*Laporan MataKuliah Struktur Data dan Algoritma*

**TUGAS SDA D**

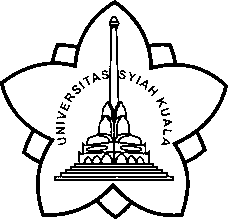
disusun untuk memenuhi tugas

mata kuliah struktur data dan algoritma

Oleh:

**FIRAH MAULIDA**

**2308107010034**



**JURUSAN INFORMATIKA**

**FAKULTAS METEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS SYIAH KUALA**

**DARUSSALAM, BANDA ACEH**

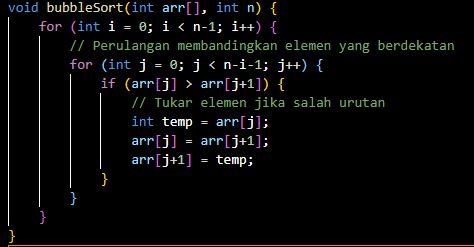
**2024**

1. **Deskripsi Algoritma**
2. **Bubble Sort**

Bubble Sort adalah algoritma sorting yang sederhana namun tidak efisien untuk data besar. Algoritma ini bekerja dengan membandingkan elemen-elemen yang berdekatan dalam array dan menukarnya jika elemen pertama lebih besar dari yang kedua. Proses ini diulang hingga tidak ada lagi pertukaran yang dilakukan, menandakan bahwa array sudah terurut.

**Waktu Kompleksitas:**

* Best case: O(n) jika array sudah terurut.
* Worst case dan Average case: O(n²).

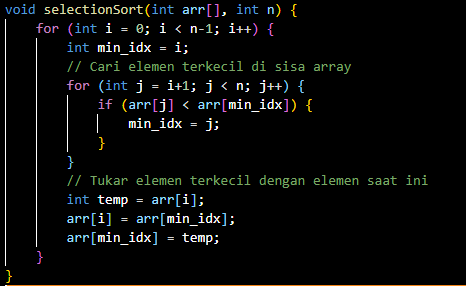


1. **Selection Sort**

Selection Sort juga merupakan algoritma yang sederhana, di mana algoritma ini bekerja dengan mencari elemen terkecil (atau terbesar, tergantung urutan) dalam array dan menukarnya dengan elemen pertama yang belum terurut. Proses ini diulang untuk elemen berikutnya hingga seluruh array terurut.

**Waktu Kompleksitas:**

* Best case, Worst case, dan Average case: O(n²).

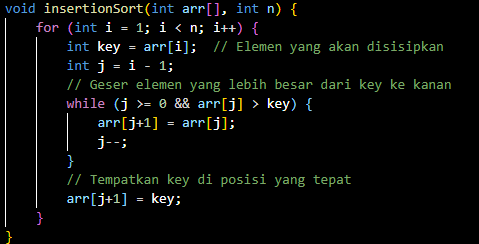


1. **Insertion Sort**

Insertion Sort bekerja dengan cara mengurutkan array secara bertahap dengan memindahkan elemen-elemen ke posisi yang sesuai di bagian array yang sudah terurut. Proses ini dimulai dari elemen kedua dan bergerak ke depan, memasukkan setiap elemen ke dalam subarray terurut dengan cara membandingkan dan memindahkannya ke posisi yang tepat.

**Waktu Kompleksitas:**

* Best case: O(n) jika array sudah terurut.
* Worst case dan Average case: O(n²).

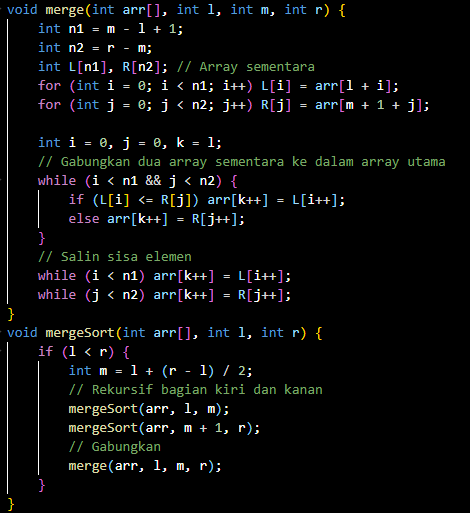


1. **Merge Sort**

Merge Sort adalah algoritma yang menggunakan pendekatan divide and conquer. Algoritma ini membagi array menjadi dua bagian, mengurutkan kedua bagian secara rekursif, dan akhirnya menggabungkannya menjadi satu array yang terurut. Merge Sort merupakan algoritma yang sangat efisien, terutama untuk data besar.

**Waktu Kompleksitas:**

* Best case, Worst case, dan Average case: O(n log n).

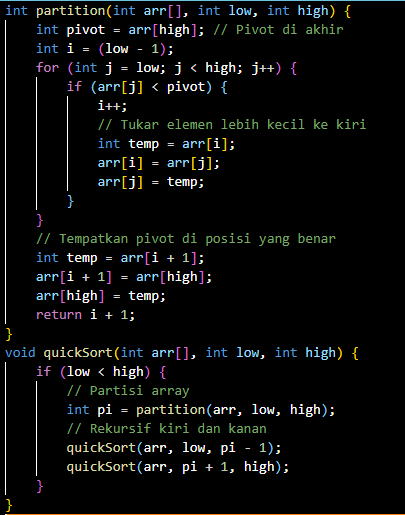


1. **Quick Sort**

Quick Sort juga menggunakan metode divide and conquer, di mana algoritma memilih elemen pivot dan membagi array menjadi dua bagian: elemen yang lebih kecil dan elemen yang lebih besar dari pivot. Kemudian, kedua bagian tersebut diurutkan secara rekursif. Quick Sort umumnya lebih cepat daripada Merge Sort dalam prakteknya meskipun memiliki kompleksitas yang sama pada kasus terburuk.

**Waktu Kompleksitas:**

* Best case dan Average case: O(n log n).
* Worst case: O(n²) (terjadi jika pivot yang dipilih tidak efektif).

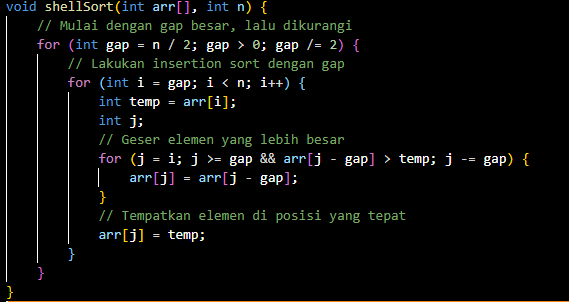


1. **Shell Sort**

Shell Sort adalah peningkatan dari Insertion Sort yang memperkenalkan konsep "gap" atau jarak antara elemen-elemen yang dibandingkan. Dengan menggunakan gap yang lebih besar pada iterasi awal, Shell Sort dapat memperbaiki efisiensi Insertion Sort untuk data yang lebih besar.

**Waktu Kompleksitas:**

* Best case: O(n log n) (tergantung pada urutan gap yang digunakan).
* Worst case dan Average case: O(n^(3/2)) atau lebih buruk tergantung pada implementasi gap.



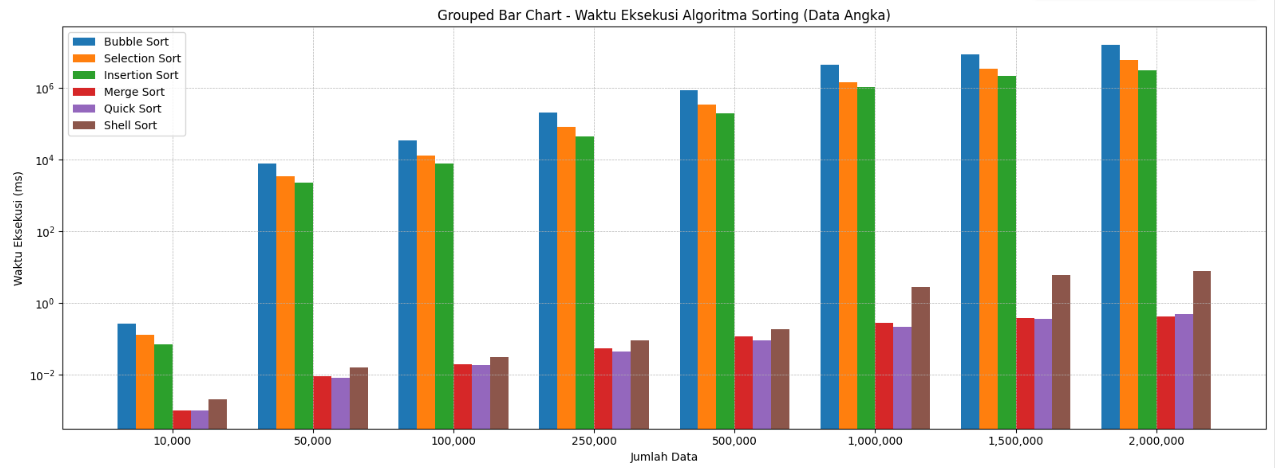
1. **Tabel Hasil Ekperimen**
2. **Tabel Angka**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Algoritma** | **Ukuran** | **Waktu (detik)** | **Memori (KB)** |
| 1. | Bubble Sort | 10.000 | 0.262000 detik | 39 KB |
| 50.000 | 7.742000 detik | 195 KB |
| 100.000 | 33.999000 detik | 391 KB |
| 250.000 | 210.799000 detik | 977 KB |
| 500.000 | 866.400000 detik | 1953 KB |
| 1.000.000 | 4497.204000 detik | 3906 KB |
| 1.500.000 | 8636.046000 detik | 5859 KB |
| 2.000.000 | 15763.723000 detik | 7813 KB |
| 2. | Selection Sort | 10.000 | 0.130000 detik | 39 KB |
| 50.000 | 3.506000 detik | 195 KB |
| 100.000 | 12.908000 detik | 391 KB |
| 250.000 | 82.955000 detik | 977 KB |
| 500.000 | 344.391000 detik | 1953 KB |
| 1.000.000 | 1468.058000 detik | 3906 KB |
| 1.500.000 | 3490.729000 detik | 5859 KB |
| 2.000.000 | 5905.591000 detik | 7813 KB |
| 3. | Insertion Sort | 10.000 | 0.069000 detik | 39 KB |
| 50.000 | 2.282000 detik | 195 KB |
| 100.000 | 7.564000 detik | 391 KB |
| 250.000 | 44.326000 detik | 977 KB |
| 500.000 | 194.203000 detik | 1953 KB |
| 1.000.000 | 1065.262000 detik | 3906 KB |
| 1.500.000 | 2221.561000 detik | 5859 KB |
| 2.000.000 | 3061.680000 detik | 7813 KB |
| 4. | Merge Sort | 10.000 | 0.001000 detik | 39 KB |
| 50.000 | 0.009000 detik | 195 KB |
| 100.000 | 0.019000 detik | 391 KB |
| 250.000 | 0.055000 detik | 977 KB |
| 500.000 | 0.116000 detik | 1953 KB |
| 1.000.000 | 0.277000 detik | 3906 KB |
| 1.500.000 | 0.379000 detik | 5859 KB |
| 2.000.000 | 0.410000 detik | 7813 KB |
| 5. | Quick Sort | 10.000 | 0.001000 detik | 39 KB |
| 50.000 | 0.008000 detik | 195 KB |
| 100.000 | 0.018000 detik | 391 KB |
| 250.000 | 0.043000 detik | 977 KB |
| 500.000 | 0.091000 detik | 1953 KB |
| 1.000.000 | 0.216000 detik | 3906 KB |
| 1.500.000 | 0.352000 detik | 5859 KB |
| 2.000.000 | 0.489000 detik | 7813 KB |
| 6. | Shell Sort | 10.000 | 0.002000 detik | 39 KB |
| 50.000 | 0.016000 detik | 195 KB |
| 100.000 | 0.030000 detik | 391 KB |
| 250.000 | 0.091000 detik | 977 KB |
| 500.000 | 0.186000 detik | 1953 KB |
| 1.000.000 | 2.715000 detik | 3906 KB |
| 1.500.000 | 5.850000 detik | 5859 KB |
| 2.000.000 | 7.768000 detik | 7813 KB |

1. **Tabel Kata**

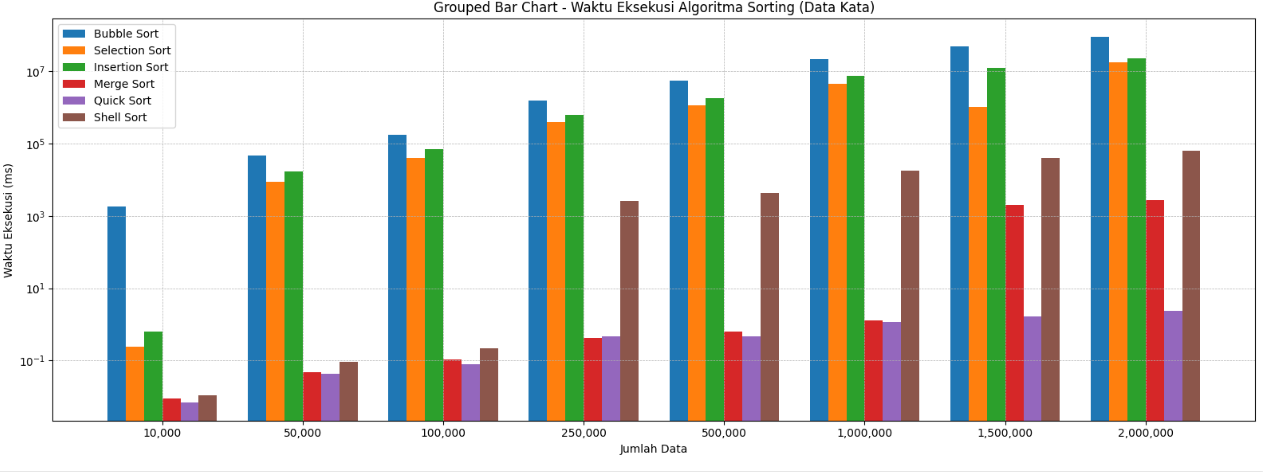
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Algoritma** | **Ukuran** | **Waktu (detik)** | **Memori (KB)** |
| 1. | Bubble Sort | 10.000 | 1.786000 detik | 977 KB |
| 50.000 | 47.279000 detik | 4883 KB |
| 100.000 | 179.133000 detik | 9766 KB |
| 250.000 | 1559.882000 detik | 24414 KB |
| 500.000 | 5397.061000 detik | 48828 KB |
| 1.000.000 | 22184.694000 detik | 97656 KB |
| 1.500.000 | 49915.685000 detik | 146484 KB |
| 2.000.000 | 88738.834000 detik | 195312 KB |
| 2. | Selection Sort | 10.000 | 0.244000 detik | 977 KB |
| 50.000 | 8.943000 detik | 4883 KB |
| 100.000 | 39.470000 detik | 9766 KB |
| 250.000 | 392.310000 detik | 24414 KB |
| 500.000 | 1119.420000 detik | 48828 KB |
| 1.000.000 | 4507.404000 detik | 97656 KB |
| 1.500.000 | 10141.67000 detik | 146484 KB |
| 2.000.000 | 18029.618000 detik | 195312 KB |
| 3. | Insertion Sort | 10.000 | 0.635000 detik | 977 KB |
| 50.000 | 16.608000 detik | 4883 KB |
| 100.000 | 68.636000 detik | 9766 KB |
| 250.000 | 625.747000 detik | 24414 KB |
| 500.000 | 1824.651000 detik | 48828 KB |
| 1.000.000 | 7493.934000 detik | 97656 KB |
| 1.500.000 | 12489.655000 detik | 146484 KB |
| 2.000.000 | 22381.639000 detik | 195312 KB |
| 4. | Merge Sort | 10.000 | 0.009000 detik | 977 KB |
| 50.000 | 0.048000 detik | 4883 KB |
| 100.000 | 0.107000 detik | 9766 KB |
| 250.000 | 0.413000 detik | 24414 KB |
| 500.000 | 0.627000 detik | 48828 KB |
| 1.000.000 | 1.305000 detik | 97656 KB |
| 1.500.000 | 2.050000 detik | 146484 KB |
| 2.000.000 | 2.693000 detik | 195312 KB |
| 5. | Quick Sort | 10.000 | 0.007000 detik | 977 KB |
| 50.000 | 0.043000 detik | 4883 KB |
| 100.000 | 0.081000 detik | 9766 KB |
| 250.000 | 0.478000 detik | 24414 KB |
| 500.000 | 0.467000 detik | 48828 KB |
| 1.000.000 | 1.155000 detik | 97656 KB |
| 1.500.000 | 1.670000 detik | 146484 KB |
| 2.000.000 | 2.319000 detik | 195312 KB |
| 6. | Shell Sort | 10.000 | 0.011000 detik | 977 KB |
| 50.000 | 0.090000 detik | 4883 KB |
| 100.000 | 0.222000 detik | 9766 KB |
| 250.000 | 2.634000 detik | 24414 KB |
| 500.000 | 4.376000 detik | 48828 KB |
| 1.000.000 | 17.642000 detik | 97656 KB |
| 1.500.000 | 39.841000 detik | 146484 KB |
| 2.000.000 | 65.116000detik | 195312 KB |

1. **Grafik Perbandingan Waktu dan Memori**
2. **Grafik Perbandingan Waktu Pada data Angka**



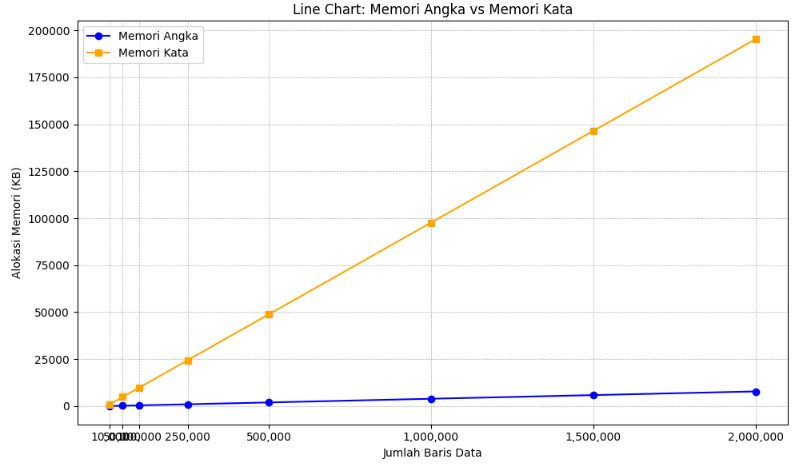
* **semakin banyak jumlah data yang diolah, semakin besar pula waktu eksekusi yang dibutuhkan** oleh seluruh algoritma.
* Bubble Sort menunjukkan waktu eksekusi paling tinggi pada seluruh ukuran data. Untuk dataset berisi 2.000.000 elemen, Bubble Sort memerlukan waktu sekitar 15.763 detik, menjadikannya algoritma dengan performa terburuk.
* Selection Sort dan Insertion Sort juga memperlihatkan kinerja yang kurang efisien. Meski sedikit lebih cepat dibandingkan Bubble Sort, keduanya tetap membutuhkan waktu yang besar, masing-masing sekitar 5.905 detik dan 3.061 detik untuk data sebanyak 2.000.000 elemen.
* Merge Sort dan Quick Sort menunjukkan performa paling efisien, dengan waktu eksekusi jauh lebih rendah dibandingkan algoritma lainnya. Pada data 2.000.000, Merge Sort dapat menyelesaikan proses hanya dalam 0,410 detik dan Quick Sort dalam 0,489 detik.
* Shell Sort berada di antara kedua kelompok tersebut: lebih cepat dibandingkan algoritma sederhana, namun tidak seefisien Merge Sort atau Quick Sort. Untuk 2.000.000 data, Shell Sort membutuhkan waktu 7.768 detik.

1. **Grafik Perbandingan Waktu Pada data Kata**



* Semakin besar ukuran data, semakin besar pula waktu eksekusi yang dibutuhkan oleh semua algoritma. Hal ini menunjukkan adanya pertumbuhan kompleksitas yang sejalan dengan pertambahan jumlah elemen yang harus diolah.
* **Bubble Sort** menunjukkan performa paling lambat di antara semua algoritma. Untuk dataset sebesar 2.000.000 elemen, Bubble Sort membutuhkan waktu sekitar **88.738 detik** dengan penggunaan memori sebesar **195.312 KB**. Ini menjadikan Bubble Sort algoritma dengan kinerja terburuk dalam pengujian ini.
* **Selection Sort** dan **Insertion Sort** juga memperlihatkan performa yang kurang efisien. Selection Sort membutuhkan waktu **18.029 detik** untuk mengolah 2.000.000 data, sementara Insertion Sort lebih lambat lagi dengan waktu **22.381 detik**, keduanya menggunakan memori sebesar **195.312 KB**.
* **Merge Sort** dan **Quick Sort** menunjukkan efisiensi yang jauh lebih baik dibandingkan algoritma lainnya. Pada data sebanyak 2.000.000 elemen, Merge Sort menyelesaikan proses sorting dalam waktu **2,693 detik** dan Quick Sort dalam waktu **2,319 detik**, dengan penggunaan memori yang sama yaitu **195.312 KB**.
* **Shell Sort** menempati posisi tengah antara algoritma sederhana dan algoritma efisien. Shell Sort membutuhkan waktu **65,116 detik** untuk menyelesaikan sorting pada dataset 2.000.000 elemen, dengan penggunaan memori yang sama, yaitu **195.312 KB**.

1. **Grafik Perbandingan Memori Pada data Angka dan Kata**



Grafik "Line Chart: Memori Angka vs Memori Kata" menunjukkan perbandingan penggunaan memori antara data bertipe angka dan data bertipe kata berdasarkan jumlah baris data yang diolah. Dari grafik terlihat bahwa alokasi memori meningkat secara linear untuk kedua tipe data seiring bertambahnya jumlah baris data. Namun, penggunaan memori untuk data kata jauh lebih besar dibandingkan data angka. Sebagai contoh, pada 1.000.000 baris data, memori yang digunakan untuk angka hanya sekitar 3.906 KB, sedangkan untuk kata mencapai sekitar 97.656 KB. Hal ini menunjukkan bahwa penyimpanan data bertipe kata memerlukan ruang yang jauh lebih besar, sekitar 25 kali lipat dibandingkan data angka. Perbedaan ini kemungkinan besar disebabkan oleh representasi data kata (string) yang secara alami lebih kompleks dan bervariasi dibandingkan representasi data numerik yang lebih sederhana dan tetap. Oleh karena itu, dalam pengolahan data berskala besar, tipe data yang digunakan sangat memengaruhi efisiensi penggunaan memori, dan data bertipe angka terbukti lebih hemat dalam penggunaan sumber daya.

1. **Analisis dan Kesimpulan**