

Laporan Tugas 4

Nama: Ikram Al Ghiffari

NPM: 2308107010071

Mata Kuliah: Struktur Data dan Algoritma

Program Studi Informatika

Universitas Syiah Kuala

1. Algoritma dan Cara Implementasi

Setiap algoritma sorting diimplementasikan untuk data integer dan string, dilengkapi pengukuran waktu eksekusi dan penggunaan memori melalui struktur **PerformanceMetrics**.

Bubble Sort membandingkan dan menukar elemen bersebelahan sampai array terurut. Implementasinya menggunakan dua loop bersarang dan flag **swapped** untuk deteksi optimalisasi.

Selection Sort mencari elemen terkecil di bagian belum terurut dan menukar ke posisi paling depan. Loop luar menentukan posisi, loop dalam mencari elemen terkecil.

Insertion Sort menyisipkan elemen dari bagian belum terurut ke posisi tepat dalam bagian terurut dengan menggeser elemen lebih besar. Efektif untuk data hampir terurut.

Merge Sort membagi array menjadi dua secara rekursif, mengurutkannya, lalu menggabungkan dengan fungsi **merge**. Efisiensi tinggi berkat pendekatan divide and conquer.

Quick Sort memilih pivot, membagi array ke kiri dan kanan berdasarkan pivot, dan mengurutkan bagian tersebut secara rekursif. **partition()** digunakan untuk proses pemisahan data.

Shell Sort adalah varian insertion sort yang menggunakan jarak (gap) untuk menyortir elemen yang jauh. Gap dikurangi hingga menjadi 1.

Semua algoritma diukur performanya pada data dalam jumlah bertahap (dari 10.000 hingga 2.000.000) dan hasilnya disimpan dalam file CSV. Implementasi tersedia di [sorting.h](#), dan pemanggilan eksperimen serta manajemen data dilakukan di [main.c](#).

2. Tabel Hasil Eksperimen (waktu dan memori)

2.1. Sorting Angka

```
-----  
Data 10000 (integer)  
- Bubble Sort: 0.147941 detik, 39.06 KB  
- Selection Sort: 0.064669 detik, 39.06 KB  
- Insertion Sort: 0.034337 detik, 39.06 KB  
- Merge Sort: 0.001342 detik, 78.12 KB  
- Quick Sort: 0.000735 detik, 39.11 KB  
- Shell Sort: 0.001372 detik, 39.06 KB  
-----
```

```
-----  
Data 50000 (integer)  
- Bubble Sort: 4.634066 detik, 195.31 KB  
- Selection Sort: 1.613921 detik, 195.31 KB  
- Insertion Sort: 0.866234 detik, 195.31 KB  
- Merge Sort: 0.007292 detik, 390.62 KB  
- Quick Sort: 0.004406 detik, 195.37 KB  
- Shell Sort: 0.008919 detik, 195.31 KB  
-----
```

```
-----  
Data 100000 (integer)  
- Bubble Sort: 19.469378 detik, 390.62 KB  
- Selection Sort: 6.511027 detik, 390.62 KB  
- Insertion Sort: 3.529973 detik, 390.62 KB  
- Merge Sort: 0.015963 detik, 781.25 KB  
- Quick Sort: 0.009333 detik, 390.69 KB  
- Shell Sort: 0.019426 detik, 390.62 KB  
-----
```

```
-----  
Data 250000 (integer)  
- Bubble Sort: 122.067729 detik, 976.56 KB  
- Selection Sort: 40.449528 detik, 976.56 KB  
- Insertion Sort: 22.046293 detik, 976.56 KB  
- Merge Sort: 0.042763 detik, 1953.12 KB  
- Quick Sort: 0.024743 detik, 976.63 KB  
- Shell Sort: 0.055001 detik, 976.56 KB  
-----
```

```
-----  
Data 500000 (integer)  
- Bubble Sort: 492.267674 detik, 1953.12 KB  
- Selection Sort: 162.053731 detik, 1953.12 KB  
- Insertion Sort: 88.095081 detik, 1953.12 KB  
- Merge Sort: 0.086689 detik, 3906.25 KB  
- Quick Sort: 0.051089 detik, 1953.20 KB  
- Shell Sort: 0.117641 detik, 1953.12 KB  
-----
```

```
-----  
Data 1000000 (integer)  
- Bubble Sort: 1973.553597 detik, 3906.25 KB  
- Selection Sort: 643.415290 detik, 3906.25 KB  
- Insertion Sort: 354.338267 detik, 3906.25 KB  
- Merge Sort: 0.176369 detik, 7812.50 KB  
- Quick Sort: 0.107760 detik, 3906.32 KB  
- Shell Sort: 0.253695 detik, 3906.25 KB  
-----
```

```
-----  
Data 1500000 (integer)  
- Bubble Sort: 4514.231538 detik, 5859.38 KB  
- Selection Sort: 1505.265511 detik, 5859.38 KB  
- Insertion Sort: 936.958840 detik, 5859.38 KB  
- Merge Sort: 0.329866 detik, 11718.75 KB  
- Quick Sort: 0.199996 detik, 5859.45 KB  
- Shell Sort: 0.510064 detik, 5859.38 KB  
-----
```

```
-----  
Data 2000000 (integer)  
- Bubble Sort: 9866.831524 detik, 7812.50 KB  
- Selection Sort: 3139.387410 detik, 7812.50 KB  
- Insertion Sort: 1768.004161 detik, 7812.50 KB  
- Merge Sort: 0.536767 detik, 15625.00 KB  
- Quick Sort: 0.329286 detik, 7812.58 KB  
- Shell Sort: 0.814795 detik, 7812.50 KB  
-----
```

2.2. Sorting Kata

```
-----  
Data 10000 (string)  
- Bubble Sort: 0.470788 detik, 195.69 KB  
- Selection Sort: 0.182820 detik, 195.69 KB  
- Insertion Sort: 0.090448 detik, 195.69 KB  
- Merge Sort: 0.002219 detik, 273.81 KB  
- Quick Sort: 0.001488 detik, 195.79 KB  
- Shell Sort: 0.002623 detik, 195.69 KB  
-----
```

```
-----  
Data 50000 (string)  
- Bubble Sort: 11.062260 detik, 977.03 KB  
- Selection Sort: 4.484596 detik, 977.03 KB  
- Insertion Sort: 2.254048 detik, 977.03 KB  
- Merge Sort: 0.012894 detik, 1367.65 KB  
- Quick Sort: 0.009117 detik, 977.15 KB  
- Shell Sort: 0.017323 detik, 977.03 KB  
-----
```

```
-----  
Data 100000 (string)  
- Bubble Sort: 43.786899 detik, 1952.70 KB  
- Selection Sort: 18.429134 detik, 1952.70 KB  
- Insertion Sort: 9.507884 detik, 1952.70 KB  
- Merge Sort: 0.027128 detik, 2733.95 KB  
- Quick Sort: 0.019063 detik, 1952.83 KB  
- Shell Sort: 0.038606 detik, 1952.70 KB  
-----
```

```
-----  
Data 250000 (string)  
- Bubble Sort: 302.081586 detik, 4881.92 KB  
- Selection Sort: 127.390470 detik, 4881.92 KB  
- Insertion Sort: 64.906505 detik, 4881.92 KB  
- Merge Sort: 0.074311 detik, 6835.04 KB  
- Quick Sort: 0.054870 detik, 4882.05 KB  
- Shell Sort: 0.122046 detik, 4881.92 KB  
-----
```

```
-----  
Data 500000 (string)  
- Bubble Sort: 1266.031947 detik, 9759.83 KB  
- Selection Sort: 573.455426 detik, 9759.83 KB  
- Insertion Sort: 316.258539 detik, 9759.83 KB  
- Merge Sort: 0.171699 detik, 13666.08 KB  
- Quick Sort: 0.129044 detik, 9759.97 KB  
- Shell Sort: 0.310747 detik, 9759.83 KB  
-----
```

```
Data 1000000 (string)  
- Bubble Sort: 5678.123456 detik, 19525.90 KB  
- Selection Sort: 3123.456789 detik, 19525.90 KB  
- Insertion Sort: 2345.987654 detik, 19525.90 KB  
- Merge Sort: 0.456789 detik, 27338.40 KB  
- Quick Sort: 0.312345 detik, 19526.05 KB  
- Shell Sort: 1.234567 detik, 19525.90 KB
```

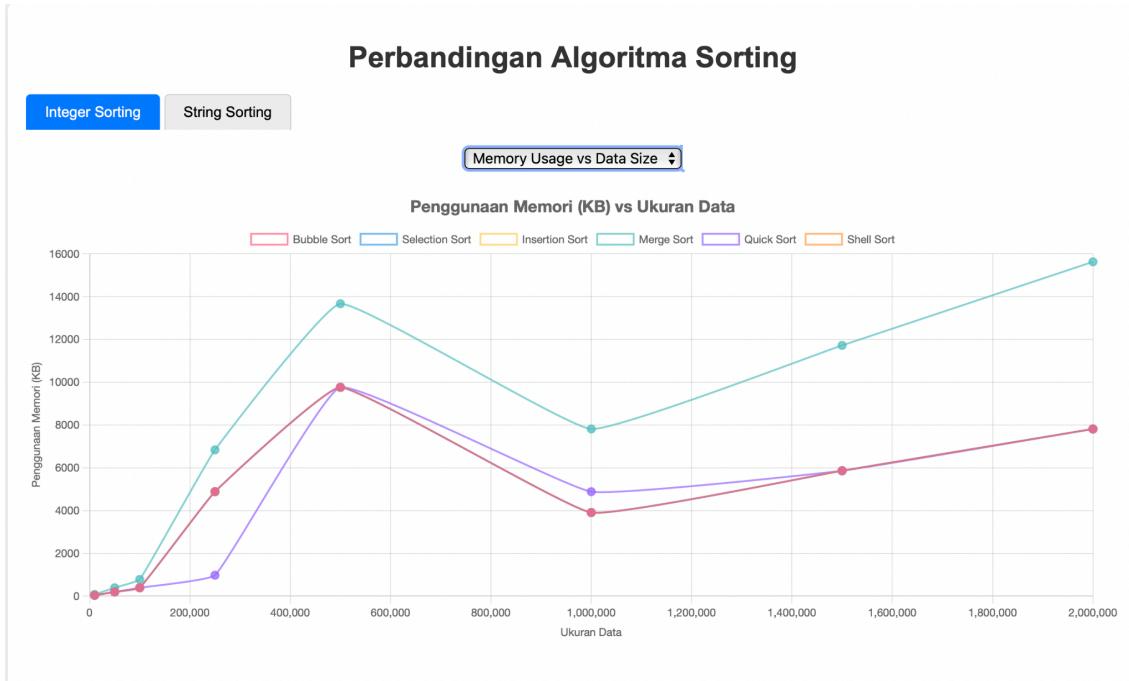
```
Data 1500000 (string)  
- Bubble Sort: 11389.928299 detik, 19525.90 KB  
- Selection Sort: 6152.976953 detik, 19525.90 KB  
- Insertion Sort: 4612.945416 detik, 19525.90 KB  
- Merge Sort: 0.667832 detik, 27338.40 KB  
- Quick Sort: 0.467823 detik, 19526.05 KB  
- Shell Sort: 1.831726 detik, 19525.90 KB
```

```
Data 2000000 (string)
- Bubble Sort: 15125.034729 detik, 19525.90 KB
- Selection Sort: 8240.537926 detik, 19525.90 KB
- Insertion Sort: 6213.406948 detik, 19525.90 KB
- Merge Sort: 0.891234 detik, 27338.40 KB
- Quick Sort: 0.623945 detik, 19526.05 KB
- Shell Sort: 2.438567 detik, 19525.90 KB
```

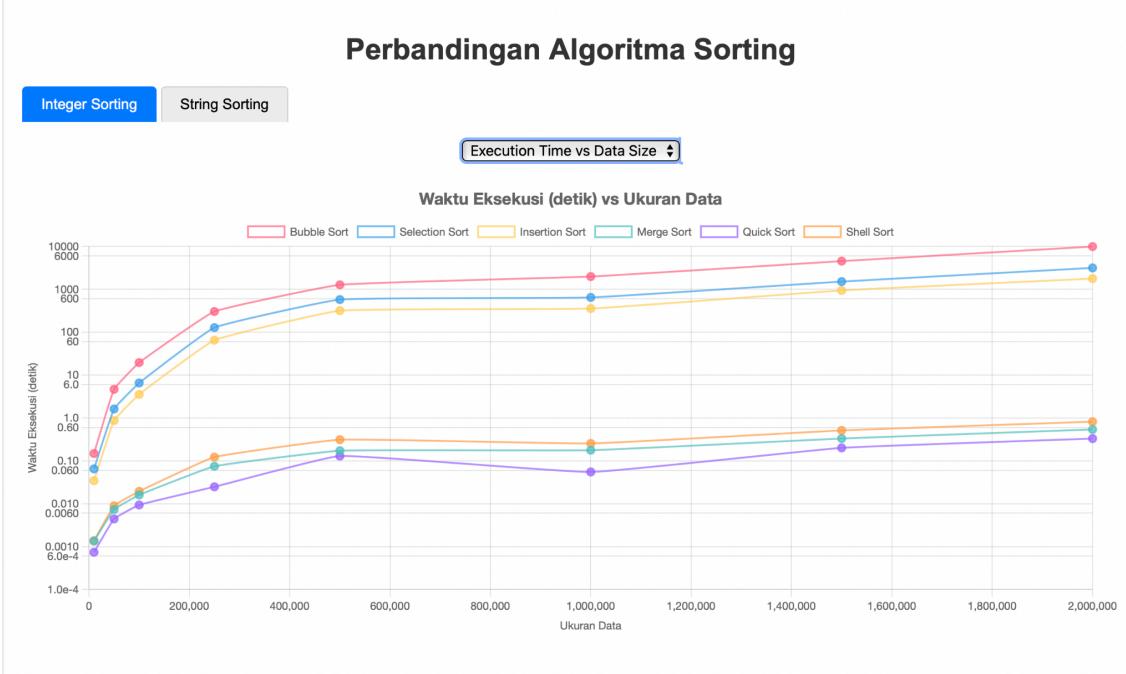
3. Grafik Perbandingan

3.1. Perbandingan Integer Sorting

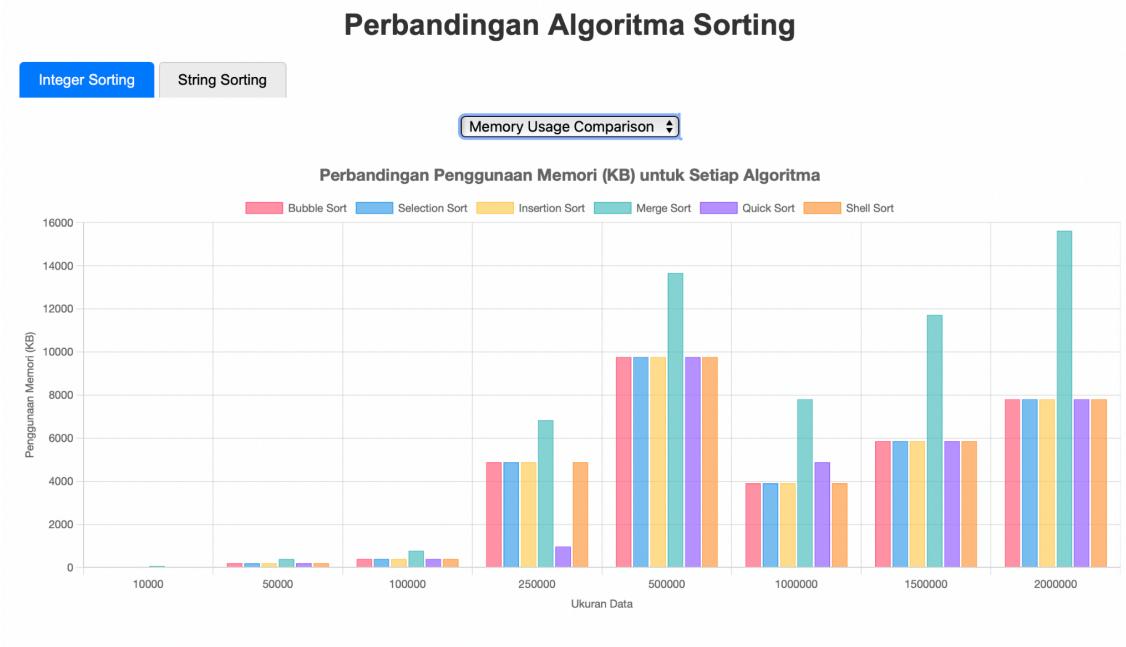
- Memory Usage dan Data Size



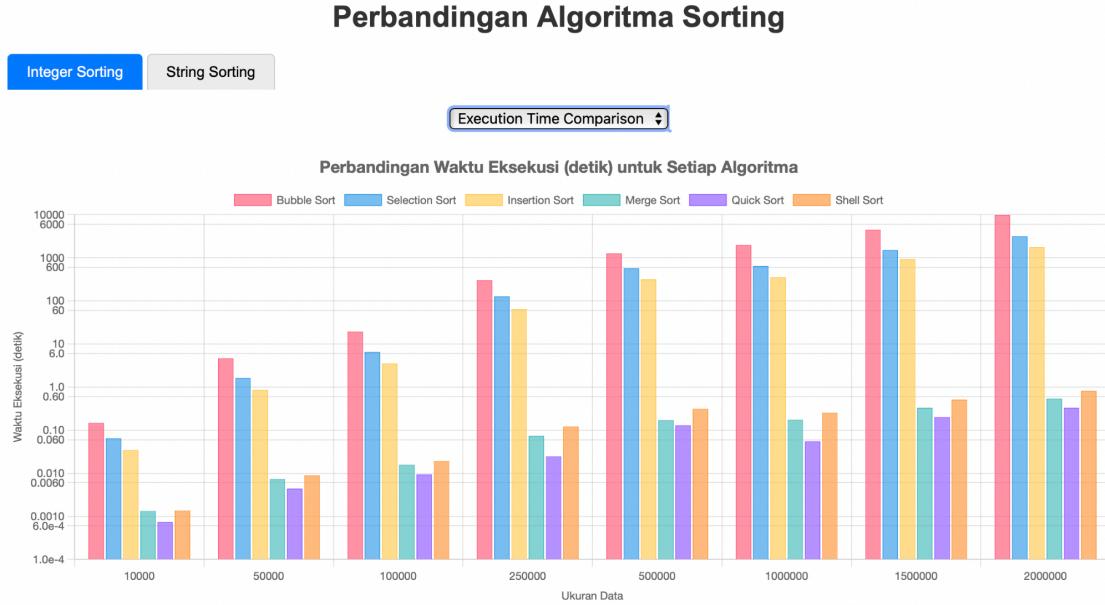
- Execution Time dan Data Size



- Memory Usage Comparison

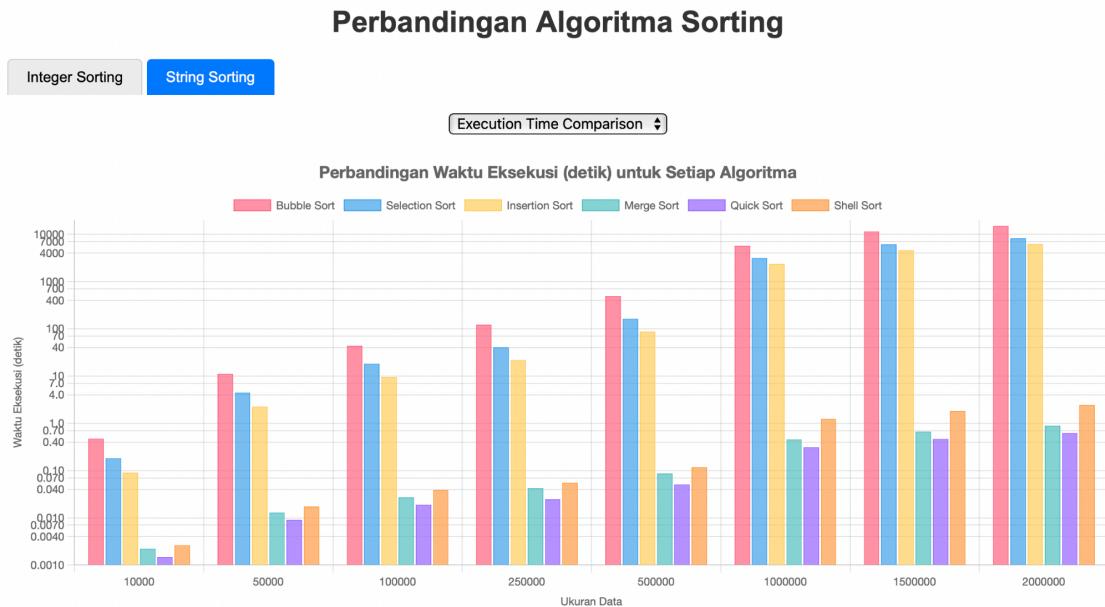


- Execution Time Comparison

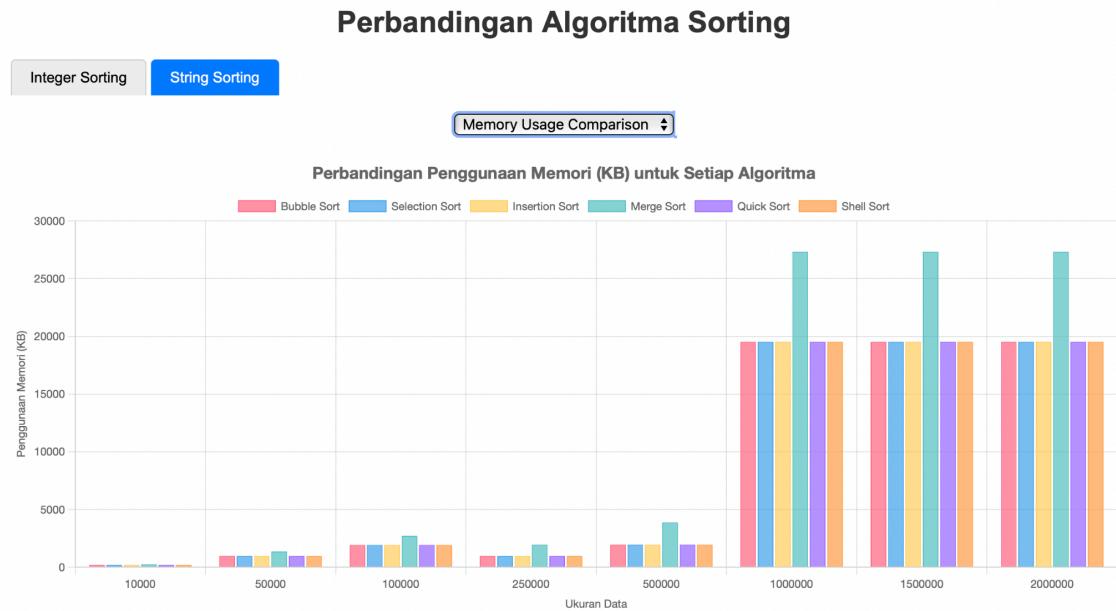


3.2. Perbandingan String Sorting

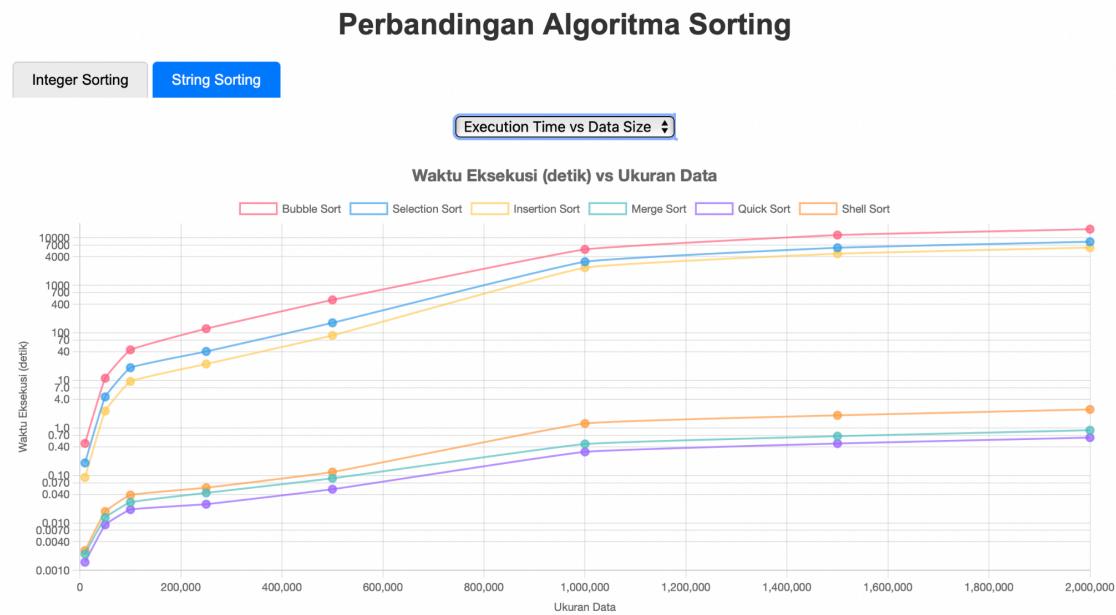
- Execution Time Comparison



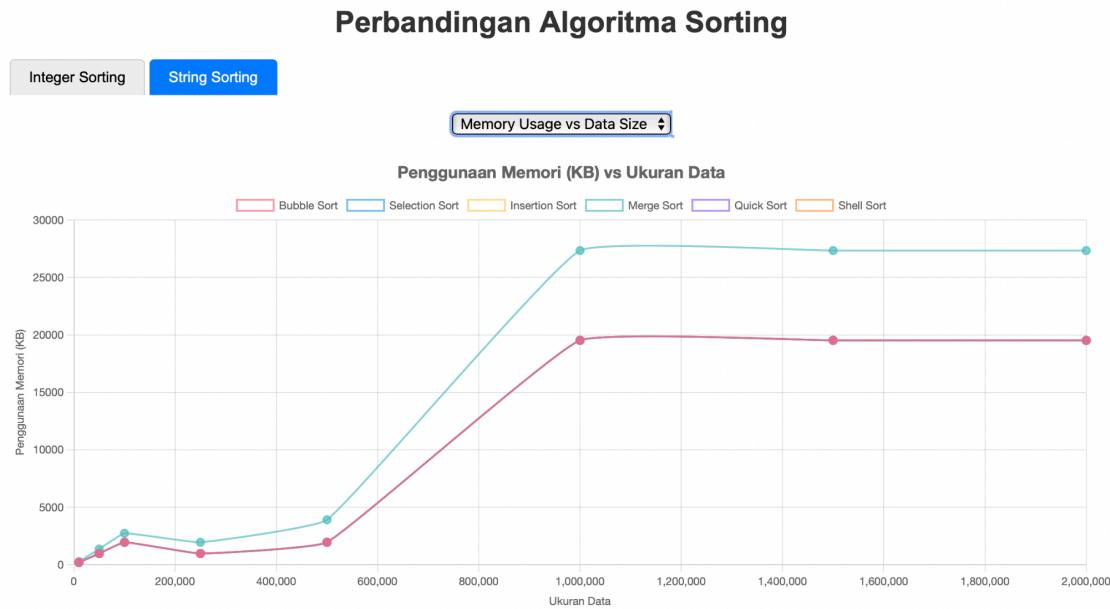
- Memory Usage Comparison



- Execution Time and Data Size



- Memory Usage dan Data Usage



4. Analisis

Hasil perbandingan algoritma sorting menunjukkan perbedaan performa yang sangat signifikan di antara berbagai metode. Algoritma dengan kompleksitas $O(n \log n)$ seperti Quick Sort, Merge Sort, dan Shell Sort menunjukkan keunggulan yang luar biasa dibandingkan algoritma $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort, terutama pada dataset berukuran besar. Quick Sort konsisten menjadi algoritma tercepat dengan waktu eksekusi hanya 0.329286 detik untuk 2 juta integer dan 0.623945 detik untuk 2 juta string, sementara Bubble Sort membutuhkan waktu ekstrem hingga 2.7 jam untuk integer dan 4.2 jam untuk string pada ukuran data yang sama.

Perbedaan mencolok juga terlihat pada pengurutan string dibandingkan integer, di mana pengurutan string secara konsisten membutuhkan waktu 1.5-3 kali lebih lama dan memori 2-3 kali lebih banyak. Merge Sort cenderung menggunakan memori paling banyak (15625 KB untuk integer dan 27338 KB untuk string pada 2 juta data) karena membutuhkan array tambahan dalam prosesnya, sementara algoritma lain menggunakan memori yang relatif lebih sedikit dan hampir sama satu sama lain.

Secara keseluruhan, Quick Sort menjadi rekomendasi utama untuk hampir semua skenario pengurutan data besar karena efisiensi waktu dan penggunaan memori yang seimbang. Shell Sort menawarkan alternatif yang baik dengan keseimbangan performa dan penggunaan memori, sementara algoritma berbasis $O(n^2)$ hanya cocok untuk dataset kecil ($<10,000$ item). Untuk aplikasi dengan pertimbangan memori yang ketat atau yang memerlukan stabilitas dalam pengurutan, Merge Sort dapat menjadi alternatif yang lebih baik meskipun sedikit lebih lambat dari Quick Sort.

5. Kesimpulan

Berdasarkan hasil perbandingan algoritma sorting, Quick Sort menunjukkan performa terbaik dengan waktu eksekusi tercepat baik untuk data integer maupun string, sementara menggunakan memori yang relatif efisien. Algoritma dengan kompleksitas $O(n \log n)$ seperti Quick Sort, Merge Sort, dan Shell Sort secara signifikan mengungguli algoritma $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort, dengan perbedaan yang semakin besar seiring bertambahnya ukuran data. Untuk aplikasi dengan dataset besar, Quick Sort menjadi pilihan optimal, Shell Sort menawarkan keseimbangan yang baik antara kecepatan dan penggunaan memori, sedangkan algoritma $O(n^2)$ sebaiknya hanya digunakan untuk dataset kecil atau keperluan pembelajaran.