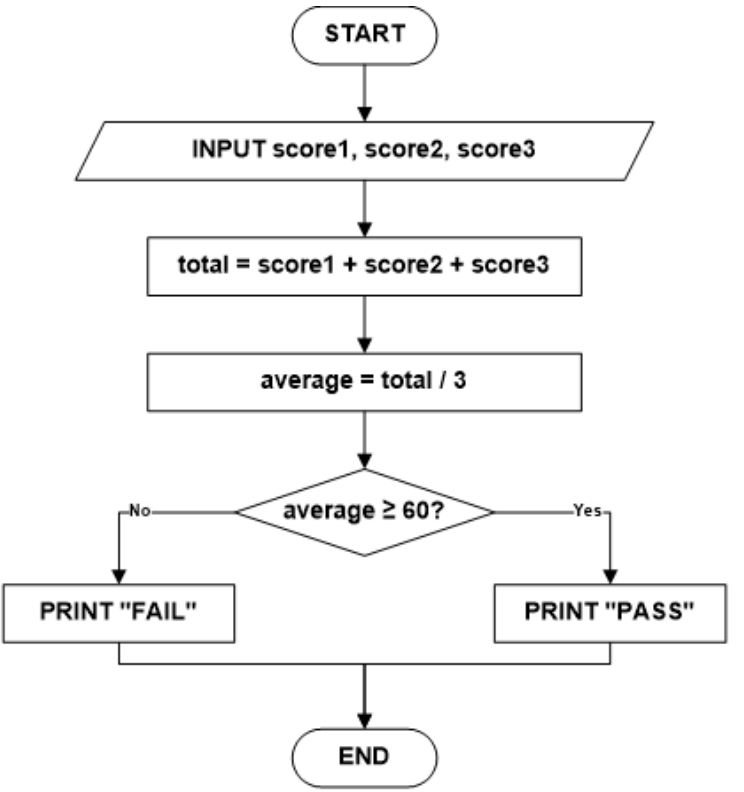
```
START
    INPUT score1, score2, score3
    total ← score1 + score2 + score3
    average ← total / 3

    IF average ≥ 60 THEN
        PRINT "PASS"
    ELSE
        PRINT "FAIL"
    ENDIF
END
```

Gambar 2. Contoh Pseudocode

3. Flowchart

Diagram alir adalah representasi visual dari suatu algoritma yang menggunakan simbol standar, seperti oval untuk awal/akhir,

persegi panjang untuk proses, dan belah ketupat untuk keputusan, yang dihubungkan oleh panah untuk menunjukkan alur kontrol. Diagram alir memungkinkan siswa untuk "melihat" struktur urutan, seleksi, dan pengulangan, yang mendukung pemahaman logika kompleks sebelum beralih ke kode. Penelitian dalam pendidikan pemrograman menunjukkan bahwa diagram alir dapat mengurangi beban kognitif dan kesalahan saat mempelajari algoritma, terutama bagi pemula, dan seringkali lebih intuitif daripada deskripsi tekstual semata. Hasil survey (Zimmermann et al., 2024) menunjukkan bahwa penggunaan diagram alur meningkatkan pemahaman, integrasi materi, dan pengetahuan secara keseluruhan. Diagram alir berfungsi sebagai jembatan dari pernyataan masalah abstrak ke solusi konkret yang dapat dieksekusi.



Gambar 3. Contoh Flowchart

G. Algoritman dan Pemrograman di Era Modern

1. Algoritma dalam Sistem Berbasis Data dan Digital

Algoritma mendukung sistem berbasis data dan digital (Balcan, 2020) dengan memproses kumpulan data yang sangat besar untuk mengekstrak wawasan, mengotomatiskan keputusan, dan mengoptimalkan operasi secara real-time. Dalam ekosistem modern seperti mesin rekomendasi (Netflix, Amazon), pengindeksan pencarian (Google), dan platform perdagangan keuangan, algoritma canggih seperti penyaringan kolaboratif, PageRank, dan jaringan saraf menganalisis pola untuk memberikan hasil yang dipersonalisasi. Sistem ini memanfaatkan pembelajaran mesin untuk kinerja adaptif, mengubah data mentah menjadi informasi yang dapat ditindaklanjuti di seluruh IoT, komputasi awan, dan kendaraan otonom. Penelitian yang diindeks Scopus menekankan peran mereka dalam skalabilitas dan efisiensi, memungkinkan analitik prediktif yang mendukung Industri 4.0 dan transformasi digital.

2. Peran Algoritma dalam AI, Big Data, dan Otomatisasi

Algoritma berfungsi sebagai inti dasar dalam AI, big data, dan otomatisasi, yang mendukung pengambilan keputusan cerdas dan optimasi proses dalam sistem modern (Al-khamees et al., 2026; Rahmani et al., 2021). Dalam AI, jaringan saraf dan algoritma pembelajaran penguatan memungkinkan pengenalan pola dan adaptasi otonom, seperti yang terlihat pada model pembelajaran mendalam yang memproses bahasa alami. Analisis big data bergantung pada algoritma terdistribusi seperti MapReduce dan PageRank untuk menangani dataset skala petabyte secara efisien, mengekstrak wawasan yang dapat

ditindaklanjuti dari informasi yang tidak terstruktur. Otomatisasi memanfaatkan algoritma optimasi (algoritma genetika, penurunan gradien) untuk otomatisasi proses robotik dan pemeliharaan prediktif, mengurangi biaya operasional di berbagai industri (Farshidi et al., 2021). Algoritma memiliki peran penting dalam pelatihan AI yang terukur dan pipeline big data real-time, yang mendorong transformasi Industri 4.0.

3. Pertimbangan Etis dan Praktis dalam Pemrograman Modern

Pemrograman modern menuntut pertimbangan etis dan praktis untuk mengurangi bias, memastikan privasi, dan mendorong akuntabilitas dalam sistem berbasis AI (Al-khamees et al., 2026). Pengembang harus mengaudit algoritma untuk memastikan keadilan, karena data pelatihan yang bias dapat melanggengkan diskriminasi dalam alat perekrutan atau pengenalan wajah—studi menunjukkan kesenjangan akurasi hingga 34% di berbagai demografi. Prinsip privasi sejak awal (privacy-by-design) mengharuskan enkripsi dan minimalisasi data sejak awal, sesuai dengan GDPR dan peraturan AI 2026 yang sedang berkembang. Secara praktis, arsitektur kode modular dan pipeline CI/CD dengan titik pemeriksaan etis memungkinkan transparansi dan deteksi bias yang cepat. Sistem kontrol versi harus mencatat keputusan algoritmik untuk keperluan audit, sementara tim pengembangan yang beragam mengurangi pemikiran yang homogen. AI yang dapat dijelaskan atau explainable AI (XAI) dan pengawasan manusia sebagai hal penting untuk penerapan yang dapat dipercaya, menyeimbangkan inovasi dengan dampak sosial.

H. Lingkungan Pengembangan Pemrograman

1. Editor Kode vs IDE

Baik editor kode maupun lingkungan pengembangan terintegrasi atau *Integrated Development Enviroment* (IDE) berfungsi sebagai komponen inti dari lingkungan pemrograman, namun keduanya berbeda secara signifikan dalam cakupan dan dukungan untuk pemecahan masalah algoritmik. Editor kode biasanya ringan, dioptimalkan untuk pengeditan teks cepat, penyorotan sintaksis, dan otomatisasi dasar, sehingga cocok untuk skrip kecil, pengeditan langsung, dan pelajar yang ingin fokus pada konstruksi bahasa inti tanpa beban alat yang berat. Sebaliknya, IDE menggabungkan editor kode dengan fasilitas tambahan seperti kompiler atau interpreter bawaan, debugging tingkat lanjut, alat refactoring, dan fitur manajemen proyek, yang dapat menyederhanakan pembuatan dan pengujian algoritma kompleks dan program yang lebih besar. Studi empiris tentang alat pengembangan perangkat lunak menunjukkan bahwa IDE dapat meningkatkan produktivitas dan deteksi kesalahan dalam tugas pemrograman terstruktur (Nathasya et al., 2019), tetapi mungkin memperkenalkan kurva pembelajaran yang lebih curam dibandingkan dengan pengaturan editor kode minimalis. Untuk mata kuliah "*Algoritma dan Pemrograman*", memulai dengan editor kode sederhana dapat membantu pemula memahami sintaks dan logika, sementara transisi bertahap ke IDE mendukung integrasi debugging dan analisis kinerja saat siswa menangani implementasi algoritma yang lebih realistis.

Tabel 3. Perbandingan Code Editor dan IDE

Aspek	Code Editor	IDE
Nature	Ringan, berfokus pada teks, startup cepat	Lingkungan yang lebih berat dan terintegrasi untuk pengembangan proyek lengkap
Tujuan utama	Menulis dan mengedit kode dengan alat bantu minimal	Pengembangan ujung-ke-ujung (meng-edit, membangun, men-debug, menguji)
Contoh umum	VS Code, Sublime Text, Vim/Neovim, Notepad++	IntelliJ IDEA, WebStorm, PyCharm, Eclipse

2. Compiler dan Intepreter

Compiler dan interpreter sama-sama merupakan prosesor bahasa yang menerjemahkan kode sumber tingkat tinggi ke dalam bentuk yang dapat dieksekusi oleh computer (Jiménez et al., 2025), tetapi keduanya berbeda dalam hal kapan dan bagaimana mereka melakukan penerjemahan ini. Compiler memproses seluruh program terlebih dahulu, menghasilkan kode mesin atau kode byte yang efisien yang dapat dijalankan berulang kali, itulah sebabnya compiler sering terintegrasi ke dalam lingkungan pengembangan modern untuk bahasa yang dikompilasi seperti C, C++, atau Java. Sebaliknya, interpreter mengeksekusi kode baris demi baris pada saat runtime, sehingga lebih mudah untuk mendeteksi dan melokalisasi kesalahan selama pengembangan, seperti pada banyak alat scripting dan pendidikan. Penelitian tentang perilaku compiler dan interpreter dalam konteks pendidikan pemrograman menyoroti bahwa keduanya dapat memengaruhi lintasan pembelajaran, konsumsi energi, dan pengalaman debugging, yang menggarisbawahi pentingnya memilih model eksekusi yang tepat untuk tugas-tugas algoritmik pengantar.

3. Tools Modern

Perangkat pengembangan modern telah berevolusi jauh melampaui editor teks sederhana dan kompiler dasar, mengintegrasikan saran yang dibantu AI (Manorat et al., 2025), ruang kerja berbasis cloud, dan lingkungan kolaboratif waktu nyata ke dalam alur kerja pemrograman. Editor kode kontemporer seperti Visual Studio Code dan IDE modern menyematkan penyelesaian kode cerdas, debugging terintegrasi, dan antarmuka kontrol versi yang menyederhanakan implementasi dan pengujian algoritma. Penggunaan tools modern ini dapat mengurangi kesalahan sintaksis, mendukung pemecahan masalah secara bertahap, dan meningkatkan keterlibatan siswa dalam kursus pengantar, terutama bila dikombinasikan dengan platform pembelajaran campuran dan lingkungan pengembangan daring. Alat-alat modern menyediakan jembatan praktis antara konsep teoretis dan praktik pengkodean langsung, sekaligus mendorong kebiasaan baik seperti desain modular, pengujian, dan refactoring.

Alat pemrograman berbasis cloud (Chauhan, 2020; Conceicao et al., 2025) dari Google, Microsoft, dan Amazon menyediakan IDE dan lingkungan yang dapat diakses melalui browser untuk pengkodean, pengujian, dan penerapan tanpa pengaturan lokal. Google Cloud Shell dan Cloud Code memungkinkan pengembangan Python/Node.js dengan integrasi yang mulus ke Compute Engine dan Cloud Run untuk aplikasi berbasis kontainer. Microsoft Visual Studio Codespaces dan Azure DevOps mendukung kolaborasi waktu nyata, pipeline CI/CD, dan proyek .NET perusahaan di seluruh cloud hybrid. Amazon AWS Cloud9 menawarkan dukungan multibahasa dengan fungsi serverless Lambda dan CodeBuild untuk build otomatis pada infrastruktur EC2/ECS.

Daftar Pustaka

Abdillah, L. A. (2009a). *Object oriented programming*. Computer Science for Education. http://blog.binadarma.ac.id/mleonaa/2009/06/26/oop/

Abdillah, L. A. (2009b). *Pemrograman II (Delphi dasar)* (4th ed.). PPP-UBD (Pusat Penerbitan dan Percetakan Universitas Bina Darma). http://blog.binadarma.ac.id/mleonaa/?p=180

Abdillah, L. A. (2009c). *Pemrograman III (Delphi database)*. PPP-UBD (Pusat Penerbitan dan Percetakan Universitas Bina Darma).

Abdillah, L. A. (2013). *Algorithms & programming*. Computer Science for Education. http://blog.binadarma.ac.id/mleonaa/teaching/programming/algorithm-and-programming-2/

Abdillah, L. A. (2016). *Programming concepts*. Computer Science for Education. https://leonabdillah.wordpress.com/teaching/programming-concepts/

Abdillah, L. A. (2026). Artificial intelligence dan machine learning di FinTech. In *FinTech*. Mega Press Nusantara.

Aho, A. V., Sethi, R., Ullman, J. D., & Lam, M. S. (2012). *Compilers: Principles, techniques, and tools* (2nd ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/compilers-principles-techniques-and-tools/P200000003472/9780133002140

Al-khamees, H. A. A., Alsmadi, M. K., Alkannad, A. A., Abugabah, A., & Abdullah, L. (2026). Advancing decision-making: A comprehensive review of intelligent systems, applications, and challenges. *Intelligent Systems with Applications*, *29*. https://www.sciencedirect.com/science/article/pii/S2667305326000062

Assyne, N., Ghanbari, H., & Pulkkinen, M. (2022). The essential competencies of software professionals: A unified competence framework. *Information and Software Technology, 151*, 107020. https://doi.org/10.1016/j.infsof.2022.107020

Balcan, M. (2020). Data-driven algorithm design. In T. Roughgarden (Ed.), *Beyond the worst-case analysis of algorithms* (pp. 1–20). Cambridge University Press.

Brooks, F. (1996). *The mythical man-month: Essays on software engineering*. Addison-Wesley.

Chauhan, A. (2020). A comparative study of cloud computing platforms. *Turkish Journal of Computer and Mathematics Education, 11*(1), 821–826. https://turcomat.org/index.php/turkbilmat/article/view/13563

Conceicao, O. D., Rema, Y. O. L., Baso, B., & Suni, G. A. (2025). Pembuatan dan deploy API di Google Cloud Platform. *Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi), 9*, 944–955. https://www.journal.lembagakita.org/index.php/jtik/article/view/3524

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). MIT Press.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269–271. https://link.springer.com/article/10.1007/BF01386390

Elamin, M. O. I. (2024). Muhammad ibn Musa al-Khwarizmi: The pioneer of algorithms and his enduring legacy in artificial intelligence. *Journal of Ecohumanism, 3*(8), 10853–10874.

Farshidi, S., Jansen, S., & Deldar, M. (2021). A decision model for programming language ecosystem selection: Seven industry case studies. *Information and Software*

Technology, *139*, 106640. https://doi.org/10.1016/j.infsof.2021.106640

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. https://mitpress.mit.edu/9780262035613/deep-learning/

Grigas, G. (1994). Algorithms and programs as different items in learning of computer programming. *Informatica*, *5*(1), 43–54. https://journals.sagepub.com/doi/epdf/10.3233/INF-1994-51-203

Guspita, Y., & Abdillah, L. A. (2021). Implementasi Naïve Bayes untuk analisis sentimen publik terhadap bentukan kabinet baru Indonesia 2019 pada media online. *BDCCS 2021*, 981–986. https://conference.binadarma.ac.id/index.php/BDCCS/article/view/2878/1131

Jiménez, E., Gordillo, A., & Calero, C. (2025). Does the compiler or interpreter version influence the energy consumption of programming languages? *Science of Computer Programming*, *243*. https://www.sciencedirect.com/science/article/abs/pii/S0167642325000097

Josi, A. J., Abdillah, L. A., & Suryayusra. (2014). Penerapan teknik web scraping pada mesin pencari artikel ilmiah. *Jurnal SISFO*, *5*(2), 159–164. https://arxiv.org/pdf/1410.5777

Knuth, D. E. (1997). *The art of computer programming* (3rd ed.). Addison-Wesley. https://www.informit.com/store/art-of-computer-programming-volume-1-fundamental-algorithms-9780201896831

Korte, S. (2025). Developing computational thinking skills in higher education through peer reflection on robotics and programming exercises. *Learning, Culture and Social Interaction, 55*. https://doi.org/10.1016/j.lcsi.2025.100947

Manorat, P., Tuarob, S., & Pongpaichet, S. (2025). Artificial intelligence in computer programming education: A systematic literature review. *Computers and Education: Artificial Intelligence, 8*, 100403. https://doi.org/10.1016/j.caeai.2025.100403

Nathasya, R. A., Karnalim, O., & Ayub, M. (2019). Integrating program and algorithm visualisation for learning data structure implementation. *Egyptian Informatics Journal, 20*(3), 193–204. https://doi.org/10.1016/j.eij.2019.05.001

Polya, G. (2014). *How to solve it: A new aspect of mathematical method*. Princeton University Press. https://press.princeton.edu/books/ebook/9781400828678/how-to-solve-it-pdf

Rahmani, A. M., Azhir, E., Ali, S., Mohammadi, M., Ahmed, O. H., Ghafour, M. Y., & Ahmed, S. H. (2021). Artificial intelligence approaches and mechanisms for big data analytics: A systematic study. *PeerJ Computer Science*, 1–28. https://doi.org/10.7717/peerj-cs.488

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Pearson.

Scherer, R., & Siddiq, F. (2020). A meta-analysis of teaching and learning computer programming. *Computers in Human Behavior, 109*. https://doi.org/10.1016/j.chb.2020.106349

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley. https://www.pearson.com/en-us/subject-catalog/p/Sedgewick-Algorithms-4th-Edition/P200000000597

Vidyarsih, P., Abdillah, L. A., & Muzakir, A. (2016). Sistem informasi pengarsipan menggunakan algoritma Levenshtein string. *SHaP-SITI 2016*, 7–12.

Wijaya, D., & Abdillah, L. A. (2023). Sentiment analysis of Omicron COVID-19 variant using Naïve Bayes classifier

and RapidMiner. *Journal of Data Sciences, 8*, 1–7. http://eprints.intimal.edu.my/1783/1/jods2023_08.pdf

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Zimmermann, A. E., King, E. E., & Bose, D. D. (2024). Effectiveness and utility of flowcharts on learning in a classroom setting: A mixed-methods study. *American Journal of Pharmaceutical Education, 88*(1). https://www.sciencedirect.com/science/article/abs/pii/S0002945923044911