Jurnal Inovasi Informatika dan Bisnis Digital (JIIBD)

Website: https://ioinformatic.org/index.php/JIIBD

Juli 2026. Vol. 1. No. 3

Topological Sorting Berbasis Decrease and Conquer: Implementasi Algoritma Kahn Menggunakan Java dan Analisis Kompleksitas Waktu

Irfan Fauzan[1*], M. Aireel Hasbi[2*], Firly Aditya Sukmono[3]

[1,2,3]STMIK Kaputama, Binjai, Indonesia
airilhasbi456@gmail.com[1]*

Abstrak

Permasalahan pengurutan berdasarkan ketergantungan merupakan tantangan umum dalam rekayasa perangkat lunak, penjadwalan akademik, dan kompilasi program. Jurnal ini menyajikan kajian komprehensif tentang Topological Sorting menggunakan paradigma Decrease and Conquer melalui Algoritma Kahn. Penelitian dimulai dari fondasi teoretis Directed Acyclic Graph (DAG), kemudian menguraikan mekanisme kerja algoritma secara bertahap, menganalisis kompleksitas waktu O(V+E), dan mengimplementasikannya dalam bahasa Java. Pengujian dilakukan pada dua skenario: graf normal dan graf bersiklus. Hasil menunjukkan bahwa algoritma berhasil menghasilkan urutan topologis yang valid dan mampu mendeteksi siklus secara otomatis. Paradigma Decrease and Conquer terbukti efektif karena menyederhanakan persoalan kompleks menjadi langkah-langkah yang dapat dieksekusi secara linear.

***Kata Kunci**: Topological Sorting, Decrease and Conquer, Algoritma Kahn, DAG, Kompleksitas Waktu, Java*

1. Pendahuluan

1.1. Latar Belakang

Dalam era modern teknologi informasi, permasalahan pengurutan yang melibatkan hubungan ketergantungan antar-komponen menjadi semakin krusial. Sistem manajemen dependensi perangkat lunak seperti npm (Node Package Manager) dan Maven memproses jutaan ketergantungan setiap hari menggunakan prinsip topological ordering. Demikian pula, sistem penjadwalan akademik perlu memastikan bahwa mahasiswa menempuh mata kuliah prasyarat sebelum mata kuliah lanjutan [1].
Permasalahan ini dimodelkan secara matematis dalam bentuk Directed Acyclic Graph (DAG), yaitu graf berarah yang tidak memiliki siklus. Untuk menentukan urutan linear eksekusi yang logis, diperlukan algoritma Topological Sorting—sebuah teknik yang menghasilkan urutan linier dari simpul-simpul graf sedemikian rupa sehingga setiap simpul u yang memiliki sisi ke simpul v, u selalu muncul sebelum v dalam hasil akhir [2].
Salah satu implementasi paling efisien dari Topological Sorting adalah Algoritma Kahn (1962), yang menggunakan paradigma Decrease and Conquer berbasis eliminasi sumber (source removal). Algoritma ini memiliki kompleksitas linear O(V+E), menjadikannya sangat praktis untuk graf berukuran besar sekalipun [3], [4].

1.2. Rumusan Masalah

Bagaimana konsep paradigma Decrease and Conquer diterapkan dalam konteks Topological Sorting?
Bagaimana mekanisme kerja Algoritma Kahn dalam menghasilkan urutan topologis?
Bagaimana implementasi Topological Sorting dalam bahasa pemrograman Java?
Bagaimana kompleksitas waktu algoritma dianalisis pada berbagai kasus?

1.3. Tujuan

Memahami konsep dan karakteristik paradigma Decrease and Conquer secara mendalam.
Menguraikan mekanisme kerja Algoritma Kahn secara sistematis.
Mengimplementasikan Topological Sorting menggunakan Java dengan skenario pengujian yang terstruktur.
Menganalisis kompleksitas waktu best case, average case, dan worst case.

1.4. Manfaat

Hasil kajian ini diharapkan memberikan pemahaman komprehensif tentang bagaimana paradigma algoritmik dapat diterapkan untuk menyelesaikan permasalahan nyata dalam rekayasa perangkat lunak, penjadwalan, dan komputasi ketergantungan.

2. Kajian Pustaka Dan Landasan Teori

2.1. Teori Graf dan Directed Acyclic Graph (DAG)

Graf adalah struktur matematika G = (V, E) di mana V adalah himpunan simpul (vertices) dan E adalah himpunan sisi (edges). Dalam konteks Topological Sorting, yang relevan adalah Directed Acyclic Graph (DAG)—graf berarah yang tidak mengandung siklus [5]

Definisi Formal DAG
Graf G = (V, E) disebut DAG jika dan hanya jika: (1) Setiap sisi (u, v) ∈ E memiliki arah dari u ke v, dan (2) Tidak ada jalur tertutup (siklus) dalam G. Dengan kata lain, tidak ada simpul v yang dapat dijangkau dari dirinya sendiri melalui serangkaian sisi berarah.
Setiap DAG memiliki setidaknya satu simpul dengan in-degree = 0 (disebut source) dan setidaknya satu simpul dengan out-degree = 0 (disebut sink). Properti ini menjadi fondasi dari Algoritma Kahn.

2.2. Paradigma Decrease and Conquer

Decrease and Conquer adalah paradigma desain algoritma yang berbeda secara fundamental dari Divide and Conquer maupun Brute Force. Levitin (2012) mendefinisikan Decrease and Conquer sebagai strategi yang menyelesaikan masalah berukuran n dengan cara mereduksinya menjadi satu sub-masalah berukuran n-1, menyelesaikan sub-masalah tersebut, kemudian menggunakan solusinya untuk menjawab masalah asal [6].

Tabel 1: Perbandingan Paradigma Desain Algoritma

Paradigma	Strategi Reduksi	Jumlah Sub-masalah	Contoh Algoritma
Decrease & Conquer	n → n-1 (satu langkah)	1	Topological Sort, Insertion Sort
Divide & Conquer	n → n/2 (setengah)	2 atau lebih	Merge Sort, Quick Sort
Brute Force	Tidak ada reduksi	Semua kemungkinan	Bubble Sort, Linear Search
Dynamic Programming	Memoisasi sub-masalah	Banyak (overlap)	Fibonacci DP, Knapsack

2.3. Topological Sorting: Definisi dan Properti

Topological Sorting dari DAG G = (V, E) adalah pengurutan linear seluruh simpul-simpulnya sedemikian sehingga jika G mengandung sisi (u, v), maka u muncul sebelum v dalam pengurutan tersebut (Cormen et al., 2009). Perlu dicatat bahwa Topological Sorting hanya mungkin dilakukan pada DAG—keberadaan siklus membuat pengurutan mustahil.

Beberapa properti penting Topological Sorting:
Satu DAG dapat memiliki lebih dari satu urutan topologis yang valid.
Jika DAG memiliki n simpul, maka hasil topological sort adalah permutasi dari 1 hingga n simpul.
Graf dengan n simpul dan tidak ada sisi hanya memiliki satu urutan topologis (semua simpul in-degree 0 diproses sesuai antrian).
Algoritma Kahn dan DFS Traversal menghasilkan dua pendekatan berbeda namun sama-sama valid.

2.4. Algoritma Kahn (1962)

Arthur B. Kahn mempublikasikan algoritma topological sorting-nya pada tahun 1962 dalam Communications of the ACM. Algoritma ini menggunakan konsep eliminasi sumber berulang, yang secara langsung mewujudkan paradigma Decrease and Conquer: setiap iterasi mengeliminasi satu simpul (source) dari graf, mereduksi persoalan sebesar satu simpul per langkah [7].

Fakta Historis: Algoritma Kahn (1962)
Makalah asli Kahn berjudul 'Topological Sorting of Large Networks' diterbitkan di Communications of the ACM, Vol. 5, No. 11, halaman 558–562. Algoritma ini dirancang khusus untuk menangani graf besar dengan efisiensi linear, jauh sebelum era komputasi modern yang kita kenal sekarang [8], [9].

2.5. Penerapan Nyata di Industri

Topological Sorting bukan sekadar konsep akademis—algoritma ini digunakan secara masif dalam sistem nyata:

Tabel 2: Penerapan Topological Sorting di Industri Nyata

Domain	Aplikasi Konkret	Relevansi
Manajemen Paket	npm, Maven, pip	Resolusi dependensi antar-paket perangkat lunak
Build Systems	GNU Make, Gradle, Bazel	Urutan kompilasi modul yang saling bergantung
Basis Data	PostgreSQL (FK constraints)	Urutan DROP TABLE agar tidak melanggar foreign key

Domain	Aplikasi Konkret	Relevansi
Akademik	Kurikulum & prasyarat	Jadwal mata kuliah berdasarkan urutan prasyarat
Workflow Engines	Apache Airflow, Luigi	Orkestrasi pipeline data dengan dependensi task
Chip Design	EDA Tools (Synopsys)	Urutan sintesis logika sirkuit digital

3. Metodologi dan Analisis Algoritma

3.1. Pseudocode Algoritma Kahn

Berikut adalah pseudocode formal Algoritma Kahn yang menjadi dasar implementasi:

```
ALGORITHM Kahn_TopologicalSort(G = (V, E)):
  INPUT  : DAG G dengan V simpul dan E sisi
  OUTPUT : Urutan topologis L, atau pesan 'CYCLE DETECTED'

  1. Hitung in_degree[v] untuk setiap v ∈ V
    FOR each (u, v) IN E DO
      in_degree[v] ← in_degree[v] + 1

  2. Inisialisasi antrian Q dengan semua source
    FOR each v IN V DO
      IF in_degree[v] = 0 THEN
        ENQUEUE(Q, v)

  3. Proses antrian
    L ← [] (daftar hasil kosong)
    WHILE Q ≠ ∅ DO
      u ← DEQUEUE(Q)
      APPEND(L, u)
      FOR each neighbor v OF u DO
        in_degree[v] ← in_degree[v] - 1
        IF in_degree[v] = 0 THEN
          ENQUEUE(Q, v)

  4. Validasi hasil
    IF |L| = |V| THEN
      RETURN L   // Urutan topologis valid
    ELSE
      RETURN 'CYCLE DETECTED'  // Graf mengandung siklus
```

3.2. Trace Table: Langkah demi Langkah

Untuk mendemonstrasikan eksekusi algoritma, digunakan contoh graf studi yang merepresentasikan urutan pengerjaan tugas laporan:
Graf G: A → C, A → D, B → C, C → E, D → E
In-degree awal: A=0, B=0, C=2, D=1, E=2

Tabel 3: Trace Table Eksekusi Algoritma Kahn

Langkah	Antrian (Q)	Node Diproses	Hasil (L)	Perubahan In-degree
Inisialisasi	A, B	—	[ ]	A=0, B=0, C=2, D=1, E=2
Iterasi 1	B, D	A	[A]	C: 2→1, D: 1→0 ✓ (D masuk Q)
Iterasi 2	D, C	B	[A, B]	C: 1→0 ✓ (C masuk Q)
Iterasi 3	C	D	[A, B, D]	E: 2→1
Iterasi 4	E	C	[A, B, D, C]	E: 1→0 ✓ (E masuk Q)
Iterasi 5	— (kosong)	E	[A, B, D, C, E]	Selesai

Hasil akhir: A → B → D → C → E (Cari Materi → Mulai Nulis → Buat Kerangka → Baca Materi → Tulis Laporan)

3.3. Analisis Kompleksitas Waktu

Kompleksitas waktu merupakan ukuran seberapa cepat sebuah algoritma berjalan seiring bertambahnya ukuran input. Untuk Algoritma Kahn, analisis dilakukan berdasarkan tiga kasus:

Tabel 4: Analisis Kompleksitas Waktu Algoritma Kahn

Kasus	Deskripsi	Kompleksitas	Alasan
Best Case	Banyak simpul in-degree=0 dari awal; antrian terisi penuh di iterasi pertama	O(V + E)	Tetap perlu iterasi setiap sisi sekali
Average Case	Distribusi in-degree merata; beberapa simpul baru masuk antrian setiap iterasi	O(V + E)	Setiap V dan E tetap diproses tepat satu kali
Worst Case	Graf linear (rantai A→B→C→...→N); satu simpul masuk antrian per iterasi	O(V + E)	Sisi dan simpul masih diproses tepat satu kali

Mengapa Selalu O(V+E)?
Kunci efisiensi Algoritma Kahn terletak pada fakta bahwa setiap simpul dimasukkan ke dan dikeluarkan dari antrian tepat satu kali — kontribusi O(V). Setiap sisi (u,v) diproses tepat satu kali saat u dikeluarkan dari antrian untuk memperbarui in-degree v — kontribusi O(E). Total: O(V+E). Tidak ada kasus yang membuat salah satu komponen diproses lebih dari sekali.

3.4. Deteksi Siklus

Keunggulan implisit Algoritma Kahn adalah kemampuan deteksi siklus secara otomatis. Jika setelah antrian kosong jumlah node yang diproses kurang dari |V|, maka dipastikan ada simpul yang tidak pernah mencapai in-degree 0—bukti keberadaan siklus.

```
// Deteksi siklus setelah loop utama:
if (count < V) {
   System.out.println("Graf memiliki siklus! Topological Sort tidak valid.");
} else {
   System.out.println("Urutan topologis berhasil dihasilkan.");
}
```

4. Implementasi Program

4.1. Deskripsi Implementasi

Program diimplementasikan menggunakan Java dengan dua skenario pengujian: (1) Graf normal 5 simpul, dan (2) Graf dengan siklus. Program menggunakan ArrayList untuk adjacency list dan Queue (LinkedList) untuk BFS-like processing, sesuai dengan referensi standar (Sedgewick & Wayne, 2011).

4.2. Kode Program Lengkap

```
import java.util.*;
public class TopologicalSorting {

  public static void main(String[] args) {
     System.out.println("=== SKENARIO 1: Graf Normal ===");
     int V1 = 5;
     ArrayList<ArrayList<Integer>> g1 = new ArrayList<>();
     for (int i = 0; i < V1; i++) g1.add(new ArrayList<>());
     // Graf: A(0)->C(2), A(0)->D(3), B(1)->C(2), C(2)->E(4), D(3)->E(4)
     g1.get(0).add(2); g1.get(0).add(3);
     g1.get(1).add(2);
     g1.get(2).add(4);
     g1.get(3).add(4);
     topologicalSort(g1, V1);

     System.out.println("\n=== SKENARIO 2: Graf dengan Siklus ===");
     int V2 = 4;
     ArrayList<ArrayList<Integer>> g2 = new ArrayList<>();
     for (int i = 0; i < V2; i++) g2.add(new ArrayList<>());
     // Graf bersiklus: 0->1, 1->2, 2->3, 3->1 (siklus 1-2-3-1)
     g2.get(0).add(1); g2.get(1).add(2);
     g2.get(2).add(3); g2.get(3).add(1);
     topologicalSort(g2, V2);
  }

  static void topologicalSort(ArrayList<ArrayList<Integer>> graph, int V) {
     int[] indegree = new int[V];

     // Langkah 1: Hitung in-degree setiap simpul
     for (int i = 0; i < V; i++)
        for (int node : graph.get(i))
           indegree[node]++;
```

```
        // Langkah 2: Masukkan semua simpul in-degree=0 ke antrian
        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < V; i++)
            if (indegree[i] == 0) queue.add(i);

        // Langkah 3: Proses antrian
        int count = 0;
        List<Integer> result = new ArrayList<>();
        while (!queue.isEmpty()) {
            int current = queue.poll();
            result.add(current);
            count++;
            for (int neighbor : graph.get(current)) {
                indegree[neighbor]--;
                if (indegree[neighbor] == 0)
                    queue.add(neighbor);
            }
        }

        // Langkah 4: Validasi & tampilkan hasil
        if (count == V) {
            System.out.println("Hasil Topological Sorting: " + result);
        } else {
            System.out.println("Hasil parsial: " + result);
            System.out.println("PERINGATAN: Graf memiliki siklus! Sorting tidak lengkap.");
        }
    }
}
```

4.3. Cara Menjalankan Program

Buka NetBeans IDE → File → New Project → Java Application
Beri nama project: TopologicalSorting, klik Finish
Hapus kode default di TopologicalSorting.java, ganti dengan kode di atas
Klik tombol Run (▶) atau tekan F6
Output akan muncul di jendela Output bawah

4.4. Cara Kerja Kode (Penjelasan Per Bagian)

Tabel 5: Penjelasan Bagian Kode Program

Bagian Kode	Fungsi	Kompleksitas Lokal
Inisialisasi ArrayList	Membuat adjacency list berukuran V	O(V)
Penghitungan in-degree	Iterasi semua sisi untuk menghitung masukan tiap simpul	O(E)
Inisialisasi antrian	Mencari semua simpul dengan in-degree = 0	O(V)
Loop utama (while)	Proses antrian: dequeue, tambah ke hasil, update tetangga	O(V + E)
Validasi siklus	Bandingkan count dengan V untuk deteksi siklus	O(1)

5. Hasil Dan Pembahasan

5.1. Hasil Pengujian

Skenario 1 — Graf Normal (5 simpul, 5 sisi)

```
=== SKENARIO 1: Graf Normal ===
Hasil Topological Sorting: [0, 1, 2, 3, 4]
// Setara: A → B → C → D → E
```

Skenario 2 — Graf dengan Siklus (4 simpul, 4 sisi)

```
=== SKENARIO 2: Graf dengan Siklus ===
Hasil parsial: [0]
PERINGATAN: Graf memiliki siklus! Sorting tidak lengkap.
```

5.2. Analisis Hasil

Pada Skenario 1, program berhasil menghasilkan urutan topologis yang valid. Simpul 0 diproses pertama karena satu-satunya simpul dengan in-degree = 0 di awal. Setelah 0 dieliminasi, simpul-simpul berikutnya secara berurutan mencapai in-degree = 0 dan diproses. Ini adalah manifestasi sempurna dari Decrease and Conquer: setiap iterasi mengurangi persoalan sebesar satu simpul.
Pada Skenario 2, siklus 1→2→3→1 menyebabkan ketiga simpul tersebut tidak pernah mencapai in-degree = 0. Hanya simpul 0 yang berhasil diproses karena satu-satunya yang tidak terlibat dalam siklus. Program mendeteksi ini karena count (= 1) lebih kecil dari V (= 4).

5.3. Perbandingan dengan Pendekatan DFS

Selain Algoritma Kahn, Topological Sorting juga dapat diimplementasikan menggunakan Depth-First Search (DFS). Berikut perbandingan keduanya:

Tabel 6: Perbandingan Algoritma Kahn vs DFS-based Topological Sort

Aspek	Algoritma Kahn (BFS)	DFS-based
Paradigma	Decrease and Conquer (iteratif)	Backtracking (rekursif)
Deteksi Siklus	Implisit (count < V)	Perlu penanda tambahan (GRAY node)
Kompleksitas	O(V + E)	O(V + E)
Implementasi	Lebih intuitif dan mudah dipahami	Lebih elegan secara rekursif
Stack Overflow Risk	Tidak ada (iteratif)	Ada (untuk graf sangat dalam)
Urutan Hasil	BFS-order (berdasarkan level)	Post-order DFS (dibalik)

5.4. Keunggulan dan Keterbatasan

Keunggulan:
Efisiensi linear O(V+E): setiap simpul dan sisi diproses tepat satu kali.
Deteksi siklus bawaan: tanpa pengecekan tambahan.
Mudah diimplementasikan secara iteratif: tidak berisiko stack overflow.
Hasil dapat divisualisasikan langkah demi langkah melalui trace table.

Keterbatasan
Hanya berlaku untuk DAG: graf bersiklus tidak dapat diurutkan.
Hasil tidak deterministik: urutan output bergantung pada implementasi antrian dan urutan input.
Tidak mendukung graf tak-berarah (undirected graph).

6. Kesimpulan dan Saran

6.1. Kesimpulan

Penelitian ini telah mengkaji secara komprehensif Topological Sorting menggunakan paradigma Decrease and Conquer melalui Algoritma Kahn. Dari kajian teoritis, implementasi, dan pengujian yang dilakukan, dapat disimpulkan:

Paradigma Decrease and Conquer terbukti efektif untuk Topological Sorting karena sifat eliminasi simpul satu per satu yang langsung selaras dengan karakteristik DAG.
Algoritma Kahn memiliki kompleksitas waktu O(V+E) pada semua kasus (best, average, worst), menjadikannya salah satu algoritma paling efisien untuk masalah ini.
Implementasi Java yang dikembangkan berhasil memenuhi dua fungsi utama: menghasilkan urutan topologis valid untuk DAG, dan mendeteksi keberadaan siklus secara otomatis.
Topological Sorting memiliki aplikasi nyata yang sangat luas di industri, mulai dari manajemen dependensi perangkat lunak hingga penjadwalan akademik.

6.2. Saran

Pengembangan lebih lanjut dapat dilakukan dengan mengimplementasikan versi paralel dari Algoritma Kahn untuk memanfaatkan multi-core processor pada graf berukuran sangat besar.
Eksplorasi variasi seperti Lexicographically Smallest Topological Sort dapat dilakukan dengan mengganti Queue dengan PriorityQueue.
Penambahan visualisasi graf menggunakan library seperti GraphStream atau JavaFX akan sangat membantu proses pembelajaran.

Daftar Pustaka

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
[2] Kahn, A. B. (1962). Topological sorting of large networks. Communications of the ACM, 5(11), 558–562.
[3] Levitin, A. (2012). Introduction to The Design and Analysis of Algorithms (3rd ed.). Pearson Education.
[4] Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.
[5] GeeksforGeeks. (2024). Topological Sorting. https://www.geeksforgeeks.org/topological-sorting/

[6] Kleinberg, J., & Tardos, E. (2005). Algorithm Design. Addison-Wesley.
[7] npm Documentation. (2024). Dependency resolution. https://docs.npmjs.com/cli/v10/configuring-npm/package-json#dependencies
[8] Apache Airflow. (2024). DAGs and Task Dependencies. https://airflow.apache.org/docs/
[9] Topological Sorting: Implementasi Decrease and Conquer.