UJIAN TENGAH SEMESTER

Mata Kuliah : Analisis Perancangan Algoritma
Dosen Pengampu : Bambang Prasetya Adhi, M.Kom.

Mencerdaskan dan
Memartabatkan Bangsa

Apriansyah Rachman
1512621003
A

PENDIDIKAN TEKNIK INFORMATIKA DAN KOMPUTER
FAKULTAS TEKNIK
UNIVERSITAS NEGERI JAKARTA
SEMESTER 120
2024

Soal

1. Berikan contoh aplikasi yang memerlukan konten algoritmik pada tingkat aplikasi, dan jelaskan fungsi algoritma yang terlibat.
2. Jelaskan mengenai Big-O Notation,Omega Notation,Theta Notation. Lebih detail disertai contoh, grafik akan memberikan nilai lebih.
3. a. { 3T(n-1), if n>0,
   T(n) = { 1, otherwise
   b. { 2T(n-1) – 1, if n>0,
   T(n) = { 1, otherwise
4. lihat code berikut dan temukan kompleksitasnya:
   a.
```
static void function(int n)
{
   int count = 0;
   for (int i = n / 2; i <= n; i++)
      for (int j = 1; j <= n; j = 2 * j)
         for (int k = 1; k <= n; k = k * 2)
            count++;
}
```
   b.
```
public class Main {
   public static void main(String[] args) {
      int n = 5; // You can change the value of n as needed
      printPattern(n);
   }

   static void printPattern(int n) {
      for (int i = 1; i <= n; i++) {
         // Print a single '*' on each line
         System.out.println("*");
      }
   }
}
```

5. Bagaimana langkah-langkah menentukan Big-Omega Ω Notation, jelaskan beserta contohnya.
6. Analisa kompleksitas perbandingan algoritma Merge sort, Quicksort, dan Heap sort dan jelaskan.

Dikerjakan secara mandiri. Boleh mencari referensi dari Internet dengan menyertakan sumber.
Dilarang copy paste langsung, gunakan bahasa sendiri/analisa dengan bahasa sendiri.
Dikumpulkan melalui ketua kelas masing-masing dengan format Noreg_Nama_UTS.pdf

Jawaban

1. Aplikasi yang memerlukan konten algoritmik pada tingkat aplikasi beserta penjelasan tentang fungsi algoritma yang terlibat :
   a. Platform Media Sosial (misalnya Facebook, Instagram, atau TikTok) :
      - Algoritma: Algoritma Pengumpanan Berita (News Feed Algorithm)
      - Fungsi: Menentukan konten yang ditampilkan kepada pengguna di umpan berita mereka. Algoritma ini mempertimbangkan berbagai faktor, seperti:
        a) Interaksi pengguna dengan konten (suka, komentar, bagikan)
        b) Konten yang dibagikan oleh teman dan koneksi pengguna
        c) Informasi tentang konten (seperti topik, relevansi, waktu publikasi)
        d) Sinyal personalisasi lainnya

      Algoritma di sini bertanggung jawab untuk menyesuaikan konten yang ditampilkan di feed pengguna. Algoritma ini mempertimbangkan faktor-faktor seperti preferensi pengguna, interaksi sebelumnya, tren yang sedang berlangsung, dan relevansi konten. Algoritma ini memanfaatkan machine learning untuk mengoptimalkan pengalaman pengguna dengan menampilkan konten yang paling mungkin diminati oleh pengguna.

   b. Mesin Pencarian (misalnya Google Search, , Bing, Yahoo) :
      - Algoritma: Algoritma Peringkat Relevansi (Relevance Ranking Algorithm)
      - Fungsi: Menentukan urutan hasil pencarian berdasarkan relevansi dengan kueri pengguna. Algoritma ini mempertimbangkan berbagai faktor, seperti :
        a) Kata kunci dalam kueri
        b) Konten halaman web
        c) Otoritas dan popularitas situs web
        d) Perilaku pengguna
        e) Sinyal kualitas lainnya

        Algoritma pencarian ini bertanggung jawab untuk memperoleh dan memprioritaskan hasil pencarian yang relevan dan berkualitas. Algoritma ini menggunakan berbagai faktor seperti relevansi kata kunci, otoritas halaman web, popularitas, dan faktor-faktor lainnya untuk menentukan peringkat setiap hasil pencarian. Algoritma pencarian ini terus diperbarui untuk meningkatkan akurasi dan kualitas hasil.

   c. Sistem Rekomendasi (misalnya Netflix, Spotify, atau Amazon) :
      - Algoritma: Sistem Rekomendasi Kolaboratif (Collaborative Filtering System)
      - Fungsi: Merekomendasikan konten kepada pengguna berdasarkan preferensi mereka dan preferensi pengguna lain yang serupa. Algoritma ini mempertimbangkan berbagai faktor, seperti :
        a) Riwayat penayangan atau pendengaran pengguna
        b) Peringkat dan ulasan pengguna
        c) Konten yang direkomendasikan kepada pengguna lain yang serupa
        d) Metadata konten (seperti genre, aktor, sutradara)

   d. Aplikasi Terjemahan Otomatis (Seperti Google Translate, Microsoft Translator)
      - Algoritma: Model Bahasa Neural (Neural Language Model)

- Fungsi: Menerjemahkan teks dari satu bahasa ke bahasa lain. Algoritma ini mempelajari pola dalam data teks terjemahan yang besar untuk menghasilkan terjemahan yang akurat dan alami.

e. Aplikasi Pengenalan Suara (Seperti Siri, Alexa, Google Assistant)
   - Algoritma: Pemrosesan Bahasa Alami (Natural Language Processing)
   - Fungsi: Mengubah ucapan manusia menjadi teks dan memahami maksud pengguna. Algoritma ini mempertimbangkan berbagai faktor, seperti :
     a) Akustik ucapan
     b) Tata bahasa dan sintaksis bahasa
     c) Konteks percakapan
     d) Pengetahuan tentang dunia

2. Tiga notasi kompleksitas yang umum digunakan dalam menganalisis kinerja algoritma adalah Big-O Notation, Omega Notation, dan Theta Notation. Berikut adalah penjelasan masing-masing :
   a. Big-O Notation (O) :

      Penjelasan :

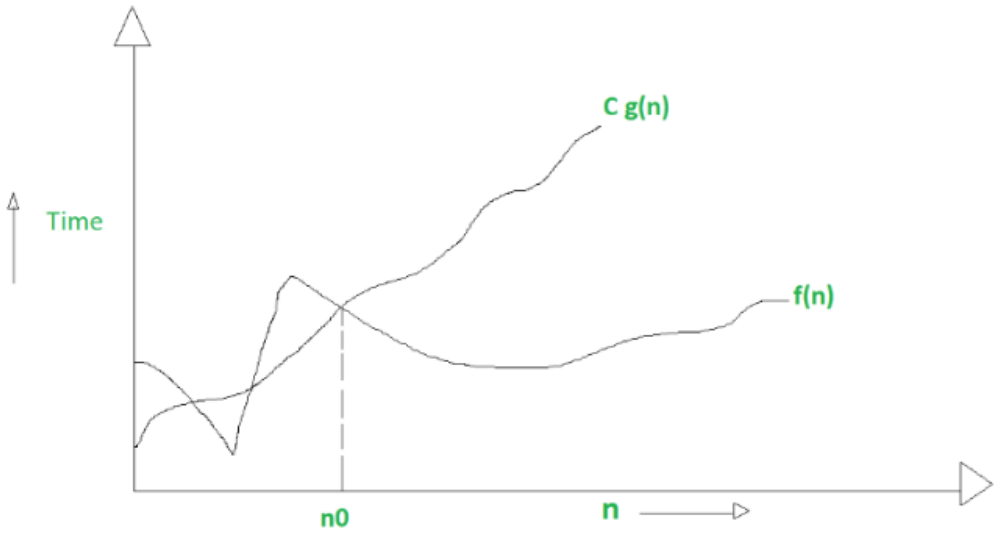
      Big-O Notation memberikan batas atas dari waktu atau ruang yang diperlukan oleh sebuah algoritma dalam menyelesaikan suatu masalah. Ini menggambarkan seberapa cepat waktu eksekusi atau berapa banyak ruang yang diperlukan oleh algoritma ketika ukuran masalah mendekati tak hingga.

      Didefinisikan sebagai batas atas dan batas atas pada suatu algoritma adalah jumlah waktu paling banyak yang dibutuhkan (kinerja kasus terburuk). Big-O Notation (O) digunakan untuk menggambarkan batas atas asimtotik .

      Contoh :

      Jika sebuah algoritma memiliki kompleksitas waktu O(n^2), itu berarti waktu eksekusi algoritma tersebut tumbuh secara kuadratik seiring dengan peningkatan ukuran input (n). Contoh lain, algoritma dengan kompleksitas O(n) memiliki pertumbuhan waktu linear terhadap ukuran input.

      Grafik :



Secara matematis, jika f(n) menggambarkan waktu berjalan suatu algoritma; f(n) adalah O(g(n)) jika terdapat konstanta positif C dan n0 sehingga, 0 <= f(n) <= Cg(n) untuk semua n >= n0.
n = digunakan untuk memberikan fungsi batas atas.

Jika suatu fungsi adalah O(n) , maka secara otomatis juga merupakan O(n-square).

Referensi : https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/

b. Omega Notation (Ω) :

Penjelasan :

Omega Notation memberikan batas bawah dari waktu atau ruang yang diperlukan oleh sebuah algoritma dalam menyelesaikan suatu masalah. Ini menunjukkan kecepatan terbaik yang dapat dicapai oleh algoritma untuk ukuran masalah tertentu.

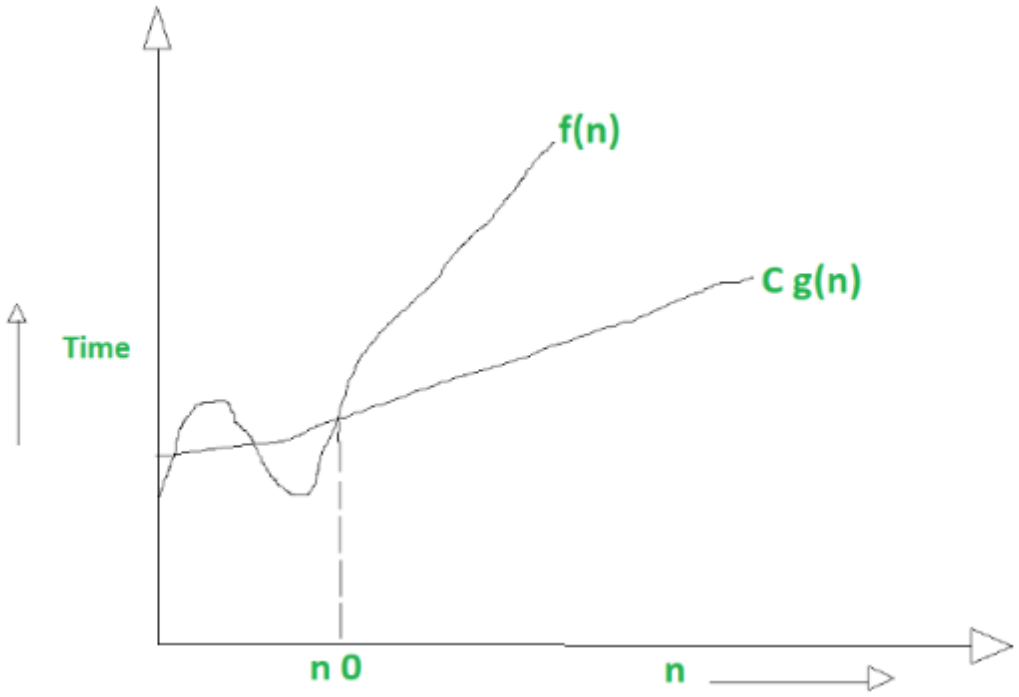
Didefinisikan sebagai batas bawah dan batas bawah pada suatu algoritma adalah jumlah waktu paling sedikit yang dibutuhkan (cara seefisien mungkin, dengan kata lain kasus terbaik).

Sama seperti notasi O yang memberikan batas atas asimtotik , notasi Ω memberikan batas bawah asimtotik .

Contoh :

Jika sebuah algoritma memiliki kompleksitas waktu Ω(n^2), itu berarti waktu eksekusi algoritma tersebut paling lambat akan tumbuh secara kuadratik seiring dengan peningkatan ukuran input (n). Contoh lain, algoritma dengan kompleksitas Ω(n) memiliki pertumbuhan waktu setidaknya linear terhadap ukuran input.

Grafik :



Misalkan f(n) menentukan waktu berjalan suatu algoritma;

f(n) dikatakan Ω(g (n)) jika terdapat konstanta positif C dan (n0) sedemikian sehingga :

0 <= Cg(n) <= f(n) untuk semua n >= n0

n = digunakan untuk memberikan batas bawah pada suatu fungsi
Jika suatu fungsi adalah Ω(n-kuadrat) maka secara otomatis juga Ω(n) .

Referensi : https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/

c. Theta Notation (θ) :

Penjelasan :

Theta Notation memberikan batas tepat dari waktu atau ruang yang diperlukan oleh sebuah algoritma dalam menyelesaikan suatu masalah. Ini memberikan perkiraan yang lebih presisi daripada Big-O dan Omega Notation karena memberikan batasan atas dan bawah yang sama.

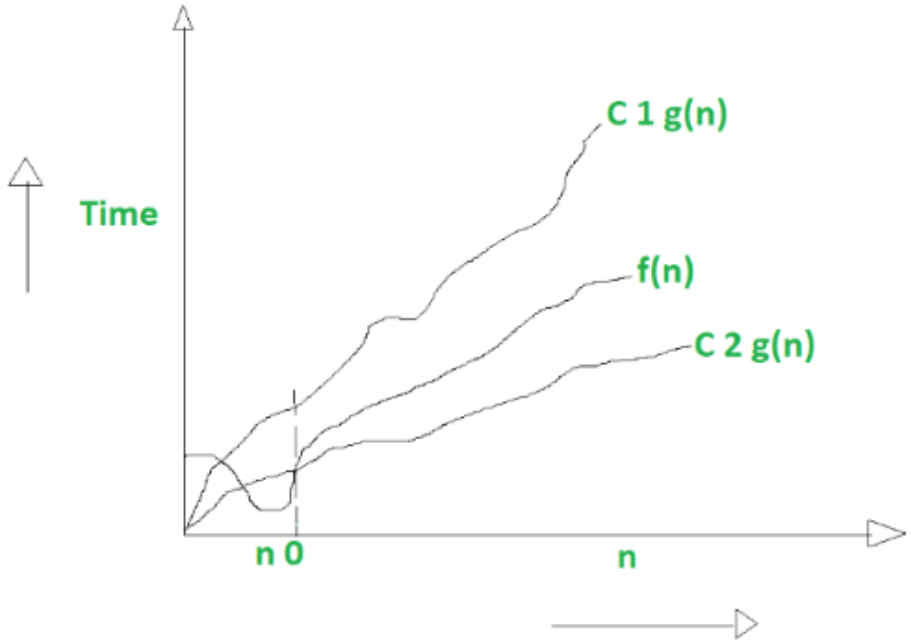
Contoh :

Jika sebuah algoritma memiliki kompleksitas waktu θ(n^2), itu berarti waktu eksekusi algoritma tersebut tumbuh secara kuadratik seiring dengan peningkatan ukuran input (n), dan juga tidak lebih buruk atau lebih baik dari itu. Didefinisikan sebagai batas paling ketat dan batas paling ketat adalah waktu terbaik dari semua waktu terburuk yang dapat diambil oleh algoritme.

Grafik :

Biarkan f(n) menentukan waktu berjalan suatu algoritma.
f(n) dikatakan Θ(g(n)) jika f(n) adalah O(g(n)) dan f(n) adalah Ω(g(n)).
Secara matematis,
0 <= f(n) <= C1g(n) untuk n >= n0
0 <= C2g(n) <= f(n) untuk n >= n0
Menggabungkan kedua persamaan tersebut, kita mendapatkan :
0 <= C2g(n) <= f(n) <= C1g(n) untuk n >= n0

Persamaan tersebut berarti terdapat konstanta positif C1 dan C2 sehingga f(n) terjepit di antara C2 g(n) dan C1g(n). Contoh grafis Big Theta (Θ) :



Referensi : https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/

3. a. { 3T(n-1), if n>0,
   T(n) = { 1, otherwise

   Untuk menyelesaikan fungsi ini, dapat menggunakan induksi matematis. Induksi matematis adalah suatu metode untuk menunjukkan bahwa suatu fungsi atau teorema benar untuk semua nilai input.
   Indeks dapat dimulai dari n = 1, karena kondisi inisial yang diberikan adalah T(n) = 1, jika n <= 0.

   Untuk n = 1, kondisi inisial benar, karena T(1) = 1, sehingga kondisi inisial tepat.

   Untuk n > 1, dapat menggunakan induksi matematis. Dengan induksi matematis, menunjukkan bahwa fungsi T(n) benar untuk semua n > 0.

   Untuk n > 1, kondisi induksi adalah:
   T(n) = 3T(n-1)
   Dengan induksi matematis, menunjukkan bahwa kondisi induksi benar untuk semua n > 0.
   Untuk n > 1, kondisi induksi benar, karena T(n) = 3T(n-1) = 3(3T(n-2)) = 9T(n-2) = 9(3T(n-3)) = 27T(n-3) = ... = 3^(n-1)T(1) = 3^(n-1) * 1 = 3^(n-1).

   Sehingga, dapat menyelesaikan fungsi T(n) = 3^(n-1) untuk semua n > 0.

   b. { 2T(n-1) – 1, if n>0,
   T(n) = { 1, otherwise
   Penyelesaian dari fungsi yang diberikan adalah:
   T(n) = 1, jika n <= 0

T(n) = 2T(n-1) – 1, jika n > 0

Untuk menyelesaikan fungsi ini, akan menggunakan induksi matematis. Induksi matematis adalah suatu metode untuk menunjukkan bahwa suatu fungsi atau teorema benar untuk semua nilai input.

Indeks bisa dimulai dari n = 1, karena kondisi inisial yang diberikan adalah T(n) = 1, jika n <= 0.

Untuk n = 1, kondisi inisial benar, karena T(1) = 1, sehingga kondisi inisial tepat.

Untuk n > 1, dapat menggunakan induksi matematis. Dengan induksi matematis, menunjukkan bahwa fungsi T(n) benar untuk semua n > 0.

Untuk n > 1, kondisi induksi adalah:
T(n) = 2T(n-1) – 1
Dengan induksi matematis, akan menunjukkan bahwa kondisi induksi benar untuk semua n > 0.

Untuk n > 1, kondisi induksi benar, karena T(n) = 2T(n-1) – 1 = 2(2T(n-2) – 1) – 1 = 4T(n-2) – 2 = 4(2T(n-3) – 1) – 2 = 8T(n-3) – 4 = ... = 2^(n-1)T(1) – (2^(n-1) – 1) = 2^(n-1) * 1 – (2^(n-1) – 1) = 2^(n-1) – 2^(n-1) + 1 = 2^(n-1) + 1.

Sehingga, dapat menyelesaikan fungsi T(n) = 2^(n-1) + 1 untuk semua n > 0.

4. a. Dalam analisis tersebut:
    - Loop pertama memiliki kompleksitas O(n) karena ia bergantung pada ukuran input (n).
    - Loop kedua memiliki kompleksitas O(log n) karena variabel j meningkat dengan kelipatan 2 setiap iterasi hingga mencapai n.
    - Loop ketiga juga memiliki kompleksitas O(log n) karena variabel k juga meningkat dengan kelipatan 2 setiap iterasi hingga mencapai n.
    - Operasi di dalam loop (penambahan count) merupakan operasi konstan yang memiliki kompleksitas O(1).
    - Melakukan iterasi dari n / 2 sampai n (inklusif), menghasilkan maksimum n/2 + 1 iterasi. Dalam notasi Big O, ini menjadi O(n).
    - Menggandakan nilainya (j = 2 * j) di setiap iterasi. Ini menciptakan progresi geometri dengan rasio umum 2. Jumlah iterasi (t) untuk j mencapai n dapat dihitung menggunakan rumus:
      j^(t-1) = n
      2^(t-1) = n
      t = log2(n) + 1 (karena looping dimulai dari 1).

    Karena loop-loop tersebut bersarang satu sama lain, kompleksitas waktu total dari algoritma adalah hasil perkalian kompleksitas dari setiap loop. Oleh karena itu, kompleksitas totalnya adalah O(n * log n * log n).

Jadi, kompleksitas waktu dari fungsi tersebut adalah O(n * log^2 n).

b. Dalam analisis tersebut:

Faktor utama yang perlu dipertimbangkan dalam kompleksitas kode ini adalah loop luar (for (int i = 1; i <= n; i++)). Loop ini beriterasi sebanyak n kali, yang menentukan berapa kali kode di dalam loop dieksekusi.

- Dalam notasi Big O, yang merepresentasikan batas atas waktu eksekusi algoritma saat ukuran input tumbuh, kompleksitas loop ini adalah O(n).
- Loop hanya melakukan iterasi sebanyak n kali, di mana n adalah ukuran input.
- Setiap iterasi loop hanya melakukan satu operasi, yaitu mencetak satu karakter di baris baru.

Karena tidak ada peningkatan eksponensial dalam ukuran input yang mempengaruhi jumlah operasi, kompleksitas waktu algoritma ini adalah linier terhadap ukuran input n. Jadi, kompleksitas waktu dari fungsi printPattern adalah O(n).

5. Bagaimana langkah-langkah menentukan Big-Omega Ω Notation, jelaskan beserta contohnya.

Big-Omega Ω Notation digunakan untuk menentukan batas bawah dari laju pertumbuhan kompleksitas waktu suatu fungsi. Ini menunjukkan jumlah waktu minimum yang dibutuhkan fungsi untuk dieksekusi, secara asimtotik.

Langkah-langkah Menentukan Big-Omega Ω Notation :

1. Mengidentifikasi Suku Dominan dengan meriksa kode fungsi dan identifikasi suku yang tumbuh paling signifikan saat ukuran input meningkat. Suku ini biasanya melibatkan operasi seperti perpangkatan atau perkalian dengan konstanta besar.
2. Abaikan Suku dengan Orde Lebih Rendah dengan mengabaikan suku dengan orde lebih rendah (suku yang tumbuh lebih lambat atau tetap konstan saat ukuran input meningkat). Suku-suku ini memiliki dampak yang dapat diabaikan pada keseluruhan perilaku asimtotik.
3. Hilangkan Faktor Konstan dari suku dominan. Faktor-faktor ini tidak memengaruhi laju pertumbuhan fungsi secara asimtotik.
4. Ekspresikan dalam Big-Omega Ω Notation untuk mewakili suku dominan yang diidentifikasi. Bentuk umumnya adalah: Ω(f(n)) = { g(n) | f(n) ≥ c * g(n) untuk beberapa konstanta positif c dan n yang cukup besar }

Contoh :
Perhatikan fungsi berikut :
T(n) = 2n^3 + 5n + 100

Langkah 1: Identifikasi Suku Dominan:

Suku dominan adalah 2n^3, karena ia tumbuh jauh lebih cepat daripada suku lainnya (5n dan 100) saat n meningkat.

Langkah 2: Abaikan Suku dengan Orde Lebih Rendah:

Abaikan suku dengan orde lebih rendah 5n dan 100.

Langkah 3: Hilangkan Faktor Konstan:
Hilangkan faktor konstan 2 dari suku dominan 2n^3.
Langkah 4: Ekspresikan dalam Notasi Big-Omega:
Menggunakan notasi Big-Omega, kita dapatkan: Ω(T(n)) = Ω(n^3)

Penjelasan:
Notasi Ω(n^3) menyiratkan bahwa terdapat konstanta c dan ukuran input n yang cukup besar sehingga T(n) selalu lebih besar atau sama dengan c * n^3 untuk semua n yang lebih besar dari ambang batas tersebut. Ini berarti laju pertumbuhan fungsi minimal secepat n^3, secara asimtotik.

Referensi : https://rizafahmi.com/2020/03/21/notasi-o-besar-big-o-notation/

6. Analisa kompleksitas perbandingan algoritma Merge sort, Quicksort, dan Heap sort dan jelaskan.
   a. Merge Sort dikenal karena kompleksitas waktu yang konsisten, yaitu O (n log n) dalam semua kasus, baik terburuk, rata-rata, maupun terbaik. Algoritma ini stabil, mudah dimengerti, dan cocok untuk data yang besar. Namun, kekurangannya adalah membutuhkan ruang tambahan untuk menyimpan array sementara selama proses penggabungan.
   b. Quick Sort, di sisi lain, memiliki kompleksitas waktu O (n log n) dalam kasus rata-rata dan terbaik, namun dapat menjadi O(n^2) dalam kasus terburuk jika pivot dipilih secara buruk. Keuntungannya adalah efisiensi untuk data yang besar dan tidak memerlukan ruang tambahan karena menggunakan pendekatan in-place sorting. Namun, Quick Sort tidak stabil dan rentan terhadap kasus terburuk jika implementasinya tidak optimal.
   c. Heap Sort, seperti Merge Sort, memiliki kompleksitas waktu waktu O (n log n) dalam semua kasus. Algoritma ini stabil dan dapat diimplementasikan dalam pendekatan in-place sorting. Namun, Heap Sort kurang populer dibandingkan dengan Merge Sort dan Quick Sort karena implementasinya yang lebih rumit.

   Berikut adalah tabel perbandingan komprehensif antara ketiga algoritma:

Algoritma	Merge Sort	Quick Sort	Heap Sort
Kasus Terburuk	O (n log n)	O(n^2)	O (n log n)
Kasus Rata-Rata	O (n log n)	O (n log n)	O (n log n)
Kasus Terbaik	O (n log n)	O (n log n)	O (n log n)
Stabilitas	Stabil	Tidak Stabil	Stabil
Ruang Tambahan	Ya	Tidak	Tidak
Penjelasan	Cocok untuk data besar, membutuhkan ruang tambahan	Cepat, tetapi rentan terhadap kasus terburuk	Kurang populer, kompleksitas seragam dalam semua kasus

Referensi : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2009-2010/Makalah0910/MakalahStrukdis0910-023.pdf