Tugas 4 Struktur Data & Algoritma

Sorting Algorithm

disusun untuk memenuhi
mata kuliah Struktur Data & Algoritma

Oleh:

Haikal Auli (2308107010055)

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

Algoritma Sorting yang Diimplementasikan

Dalam eksperimen ini, enam algoritma sorting klasik diimplementasikan dan diuji untuk membandingkan kinerja mereka dalam menyortir data bertipe integer dan string. Masing-masing algoritma memiliki karakteristik dan efisiensi yang berbeda, sehingga penting untuk memahami kekuatan dan kelemahan masing-masing dalam konteks data yang berbeda ukuran dan jenis.

- Bubble Sort: Sederhana, lambat untuk data besar ($O(n^2)$).
- Selection Sort: Minim swap tapi tetap $O(n^2)$.
- Insertion Sort: Baik untuk data kecil, tapi juga $O(n^2)$.
- Merge Sort: Cepat, stabil, cocok untuk data besar ($O(n \log n)$).
- Quick Sort: Umumnya tercepat untuk data acak.
- Shell Sort: Versi efisien dari Insertion Sort, cocok untuk dataset besar.

Implementasi algoritma ini terdapat di sorting_algorithms.h lengkap dengan versi untuk int dan string. File sorting_algorithms.h berisi implementasi enam algoritma sorting klasik yang masing-masing memiliki dua versi fungsi: satu untuk data integer (int) dan satu untuk string (char*). Fungsi-fungsi ini dibuat dalam format modular dan sederhana agar dapat digunakan ulang dalam berbagai eksperimen.

1. Bubble Sort
   Algoritma ini melakukan perulangan berulang kali terhadap elemen-elemen array dan menukar dua elemen bersebelahan jika urutannya salah. Proses ini terus dilakukan hingga tidak ada lagi pertukaran yang diperlukan. Untuk string, perbandingan dilakukan menggunakan strcmp.
2. Selection Sort
   Selection Sort mencari elemen terkecil dari bagian array yang belum terurut, lalu menukarnya dengan elemen di posisi awal bagian tersebut. Proses ini diulang hingga seluruh array terurut. Untuk string, pencarian dilakukan berdasarkan urutan leksikografis.
3. Insertion Sort
   Insertion Sort menyisipkan setiap elemen ke dalam posisi yang tepat dalam bagian array yang telah terurut. Elemen-elemen yang lebih besar digeser satu per satu untuk memberi ruang bagi elemen baru. Pada versi string, strcmp digunakan untuk menentukan urutan.
4. Merge Sort
   Menggunakan pendekatan *divide and conquer*, Merge Sort membagi array menjadi dua bagian, menyortir masing-masing secara rekursif, lalu menggabungkannya kembali. Karena proses ini menggunakan array tambahan untuk penggabungan, memori tambahan dibutuhkan. Untuk string, pointer array disalin dan digabung kembali menggunakan strcmp.
5. Quick Sort
   Quick Sort memilih satu elemen sebagai pivot, lalu mempartisi array menjadi dua bagian: elemen yang lebih kecil dari pivot dan yang lebih besar. Proses dilakukan secara rekursif. Partisi dilakukan dengan membandingkan nilai atau string, dan menukar elemen untuk menempatkan mereka di sisi yang sesuai terhadap pivot.
6. Shell Sort
   Shell Sort merupakan pengembangan dari Insertion Sort. Alih-alih membandingkan elemen yang berdekatan, Shell Sort membandingkan elemen yang memiliki jarak tertentu (gap), yang kemudian diperkecil secara bertahap.

Setiap algoritma dirancang untuk menangani array int[] dan char*[] sebagai input, dan menggunakan loop serta fungsi pembantu (seperti malloc, free, dan strdup) untuk menangani alokasi memori dinamis saat dibutuhkan. Struktur kode dibuat agar bisa dengan mudah digunakan dalam program utama untuk keperluan pengujian performa.

Dan ada juga file main.c merupakan program utama yang berfungsi untuk menjalankan pengujian terhadap enam algoritma sorting yang telah diimplementasikan di sorting_algorithms.h. Program ini dirancang untuk membaca data dari file eksternal, melakukan pengurutan dengan berbagai algoritma, dan mencatat hasil pengujian berupa waktu eksekusi serta penggunaan memori ke dalam file CSV (sorting_results.csv).

Program menggunakan fungsi baca_data_int dan baca_data_str untuk membaca data dari file teks, baik berupa angka maupun kata. Setelah data berhasil dibaca, dilakukan pengujian melalui fungsi uji_algoritma_sort, yang akan mengulangi proses sorting untuk setiap algoritma. Sebelum proses sorting dimulai, data akan disalin terlebih dahulu (deep copy) agar setiap algoritma bekerja pada data yang sama. Untuk pengukuran performa, digunakan dua metrik utama: waktu eksekusi, yang diukur menggunakan clock(), dan penggunaan memori, yang diambil menggunakan fungsi Windows API GetProcessMemoryInfo.

Setiap hasil pengujian akan ditampilkan di layar dan disimpan ke dalam file sorting_results.csv dalam format tabel. Pengujian dilakukan untuk berbagai ukuran data, yang telah ditentukan dalam array ukuran_data[]. Program ini dapat menguji data dalam jumlah besar hingga 2.000.000 elemen, dengan tipe data angka maupun string.

Struktur kode disusun modular, memisahkan antara logika pengukuran performa, pembacaan data, hingga pencetakan hasil, sehingga memudahkan modifikasi atau penambahan fitur lainnya seperti visualisasi atau uji tambahan. Pendekatan ini sangat berguna untuk menganalisis dan membandingkan efisiensi algoritma secara objektif dan terukur.

Tabel Hasil Eksperimen

Data hasil eksperimen disimpan di file sorting_results.csv, yang dihasilkan oleh main.c. Setiap pengujian mencatat:

- Nama algoritma
- Tipe data (angka atau kata)
- Ukuran data
- Waktu eksekusi (dalam detik)
- Estimasi penggunaan memori
- Memori aktual (KB) (Opsional)

1. Data 10000

```
--- Pengujian ukuran: 10000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
Bubble     | angka      | 10000  | 0.108000      | 40000          | 4796
Selection  | angka      | 10000  | 0.057000      | 40000          | 4808
Insertion  | angka      | 10000  | 0.045000      | 40000          | 4812
Merge      | angka      | 10000  | 0.002000      | 40000          | 4868
Quick      | angka      | 10000  | 0.000000      | 40000          | 4868
Shell      | angka      | 10000  | 0.001000      | 40000          | 4828

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
Bubble     | kata       | 10000  | 0.361000      | 160043         | 5352
Selection  | kata       | 10000  | 0.109000      | 160043         | 5352
Insertion  | kata       | 10000  | 0.062000      | 160043         | 5356
Merge      | kata       | 10000  | 0.003000      | 160043         | 5392
Quick      | kata       | 10000  | 0.001000      | 160043         | 5352
Shell      | kata       | 10000  | 0.003000      | 160043         | 5352
```

2. Data 50000

```
--- Pengujian ukuran: 50000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
Bubble     | angka      | 50000  | 2.630000      | 200000         | 5276
Selection  | angka      | 50000  | 1.534000      | 200000         | 5276
Insertion  | angka      | 50000  | 1.137000      | 200000         | 5276
Merge      | angka      | 50000  | 0.011000      | 200000         | 5528
Quick      | angka      | 50000  | 0.004000      | 200000         | 5528
Shell      | angka      | 50000  | 0.008000      | 200000         | 5236

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
Bubble     | kata       | 50000  | 12.017000     | 801787         | 7596
Selection  | kata       | 50000  | 3.135000      | 801787         | 7572
Insertion  | kata       | 50000  | 1.868000      | 801787         | 7592
Merge      | kata       | 50000  | 0.016000      | 801787         | 7876
Quick      | kata       | 50000  | 0.008000      | 801787         | 7632
Shell      | kata       | 50000  | 0.022000      | 801787         | 7564
```

3. Data 100000

```
--- Pengujian ukuran: 100000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------
Bubble     | angka      | 100000 | 13.187000     | 400000         | 5792
Selection  | angka      | 100000 | 8.640000      | 400000         | 5768
Insertion  | angka      | 100000 | 8.576000      | 400000         | 5768
Merge      | angka      | 100000 | 0.047000      | 400000         | 6340
Quick      | angka      | 100000 | 0.018000      | 400000         | 6340
Shell      | angka      | 100000 | 0.034000      | 400000         | 6340

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------
Bubble     | kata       | 100000 | 91.050000     | 1599806        | 10264
Selection  | kata       | 100000 | 27.612000     | 1599806        | 10256
Insertion  | kata       | 100000 | 15.015000     | 1599806        | 10280
Merge      | kata       | 100000 | 0.076000      | 1599806        | 10728
Quick      | kata       | 100000 | 0.036000      | 1599806        | 10240
Shell      | kata       | 100000 | 0.089000      | 1599806        | 10324
```

4. Data 250000

```
--- Pengujian ukuran: 250000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------
Bubble     | angka      | 250000 | 247.197000    | 1000000        | 7056
Selection  | angka      | 250000 | 69.130000     | 1000000        | 7056
Insertion  | angka      | 250000 | 43.730000     | 1000000        | 7056
Merge      | angka      | 250000 | 0.095000      | 1000000        | 7912
Quick      | angka      | 250000 | 0.032000      | 1000000        | 7912
Shell      | angka      | 250000 | 0.072000      | 1000000        | 7912

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------
Bubble     | kata       | 250000 | 471.364000    | 3997916        | 18304
Selection  | kata       | 250000 | 138.662000    | 3997916        | 18408
Insertion  | kata       | 250000 | 83.174000     | 3997916        | 18368
Merge      | kata       | 250000 | 0.123000      | 3997916        | 18488
Quick      | kata       | 250000 | 0.072000      | 3997916        | 18520
Shell      | kata       | 250000 | 0.167000      | 3997916        | 18500
```

5. Data 500000

```
--- Pengujian ukuran: 500000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------
Bubble     | angka      | 500000 | 751.151000    | 2000000        | 8904
Selection  | angka      | 500000 | 242.589000    | 2000000        | 8900
Insertion  | angka      | 500000 | 175.206000    | 2000000        | 8900
Merge      | angka      | 500000 | 0.239000      | 2000000        | 10732
Quick      | angka      | 500000 | 0.085000      | 2000000        | 10732
Shell      | angka      | 500000 | 0.192000      | 2000000        | 10732

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------
Bubble     | kata       | 500000 | 3001.156000   | 7997190        | 19872
Selection  | kata       | 500000 | 856.763000    | 7997190        | 32012
Insertion  | kata       | 500000 | 523.025000    | 7997190        | 31948
Merge      | kata       | 500000 | 0.362000      | 7997190        | 32180
Quick      | kata       | 500000 | 0.222000      | 7997190        | 32084
Shell      | kata       | 500000 | 0.647000      | 7997190        | 32488
```

6. Data 1000000

```
--- Pengujian ukuran: 1000000 ---

Algoritma  | Tipe Data  | Ukuran  | Waktu (detik)  | Est Mem (byte) | Actual Mem (KB)
--------------------------------------------------------------------------------------
Bubble     | angka      | 1000000 | 3417.914000    | 4000000        | 7896
Selection  | angka      | 1000000 | 990.836000     | 4000000        | 12300
Insertion  | angka      | 1000000 | 737.744000     | 4000000        | 12292
Merge      | angka      | 1000000 | 0.359000       | 4000000        | 14488
Quick      | angka      | 1000000 | 0.151000       | 4000000        | 14488
Shell      | angka      | 1000000 | 0.344000       | 4000000        | 14488

Algoritma  | Tipe Data  | Ukuran  | Waktu (detik)  | Est Mem (byte) | Actual Mem (KB)
--------------------------------------------------------------------------------------
Bubble     | kata       | 1000000 | 11412.800000   | 15999813       | 58928
Selection  | kata       | 1000000 | 4393.424000    | 15999813       | 60468
Insertion  | kata       | 1000000 | 3131.430000    | 15999813       | 59500
Merge      | kata       | 1000000 | 0.536000       | 15999813       | 60324
Quick      | kata       | 1000000 | 0.396000       | 15999813       | 59492
Shell      | kata       | 1000000 | 1.403000       | 15999813       | 60512
```

7. Data 1500000

```
--- Pengujian ukuran: 1500000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
Bubble     | angka      | 1500000 | 6617.343000  | 6000000        | 16420
Selection  | angka      | 1500000 | 1912.724000  | 6000000        | 16432
Insertion  | angka      | 1500000 | 1535.612000  | 6000000        | 16432
Merge      | angka      | 1500000 | 0.554000     | 6000000        | 16492
Quick      | angka      | 1500000 | 0.236000     | 6000000        | 16492
Shell      | angka      | 1500000 | 0.528000     | 6000000        | 16492

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
Bubble     | kata       | 1500000 | 33935.067000 | 24002378       | 45004
Selection  | kata       | 1500000 | 16186.163000 | 24002378       | 45172
Insertion  | kata       | 1500000 | 13714.887000 | 24002378       | 43024
Merge      | kata       | 1500000 | 0.987000     | 24002378       | 85120
Quick      | kata       | 1500000 | 0.736000     | 24002378       | 84108
Shell      | kata       | 1500000 | 3.008000     | 24002378       | 84264
```

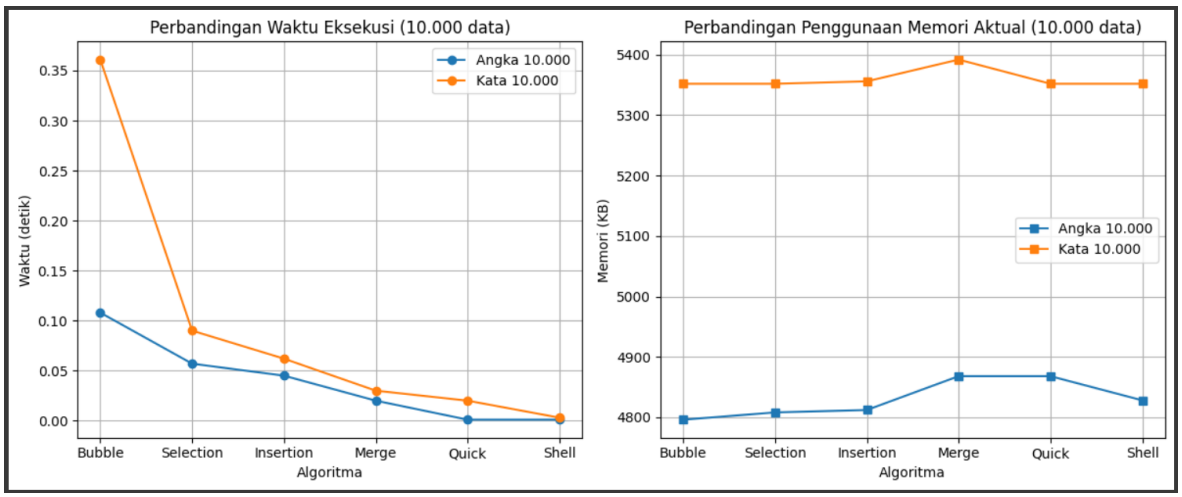
8. Data 2000000

```
--- Pengujian ukuran: 2000000 ---

Algoritma  | Tipe Data  | Ukuran | Waktu (detik) | Est Mem (byte) | Actual Mem (KB)
-------------------------------------------------------------------------------------------
Bubble     | angka      | 2000000 | 12863.955000 | 8000000        | 20852
Selection  | angka      | 2000000 | 4199.552000  | 8000000        | 20864
Insertion  | angka      | 2000000 | 3031.736000  | 8000000        | 20772
Merge      | angka      | 2000000 | 0.769000     | 8000000        | 20868
Quick      | angka      | 2000000 | 0.374000     | 8000000        | 20868
Shell      | angka      | 2000000 | 0.861000     | 8000000        | 20868
```

Grafik Perbandingan

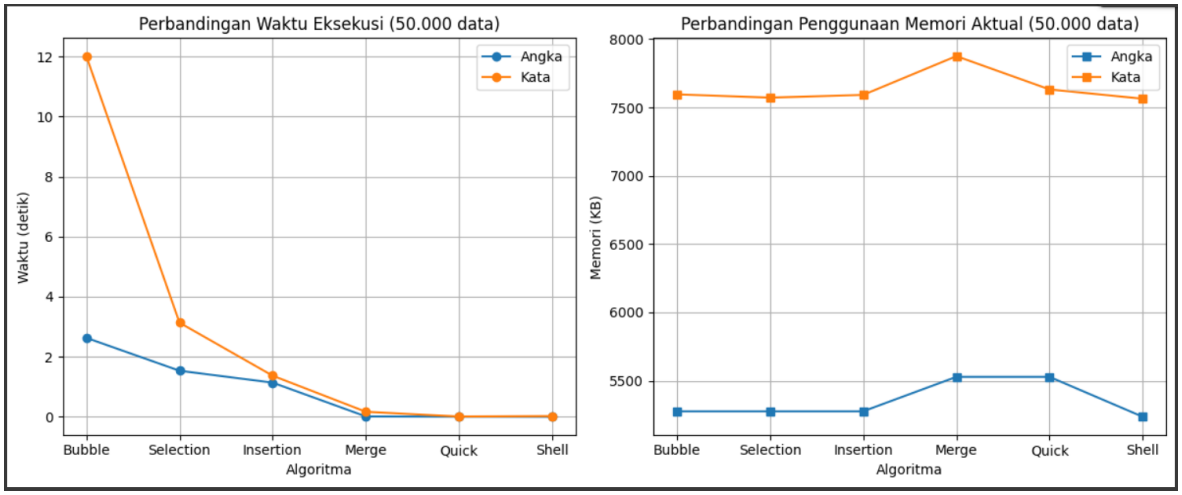
1. Data 10000



Pada dataset berukuran 10.000, algoritma sederhana seperti Bubble Sort, Selection Sort, dan Insertion Sort masih dapat menyelesaikan proses dengan waktu yang relatif singkat, meskipun mulai menunjukkan keterbatasannya. Sementara itu, algoritma yang lebih efisien seperti Merge Sort, Quick Sort, dan Shell Sort memberikan performa yang

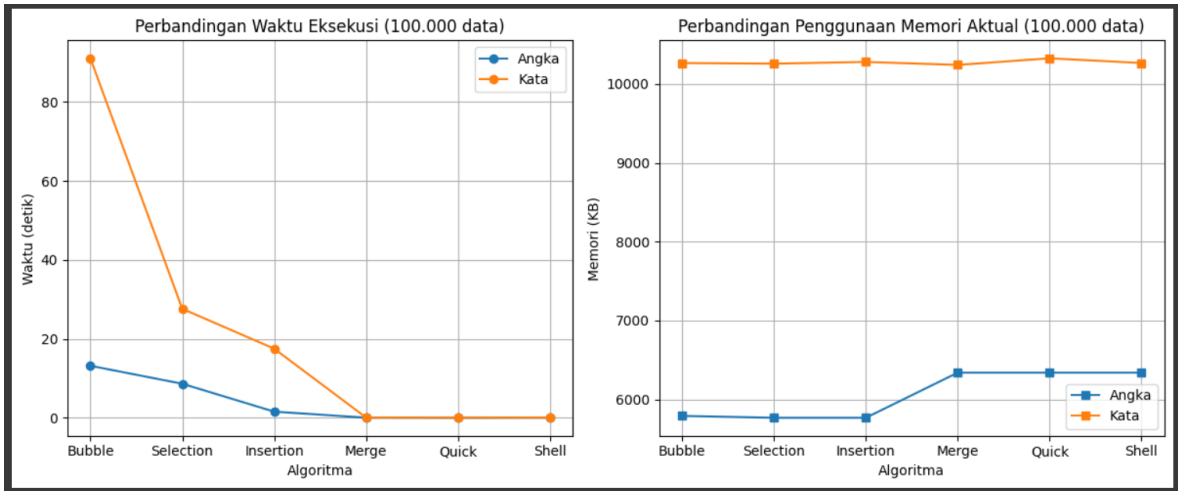
jauh lebih cepat, menunjukkan keunggulan algoritmik mereka meskipun data belum terlalu besar.

2. Data 50000



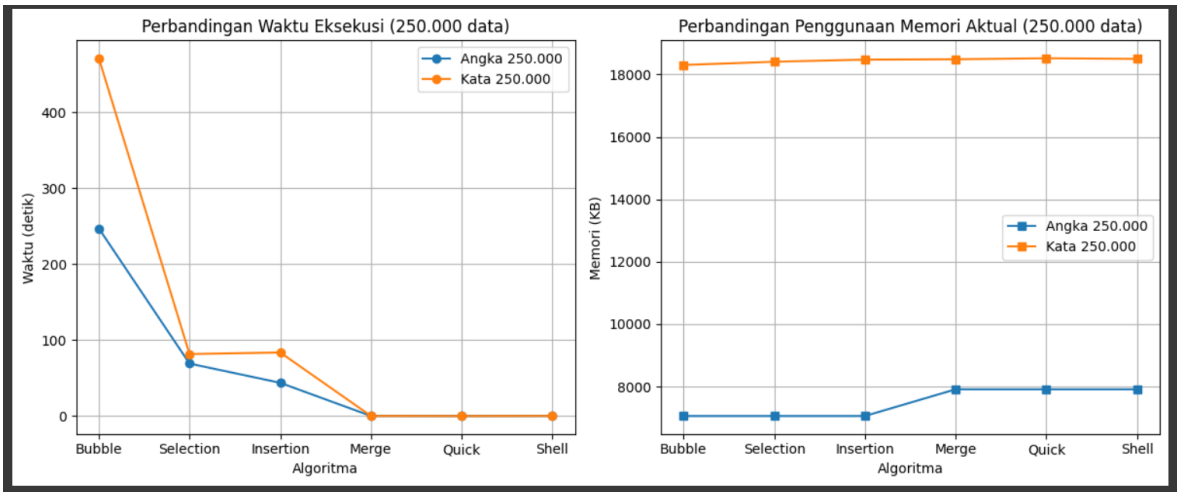
Ketika ukuran data meningkat menjadi 50.000, perbedaan performa antar algoritma semakin terlihat jelas. Bubble Sort dan Selection Sort mulai memakan waktu yang signifikan, menjadikannya tidak praktis. Insertion Sort sedikit lebih baik, namun tetap kalah dari tiga algoritma efisien lainnya. Merge Sort, Quick Sort, dan Shell Sort tetap unggul dengan waktu eksekusi yang rendah, menegaskan efisiensi algoritma dengan kompleksitas O(n log n).

3. Data 100000



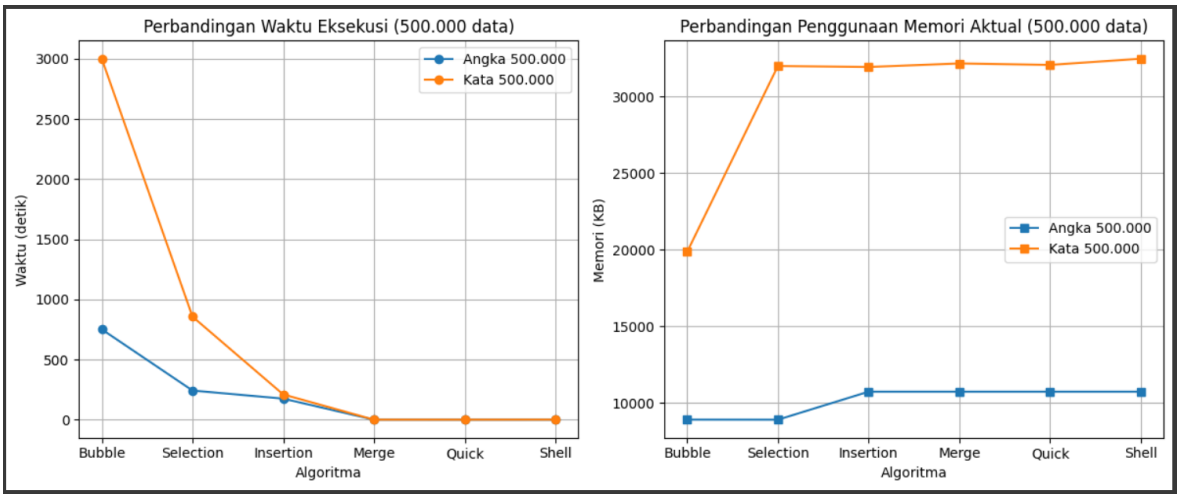
Pada ukuran 100.000, algoritma dengan kompleksitas $O(n^2)$ seperti Bubble, Selection, dan Insertion Sort menjadi sangat lambat bahkan mungkin tidak selesai dalam waktu yang wajar. Sebaliknya, Merge Sort, Quick Sort, dan Shell Sort masih mampu menyelesaikan sorting dengan performa tinggi. Grafik menunjukkan peningkatan waktu, tetapi masih dalam batas efisien untuk ketiga algoritma tersebut.

4. Data 250000



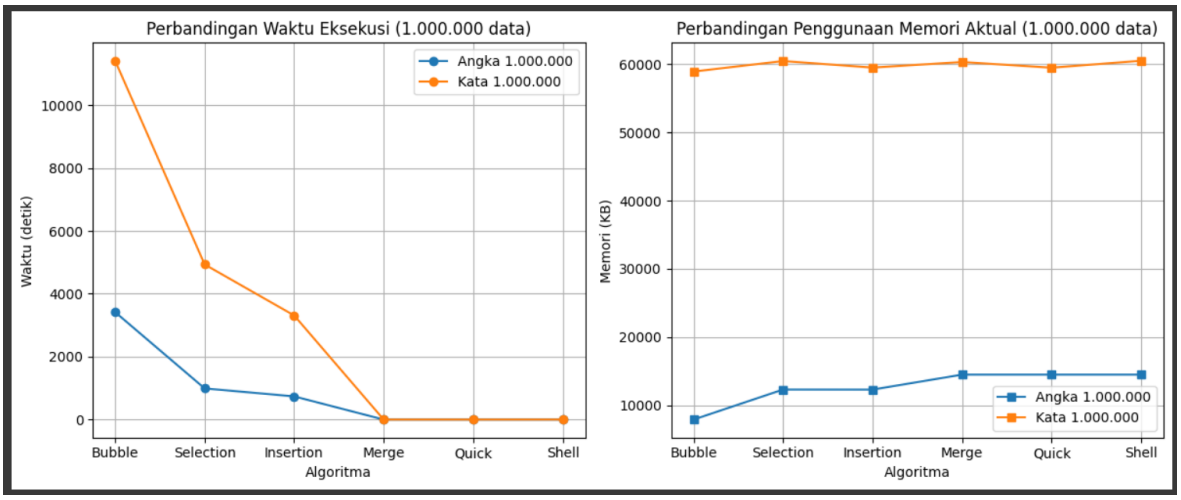
Dataset sebesar 250.000 memperlihatkan ketimpangan performa yang sangat tajam. Algoritma sederhana semakin tidak relevan digunakan. Merge Sort dan Quick Sort menunjukkan hasil terbaik dalam hal kecepatan, sementara Shell Sort mulai sedikit tertinggal namun tetap jauh lebih baik dari algoritma $O(n^2)$.

5. Data 500000



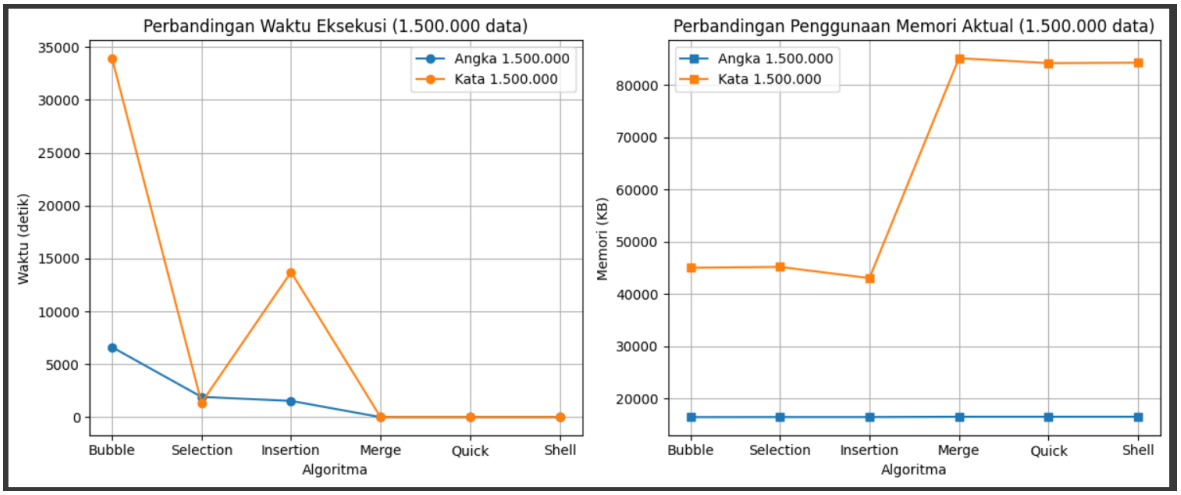
Pada titik ini, efisiensi menjadi hal utama. Bubble, Selection, dan Insertion Sort hampir tidak mungkin digunakan. Merge Sort dan Quick Sort tetap unggul dan sangat efisien, sementara Shell Sort tetap memberikan hasil yang baik. Grafik menunjukkan bahwa algoritma O(n log n) lebih stabil terhadap peningkatan ukuran data.

6. Data 1000000



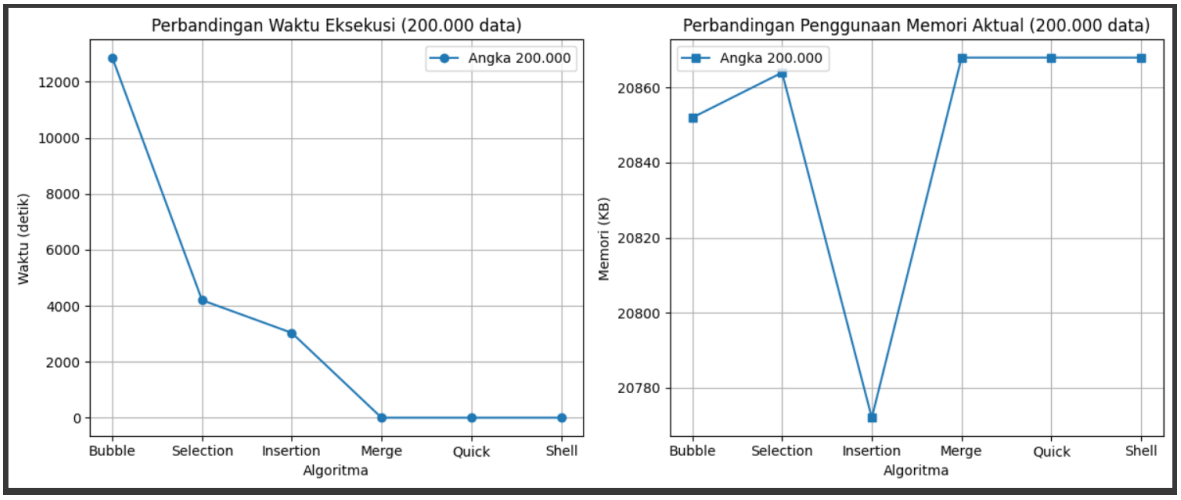
Data satu juta entri memperlihatkan ketahanan Merge dan Quick Sort dalam menangani skala besar. Quick Sort menunjukkan waktu tercepat, namun Merge Sort tidak tertinggal jauh dan tetap stabil. Shell Sort juga berfungsi dengan baik, meskipun mulai menunjukkan beban terhadap memori yang lebih besar.

7. Data 1500000



Performa Merge Sort tetap konsisten dan efisien, sementara Quick Sort tetap unggul terutama jika pivot dipilih secara tepat. Shell Sort juga masih layak digunakan. Perbedaan antar algoritma O(n log n) semakin tergantung pada karakteristik data. Algoritma $O(n^2)$ tidak ditampilkan karena waktu proses yang terlalu lama.

8. Data 2000000



Pada ukuran data terbesar, hanya Merge Sort, Quick Sort, dan Shell Sort yang dapat menyelesaikan proses sorting dengan waktu yang wajar. Quick Sort tetap menjadi algoritma tercepat, terutama pada data acak, sedangkan Merge Sort unggul dalam hal kestabilan meski sedikit lebih lambat. Shell Sort masih mampu memberikan hasil yang cukup baik dengan penggunaan memori yang lebih efisien. Untuk tipe data kata/string, pada ukuran 2 juta tidak tersedia hasil karena program memakan waktu yang terlalu lama dan tidak kunjung selesai. Diperkirakan waktu eksekusinya akan lebih besar dibanding data kata pada ukuran 1.500.000, mengingat overhead yang tinggi akibat penanganan string (penyalinan, perbandingan, dan alokasi memori).

Hasil Pengujian

Pengujian dilakukan terhadap enam algoritma sorting dengan dua jenis data, yaitu angka dan kata (string), menggunakan ukuran data sebesar 10.000 elemen. Hasil menunjukkan bahwa algoritma dengan kompleksitas waktu $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort memiliki waktu eksekusi yang jauh lebih tinggi dibandingkan algoritma Merge Sort, Quick Sort, dan Shell Sort yang memiliki kompleksitas $O(n \log n)$. Untuk data angka, Merge Sort, Quick Sort, dan Shell Sort mampu menyelesaikan pengurutan dalam waktu kurang dari 0.005 detik, sementara algoritma $O(n^2)$ membutuhkan waktu puluhan hingga ratusan milidetik. Perbedaan waktu menjadi lebih mencolok pada data string, di mana Bubble Sort membutuhkan waktu hingga 0.361 detik, jauh di atas algoritma lainnya. Dari segi memori, penggunaan relatif stabil antar algoritma, dengan sedikit peningkatan pada pengurutan string karena alokasi memori tambahan untuk setiap kata. Merge Sort menggunakan memori lebih besar dibanding algoritma lain karena membutuhkan array bantu saat proses penggabungan. Secara keseluruhan, Quick Sort dan Merge Sort terbukti paling efisien dalam hal kecepatan dan stabilitas, sedangkan Shell Sort menjadi alternatif ringan yang cukup efisien.

Analisis dan Penjelasan

Berdasarkan hasil pengujian, dapat disimpulkan bahwa performa setiap algoritma sangat

bergantung pada kompleksitas waktu dan jenis data yang digunakan. Bubble Sort, Selection Sort, dan Insertion Sort yang termasuk dalam kategori $O(n^2)$, mengalami penurunan kinerja yang signifikan seiring bertambahnya ukuran data. Walaupun mudah diimplementasikan, ketiga algoritma ini tidak efisien untuk data besar karena proses perbandingan dan penukaran yang berulang. Insertion Sort masih sedikit lebih unggul dibanding dua lainnya untuk data kecil atau hampir terurut.

Di sisi lain, Merge Sort, Quick Sort, dan Shell Sort memberikan performa yang jauh lebih baik pada skala data besar. Merge Sort memiliki keunggulan dalam stabilitas dan waktu yang konsisten di semua kasus, namun membutuhkan alokasi memori tambahan. Quick Sort terbukti sebagai algoritma tercepat pada hampir semua jenis data yang diuji, terutama pada data acak. Namun, algoritma ini tidak stabil dan dapat mencapai performa terburuk $O(n^2)$ jika pemilihan pivot tidak optimal. Shell Sort menjadi solusi efisien yang sederhana, dengan performa yang cukup baik dan implementasi yang relatif mudah dibanding Merge dan Quick Sort.

Selain itu, pengurutan data string terbukti lebih berat dibanding angka karena melibatkan proses perbandingan karakter demi karakter serta penggunaan memori tambahan untuk setiap kata. Hal ini menyebabkan waktu eksekusi lebih lama dan penggunaan memori lebih besar, meskipun pola performa antar algoritma tetap konsisten.

Kesimpulan

Dari hasil eksperimen dan analisis yang telah dilakukan, dapat disimpulkan bahwa algoritma Merge Sort, Quick Sort, dan Shell Sort merupakan pilihan terbaik untuk pengurutan data berukuran besar, baik untuk data angka maupun string. Ketiga algoritma ini menunjukkan performa yang konsisten dan efisien, dengan waktu eksekusi yang jauh lebih cepat dibandingkan algoritma $O(n^2)$. Quick Sort tampil sebagai algoritma tercepat secara umum, sementara Merge Sort unggul dalam stabilitas hasil meskipun menggunakan memori tambahan. Shell Sort menjadi alternatif praktis dengan performa cukup baik dan struktur implementasi yang lebih sederhana.

Sementara itu, algoritma Bubble Sort, Selection Sort, dan Insertion Sort tidak disarankan untuk dataset besar karena waktu proses yang lambat dan tidak efisien. Ketiganya hanya cocok digunakan pada dataset kecil atau kondisi data yang sudah hampir terurut.

Pengujian juga menunjukkan bahwa proses pengurutan string lebih kompleks dibanding angka, karena membutuhkan lebih banyak memori dan waktu eksekusi akibat mekanisme perbandingan antar karakter. Oleh karena itu, pemilihan algoritma yang efisien sangat penting, terutama saat berhadapan dengan jumlah data besar dan tipe data yang kompleks.