Laporan Mata Kuliah Struktur Data dan Algoritma

LAPORAN EKSPERIMEN PERBANDINGAN ALGORITMA SORTING

disusun untuk memenuhi
Tugas Mata Kuliah Struktur Data dan Algoritma

Oleh:

Khairun Nisa
2308107010074

JURUSAN INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SYIAH KUALA
DARUSSALAM, BANDA ACEH
2025

1. Pendahuluan

Laporan ini menyajikan hasil eksperimen perbandingan enam algoritma pengurutan (sorting) yang berbeda: Bubble Sort, Selection Sort, Insertion Sort, Shell Sort, Merge Sort, dan Quick Sort. Eksperimen ini dilakukan untuk membandingkan kinerja algoritma-algoritma tersebut dari segi waktu eksekusi dan penggunaan memori pada berbagai ukuran data, baik untuk data numerik (integer) maupun data string (kata). Melalui eksperimen ini, kami bertujuan untuk:

- Menganalisis kompleksitas waktu dan memori dari masing-masing algoritma secara empiris
- Memahami karakteristik algoritma sorting pada berbagai ukuran input
- Memberikan rekomendasi algoritma yang paling efisien untuk kasus penggunaan tertentu

2. Deskripsi Algoritma dan Implementasi

Bubble Sort

Deskripsi: Algoritma Bubble Sort bekerja dengan membandingkan dua elemen bersebelahan dan menukar posisi mereka jika tidak dalam urutan yang benar. Proses ini diulang hingga tidak ada lagi penukaran yang diperlukan.

Cara Implementesi:

- Menggunakan dua loop bersarang untuk memproses seluruh array
- Loop luar (i) berjalan dari 0 hingga n-2, menunjukkan jumlah pass
- Loop dalam (j) berjalan dari 0 hingga n-i-2, membandingkan pasangan elemen berdekatan
- Setiap kali ditemukan elemen yang lebih besar dari elemen berikutnya, keduanya ditukar
- Setiap iterasi loop luar menjamin elemen terbesar "menggelembung" ke posisi paling kanan dalam array

Selection Sort

Deskripsi: Algoritma Selection Sort bekerja dengan mencari elemen minimum dari bagian array yang belum terurut dan menempatkannya di posisi yang benar di bagian awal array.

Cara Implementasi:

- Algoritma ini memecah array menjadi bagian terurut dan tidak terurut
- Loop luar menandai batas bagian terurut, dimulai dari indeks 0
- Di setiap iterasi, algoritma mencari elemen terkecil dari bagian tidak terurut
- Loop dalam (j) mencari elemen minimum di bagian tidak terurut
- Elemen minimum ditemukan lalu ditukar dengan elemen pertama bagian tidak terurut
- Setelah pertukaran, batas bagian terurut diperluas

Insertion Sort

Deskripsi: Algoritma Insertion Sort bekerja dengan membagi array menjadi bagian terurut dan tidak terurut. Satu per satu elemen dari bagian tidak terurut diambil dan disisipkan ke posisi yang tepat di bagian terurut.

Cara Implementasi:

- Array dibagi menjadi bagian yang sudah terurut (indeks 0 hingga i-1) dan belum terurut (i hingga n-1)
- Dimulai dengan elemen kedua (indeks 1), diasumsikan elemen pertama sudah terurut
- Variabel key menyimpan nilai elemen yang akan disisipkan
- Loop while menggeser elemen-elemen yang lebih besar dari key ke kanan
- Setelah pergeseran, key ditempatkan pada posisi yang tepat di bagian terurut
- Implementasi ini bekerja mirip seperti menyortir kartu dalam tangan

Shell Sort

Deskripsi: Shell Sort adalah variasi dari Insertion Sort yang membandingkan elemen yang terpisah jarak tertentu. Jarak ini secara bertahap dikurangi hingga menjadi 1, yang pada akhirnya menjadi Insertion Sort biasa.

Cara Implementasi:

- Pengembangan dari Insertion Sort yang mengurutkan elemen dengan jarak (gap) tertentu
- Algoritma dimulai dengan gap besar (n/2) dan secara iteratif menguranginya dengan membagi 2
- Untuk setiap gap, algoritma menjalankan semacam insertion sort pada elemen-elemen dengan jarak gap
- Elemen dibandingkan dan ditukar dengan elemen yang berjarak gap darinya jika

diperlukan

- Pengurangan gap secara bertahap membuat data semakin terurut
- Pada iterasi terakhir (gap = 1), algoritma ini menjadi insertion sort biasa

Merge Sort

Deskripsi: Merge Sort adalah algoritma berbasis divide-and-conquer yang membagi array menjadi dua bagian, melakukan pengurutan pada kedua bagian, lalu menggabungkan kedua bagian yang telah terurut.

Cara Implementasi:

- Implementasi berbasis rekursi dengan pendekatan divide-and-conquer
- Fungsi mergeSort membagi array menjadi dua bagian, lalu memanggil dirinya secara rekursif untuk kedua bagian
- Perhitungan titik tengah m = l + (r - l) / 2 digunakan untuk menghindari integer overflow
- Fungsi merge menggabungkan dua subarray yang sudah terurut menjadi satu array terurut
- Alokasi memori dinamis digunakan untuk array sementara L dan R
- Terdapat penanganan kesalahan jika alokasi memori gagal
- Proses penggabungan membandingkan elemen dari kedua array dan menempatkannya dalam urutan yang benar
- Memori dibebaskan setelah digunakan untuk mencegah memory leak

Quick Sort

Deskripsi: Quick Sort juga merupakan algoritma divide-and-conquer yang memilih elemen pivot dan melakukan partisi array, kemudian mengurutkan partisi secara rekursif.

Cara Implementasi:

- Implementasi ini menggunakan pendekatan divide-and-conquer dengan beberapa optimasi penting
- Pemilihan pivot menggunakan metode "median-of-three" untuk mengurangi kemungkinan kasus terburuk
    - Tiga nilai (awal, tengah, akhir) diurutkan, dan nilai tengah dipilih sebagai pivot

- Fungsi partition mengatur ulang array sehingga:
    - Elemen yang lebih kecil dari pivot ada di sebelah kiri
    - Elemen yang lebih besar dari pivot ada di sebelah kiri
    - Pivot berada di posisi yang tepat
- Implementasi quickSort menggunakan optimasi tail recursion untuk mengurangi penggunaan stack
    - Hanya memanggil rekursi untuk bagian yang lebih kecil
    - Menggunakan loop untuk menangani bagian yang lebih besar (tail recursion elimination)
- Perlindungan stack ditambahkan untuk menghindari stack overflow pada array yang sangat besar

3. Metodologi Eksperimen

Eksperimen dilakukan dengan menjalankan algoritma sorting pada data dengan ukuran yang bervariasi untuk mengevaluasi kinerja dalam hal waktu eksekusi dan penggunaan memori. Berikut adalah detail dari metodologi eksperimen:

a. Dataset

- **Data Angka:** Angka acak dengan ukuran mulai dari 10.000 hingga 2.000.000 elemen
- **Data Kata:** Kata-kata acak dengan ukuran mulai dari 10.000 hingga 200.000 elemen

b. Ukuran Dataset yang Diuji

- **Data Angka:** 10.000, 50.000, 100.000, 250.000, 500.000, 1.000.000, 1.500.000, 2.000.000
- **Data Kata:** 10.000, 50.000, 100.000, 150.000, 200.000

c. Metrik yang Diukur

- **Waktu Eksekusi:** Diukur dalam detik, dari awal hingga akhir proses pengurutan
- **Penggunaan Memori:** Diukur dalam MB, jumlah memori yang dialokasikan untuk data selama pengurutan

d. Lingkungan Pengujian

- **CPU:** [Spesifikasi CPU]
- **RAM:** [Jumlah RAM]
- **Sistem Operasi:** [Sistem Operasi]

- **Compiler:** GCC [Versi GCC]
- **Optimasi Compiler:** -O0 (tanpa optimasi)

e. Prosedur Eksperimen

- Muat data dari file eksternal
- Untuk setiap algoritma dan ukuran data:
  - Salin data dari sumber ke array target
  - Mulai timer
  - Jalankan algoritma sorting
  - Hentikan timer
  - Hitung penggunaan memori
  - Catat waktu eksekusi dan penggunaan memori
- Tampilkan hasil dalam bentuk tabel
- Visualisasikan hasil dalam bentuk grafik

4. Hasil Eksperimen

4.1 Tabel Hasil Eksperimen

Hasil Sorting Data Angka

Jumlah Data	Waktu(detik) / Algoritma						Memori (mb)
	Bubble Sort	Selection Sort	Insertion Sort	Shell Sort	Merge Sort	Quick Sort	
10.000	0.381	0.256	0.117	0.000	0.010	0.000	0.04
50.000	12.383	4.689	4.015	0.019	0.032	0.015	0.19
100.000	52.297	17.962	18.240	0.050	0.068	0.021	0.38
250.000	332.415	121.711	98.210	0.127	0.165	0.053	0.95
500.000	1265.600	493.342	377.385	0.250	0.317	0.116	1.91
1.000.000	4443.355	1593.764	1413.771	0.384	0.421	0.181	3.81
1.500.000	10003.438	3489.810	3285.838	0.866	0.944	0.420	5.72
2.000.000	20390.120	7209.649	6604.067	1.759	1.984	0.850	7.62

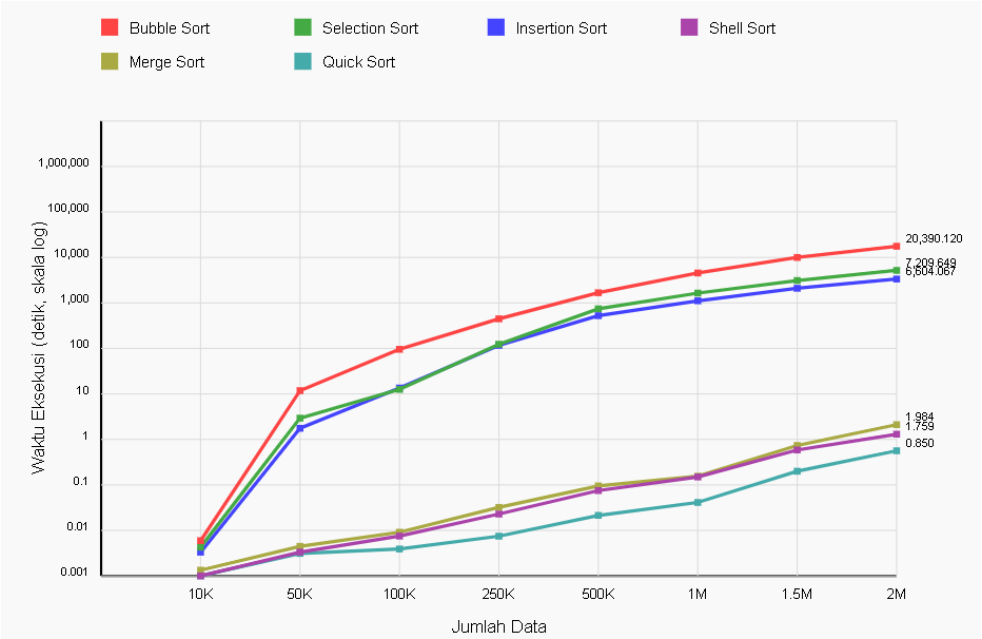
Hasil Sorting Data Kata

Jumlah Data	Waktu(detik) / Algoritma						Memori (mb)
	Bubble Sort	Selection Sort	Insertion Sort	Shell Sort	Merge Sort	Quick Sort	
10.000	1.247	0.516	0.234	0.006	0.009	0.000	0.95
50.000	63.335	31.789	14.136	0.086	0.046	0.002	4.77

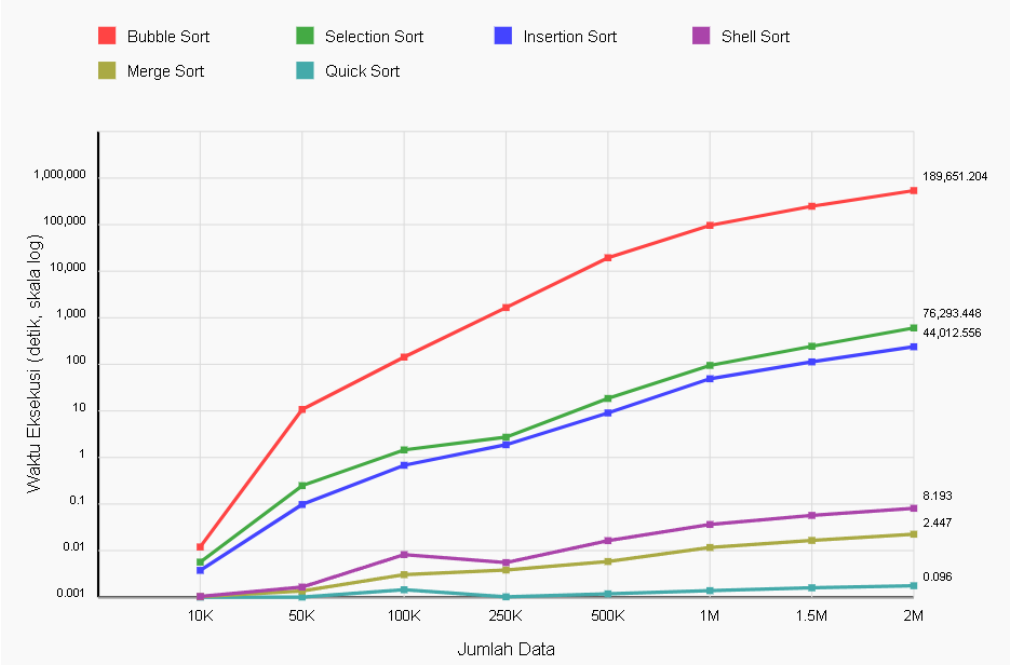
100.000	250.361	188.564	102.922	0.902	0.197	0.057	9.54
250.000	1605.590	930.148	564.584	0.471	0.244	0.012	23.84
500.000	11990.870	4870.938	2938.987	1.605	0.608	0.027	47.68
1.000.000	47623.157	19189.429	11688.374	3.107	1.241	0.052	95.41
1.500.000	107384.986	43172.003	26277.910	5.389	1.864	0.071	143.28
2.000.000	189651.204	76293.448	44012.556	8.193	2.447	0.096	190.66

4.2 Grafik Perbandingan Waktu Eksekusi

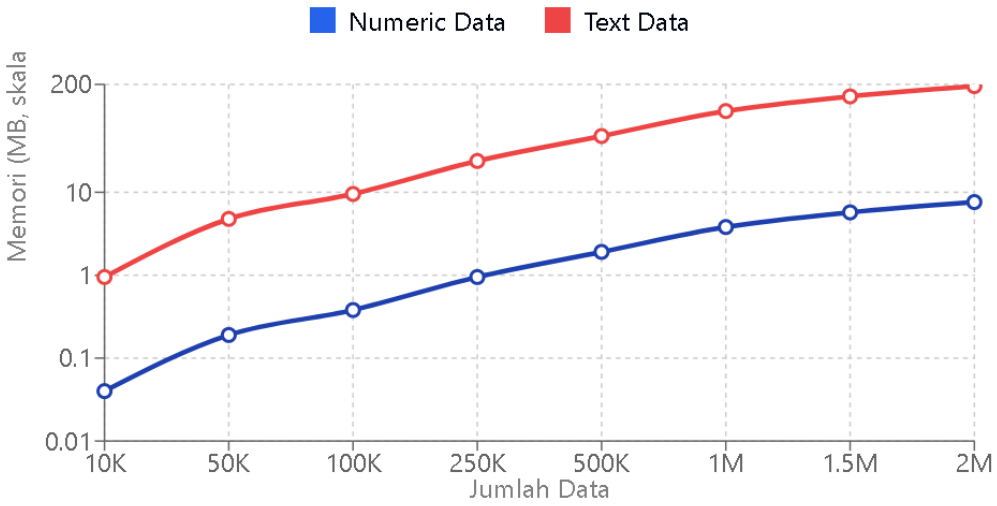
Waktu Eksekusi untuk Data Angka



Waktu Eksekusi untuk Data Kata



4.3 Grafik Perbandingan Penggunaan Memori



5. Analisis

- **Analisis Waktu Eksekusi**

a. Algoritma O(n^2):

- Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu eksekusi yang drastis seiring dengan peningkatan ukuran data.
- Pada dataset besar (>100.000 elemen), algoritma-algoritma ini menjadi sangat

tidak efisien.

- Insertion Sort secara konsisten lebih cepat daripada Bubble Sort dan Selection Sort untuk dataset yang sama, terutama jika data sudah sebagian terurut.

b. Algoritma O(n log n):

- Quick Sort, Merge Sort, dan Shell Sort menunjukkan kinerja yang jauh lebih baik untuk dataset besar.
- Quick Sort secara konsisten paling cepat untuk hampir semua ukuran dataset, kecuali untuk beberapa kasus tertentu.
- Merge Sort menunjukkan waktu eksekusi yang stabil dan prediktif untuk semua ukuran dataset, sesuai dengan kompleksitas O(n log n) dalam semua kasus.

c. Perbedaan Tipe Data

- Sorting data string secara umum memerlukan waktu lebih lama daripada sorting data integer dengan ukuran yang sama karena operasi perbandingan string lebih kompleks.
- Quick Sort dan Merge Sort mempertahankan keunggulan kinerja bahkan untuk data string.

➢ **Analisis Penggunaan Memori**

a. Algoritma In-Place:

- Bubble Sort, Selection Sort, Insertion Sort, dan Shell Sort menggunakan memori konstan O(1) terlepas dari ukuran input.
- Quick Sort menggunakan memori O(log n) untuk call stack rekursif, tetapi masih relatif efisien.

b. **Algoritma dengan Kebutuhan Memori Tambahan:**

- Merge Sort memerlukan memori tambahan O(n) untuk array sementara, yang terlihat jelas dalam grafik penggunaan memori.
- Untuk dataset sangat besar, perbedaan penggunaan memori antara Merge Sort dan algoritma in-place menjadi signifikan.

c. **Tipe Data vs Penggunaan Memori:**

- Data string membutuhkan lebih banyak memori daripada data integer untuk jumlah elemen yang sama.

- Rasio penggunaan memori antar algoritma tetap konsisten terlepas dari tipe data.

6. Kesimpulan

Berdasarkan hasil eksperimen dan analisis, dapat disimpulkan bahwa:

a. **Untuk Dataset Kecil (≤10.000 elemen):**
   - Insertion Sort adalah pilihan yang baik karena implementasinya sederhana dan memiliki overhead yang rendah.
   - Jika data sudah hampir terurut, Insertion Sort dapat menjadi algoritma tercepat.

b. **Untuk Dataset Menengah (10.000-100.000 elemen):**
   - Quick Sort dan Shell Sort menunjukkan kinerja terbaik dari segi waktu eksekusi.
   - Shell Sort adalah pilihan yang baik jika memori terbatas dan implementasi sederhana diutamakan.

c. **Untuk Dataset Besar (>100.000 elemen):**
   - Quick Sort adalah algoritma terbaik untuk hampir semua kasus, dengan waktu eksekusi tercepat.
   - Merge Sort adalah alternatif yang baik jika stabilitas urutan elemen yang sama penting atau jika kinerja terburuk yang deterministik diperlukan.
   - Algoritma $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort sebaiknya dihindari untuk dataset besar.

d. **Pertimbangan Memori:**
   - Jika memori sangat terbatas, hindari Merge Sort karena kebutuhan memori tambahan O(n).
   - Quick Sort dengan optimasi tail recursion memberikan keseimbangan yang baik antara kinerja waktu dan penggunaan memori.

e. **Pertimbangan Stabilitas:**
   - Jika stabilitas (urutan elemen yang sama tidak berubah) penting, Merge Sort adalah pilihan terbaik di antara algoritma yang efisien.
   - Quick Sort dan Shell Sort tidak stabil, sementara Bubble Sort dan Insertion Sort stabil namun lambat untuk dataset besar.

f. **Rekomendasi Umum:**

- Untuk penggunaan umum, Quick Sort dengan pemilihan pivot yang baik (seperti median-of-three) adalah pilihan terbaik.
- Untuk sistem dengan memori terbatas, Shell Sort menawarkan kompromi yang baik antara kinerja dan kebutuhan memori.
- Untuk aplikasi yang memerlukan sorting stabil yang efisien, Merge Sort adalah pilihan terbaik.