TUGAS 4
STRUKTUR DATA DAN ALGORITMA

disusun untuk memenuhi
tugas 4 struktur data dan algoritma

Oleh:

Muadz Fauzi
2308107010042

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

A. Deskripsi Algoritma dan Cara Implementasi

a. Quick Sort

Algoritma:

Quick Sort adalah algoritma pengurutan berbasis metode Divide and Conquer. Konsep dasarnya adalah memilih satu elemen sebagai **pivot**, lalu membagi data menjadi dua bagian: data yang lebih kecil dari pivot dan data yang lebih besar dari pivot. Proses ini dilakukan secara rekursif hingga seluruh elemen terurut.

Tahapan:

- Pilih pivot (biasanya elemen terakhir).
- Partisi array ke dua bagian: kecil dan besar dari pivot.
- Tempatkan pivot di posisi yang benar.
- Rekursif panggil Quick Sort untuk bagian kiri dan kanan.

Implementasi kode program:

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Fungsi partition() akan menempatkan pivot di posisi akhirnya, lalu quickSort() dipanggil ke bagian kiri dan kanan array. Untuk string menggunakan perbandingan strcmp() dan strcpy().

b. Shell Sort

Algoritma:

Shell Sort adalah pengembangan dari Insertion Sort yang menggunakan jarak antar elemen (**gap**) untuk membandingkan elemen yang berjauhan terlebih dahulu. Nilai gap secara bertahap dikurangi hingga menjadi 1, sehingga algoritma berubah menjadi Insertion Sort di tahap akhir.

Tahapan:

- Tentukan nilai gap (misal n/2).
- Lakukan pengurutan elemen yang memiliki jarak gap.
- Kurangi gap menjadi setengahnya, ulangi proses.
- Jika gap = 1, lakukan Insertion Sort seperti biasa.

Implementasi kode program:

```
void shellSort(int arr[], int n) {
    for (int gap = n/2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j-gap] > temp; j -= gap)
                arr[j] = arr[j-gap];
            arr[j] = temp;
        }
    }
}
```

Untuk tipe data string, perbandingan menggunakan strcmp(), dan pertukaran dilakukan dengan strcpy(). Penggunaan gap mempercepat proses sorting pada data besar.

c. Merge Sort

Algoritma:

Merge Sort juga menggunakan metode Divide and Conquer. Data dibagi dua secara rekursif hingga bagian terkecil, kemudian digabung kembali dalam keadaan terurut menggunakan fungsi merge().

Tahapan:

- Bagi array menjadi dua bagian.
- Urutkan masing-masing bagian secara rekursif.
- Gabungkan dua bagian menggunakan fungsi merge().

Implementasi kode program:

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

Fungsi merge() bertugas menggabungkan dua array kecil menjadi satu array besar terurut. Untuk string, proses perbandingan menggunakan strcmp().

d. Insertion Sort

Algoritma:

Insertion Sort menyusun array satu per satu, dimulai dari elemen kedua. Setiap elemen dibandingkan mundur dengan elemen sebelumnya dan dipindahkan ke posisi yang sesuai.

Tahapan:

- Ambil elemen sebagai key.
- Bandingkan dengan elemen sebelumnya.
- Geser elemen ke kanan jika lebih besar.
- Tempatkan key di posisi yang tepat.

Implementasi kode program:

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}
```

Pada string, perbandingan menggunakan strcmp() dan penyalinan nilai string dilakukan dengan strcpy().

e. Selection Sort

Algoritma:

Selection Sort mencari elemen terkecil di array yang belum terurut dan memindahkannya ke posisi paling depan. Proses diulang hingga semua elemen terurut.

Tahapan:

- Temukan elemen terkecil.
- Tukar dengan elemen pertama array yang belum terurut.
- Ulangi untuk elemen berikutnya.

Implementasi kode program:

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        int minIdx = i;
        for (int j = i+1; j < n; j++)
            if (arr[j] < arr[minIdx])
                minIdx = j;
        int temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
    }
}
```

Untuk string, perbandingan nilai menggunakan strcmp(), dan penukaran menggunakan strcpy().

f. Bubble Sort

Algoritma:

Bubble Sort adalah algoritma sederhana yang membandingkan dua elemen bersebelahan dan menukar jika urutannya salah. Proses diulang hingga seluruh array terurut.

Tahapan:

- Bandingkan dua elemen bersebelahan.
- Tukar jika salah urut.
- Ulangi proses hingga tidak ada lagi pertukaran.

Implementasi kode program:

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

Versi string menggunakan strcmp() untuk membandingkan string, dan strcpy() untuk menukar posisi string.

B. Tabel Hasil Eksperimen (waktu dan memori)

a. Hasil Pengujian 10000 Angka dan Kata

Angka		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,001	78,1
Shell Sort	0,001	78,1
Merge Sort	0,001	78,1
Insertion Sort	0,027	78,1
Selection Sort	0,042	78,1
Bubble Sort	0,101	78,1
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,006	1953,1
Shell Sort	0,008	1953,1
Merge Sort	0,009	1953,1
Insertion Sort	0,405	1953,1
Selection Sort	0,130	1953,1
Bubble Sort	1,195	1953,1

b. Hasil Pengujian 50000 Angka dan Kata

Angka

Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,005	390,6
Shell Sort	0,008	390,6
Merge Sort	0,005	390,6
Insertion Sort	0,630	390,6
Selection Sort	0,996	390,6
Bubble Sort	4,432	390,6
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,035	9765,6
Shell Sort	0,059	9765,6
Merge Sort	0,052	9765,6
Insertion Sort	10,815	9765,6
Selection Sort	3,530	9765,6
Bubble Sort	31,210	9765,6

c. Hasil Pengujian 100000 Angka dan Kata

Angka		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,009	781,3
Shell Sort	0,017	781,3
Merge Sort	0,011	781,3
Insertion Sort	2,606	781,3
Selection Sort	4,088	781,3
Bubble Sort	18,368	781,3
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,069	19531,3
Shell Sort	0,126	19531,3
Merge Sort	0,106	19531,3
Insertion Sort	45,425	19531,3
Selection Sort	15,919	19531,3
Bubble Sort	132,492	19531,3

d. Hasil Pengujian 250000 Angka dan Kata

Angka		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,023	1953,1
Shell Sort	0,047	1953,1
Merge Sort	0,032	1953,1
Insertion Sort	16,010	1953,1
Selection Sort	25,119	1953,1
Bubble Sort	120,061	1953,1
Huruf		
Algoritma	Waktu (s)	Memori (KB)

Quick Sort	0,186	48828,1
Shell Sort	0,340	48828,1
Merge Sort	0,263	48828,1
Insertion Sort	371,186	48828,1
Selection Sort	153,212	48828,1
Bubble Sort	892,804	48828,1

e. Hasil Pengujian 500000 Angka dan Kata

Angka		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,049	3906,3
Shell Sort	0,102	3906,3
Merge Sort	0,066	3906,3
Insertion Sort	65,220	3906,3
Selection Sort	101,884	3906,3
Buble Sort	488,257	3906,3
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,391	97656,3
Shell Sort	0,826	97656,3
Merge Sort	0,546	97656,3
Insertion Sort	1296,903	97656,3
Selection Sort	743,363	97656,3
Bubble Sort	3444,651	97656,3

f. Hasil Pengujian 1000000 Angka dan Kata

Angka		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,135	7812,5
Shell Sort	0,230	7812,5
Merge Sort	0,199	7812,5
Insertion Sort	744,911	7812,5
Selection Sort	543,269	7812,5
Bubble Sort	3445,428	7812,5
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,839	19531
Shell Sort	1,729	19531
Merge Sort	1,173	19531
Insertion Sort	46,437	19531
Selection Sort	17,439	19531
Bubble Sort	126,815	19531

g. Hasil Pengujian 1500000 Angka dan Kata

Angka

Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,193	11718,8
Shell Sort	0,373	11718,8
Merge Sort	0,193	11718,8
Insertion Sort	1438,305	11718,8
Selection Sort	974,993	11718,8
Bubble Sort	7545,899	11718,8
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	1,366	292968
Shell Sort	3,019	292968
Merge Sort	1,812	292968
Insertion Sort	20448,201	292968
Selection Sort	13689,376	292968
Bubble Sort	31001,9	292968

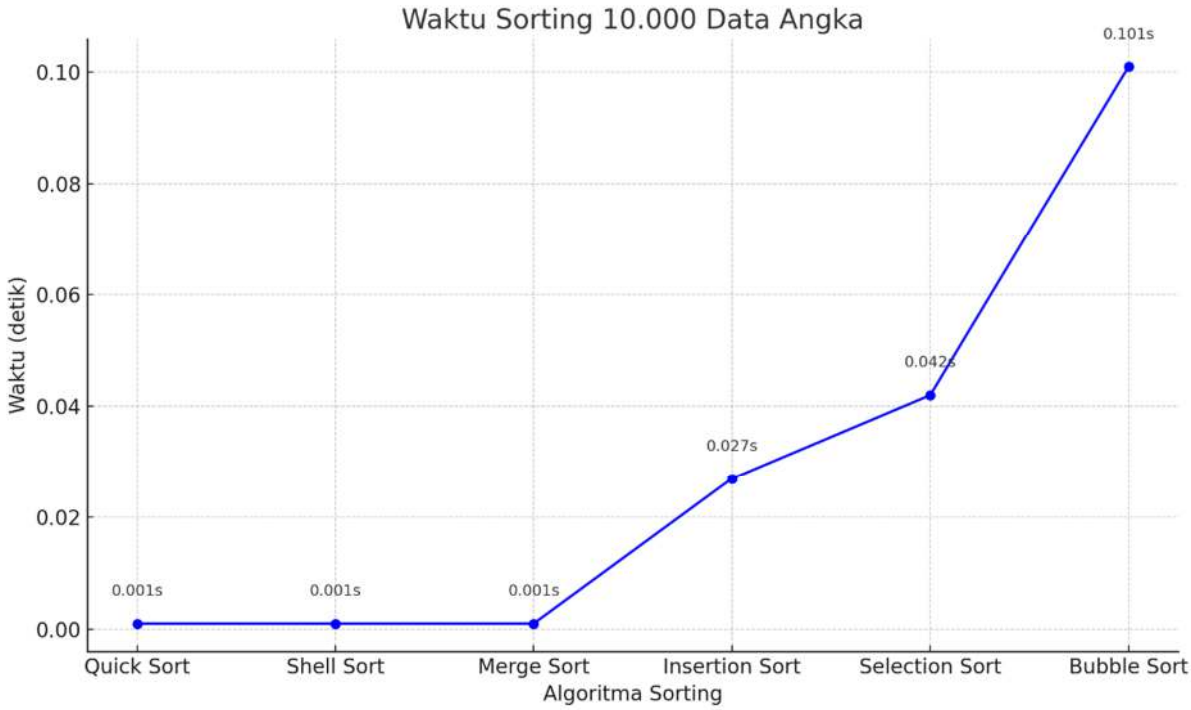
h. Hasil Pengujian 2000000 Angka dan Kata

Angka		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	0,288	15625
Shell Sort	0,493	15625
Merge Sort	0,415	15625
Insertion Sort	1137,444	15625
Selection Sort	1602,993	15625
Bubble Sort	7871,247	15625
Huruf		
Algoritma	Waktu (s)	Memori (KB)
Quick Sort	2,398	390625
Shell Sort	1,775	390625
Merge Sort	2,398	390625
Insertion Sort	36342,07	390625
Selection Sort	24328,26	390625
Bubble Sort	27564,56	390625

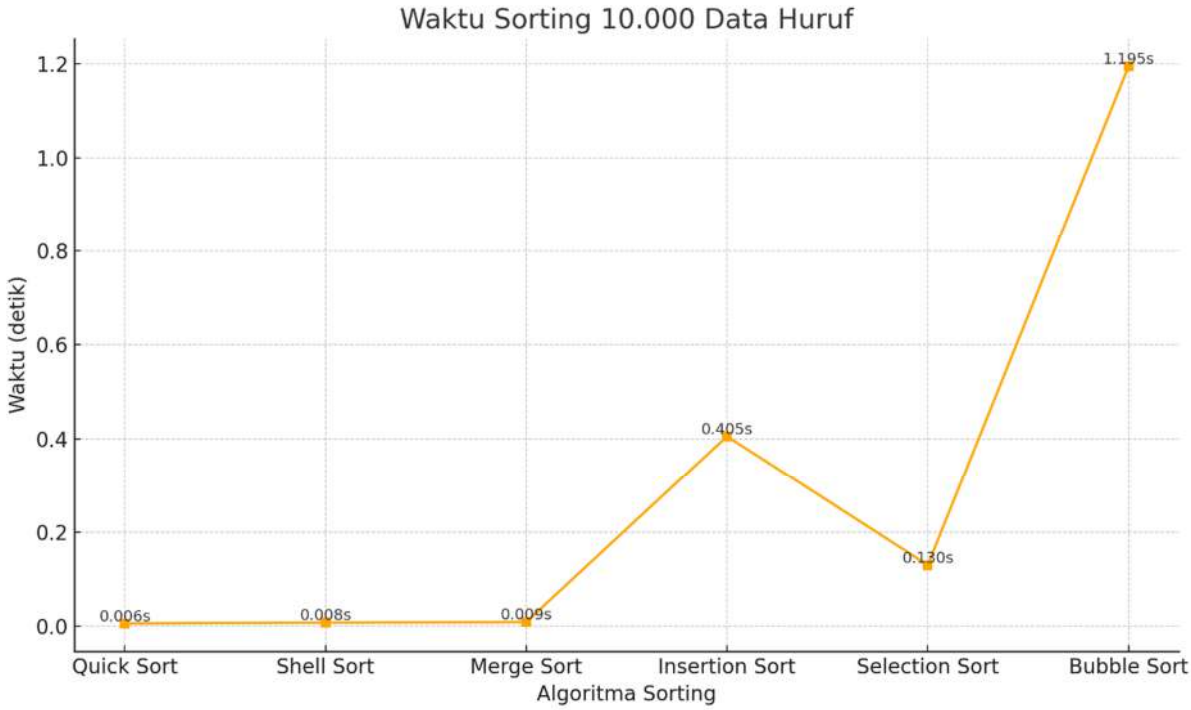
C. Grafik Perbandingan Waktu dan Memori

a. Grafik Pengujian data 10000

- Angka

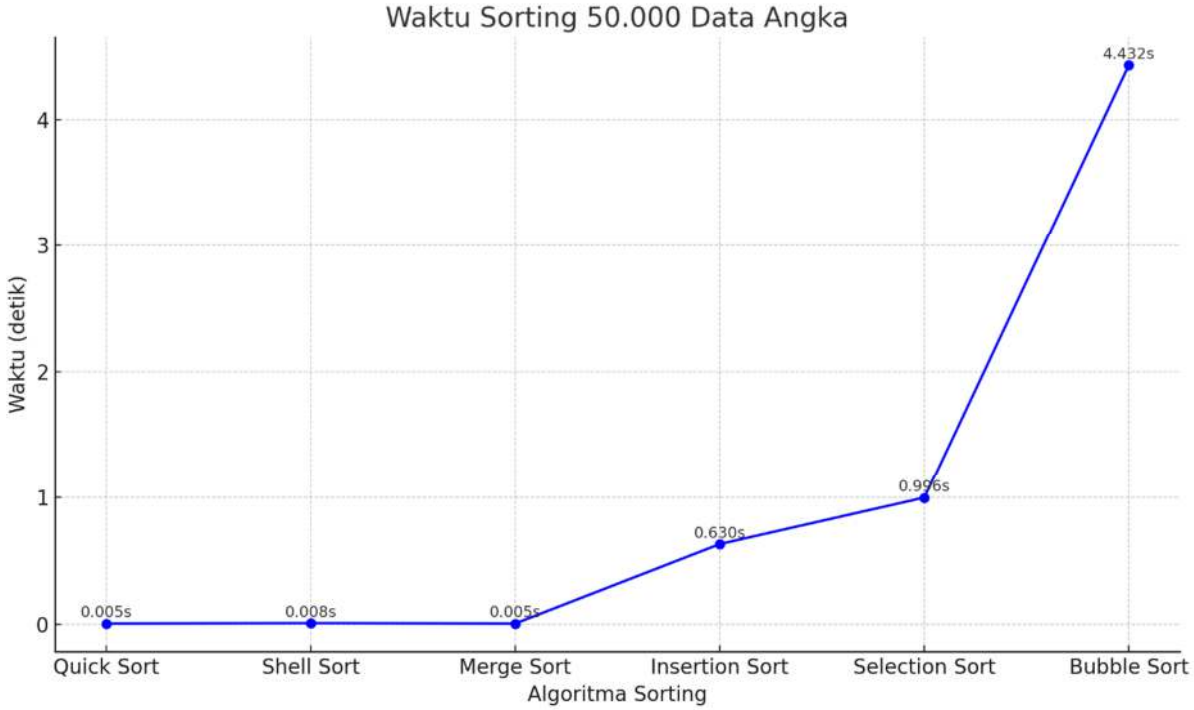


- Kata

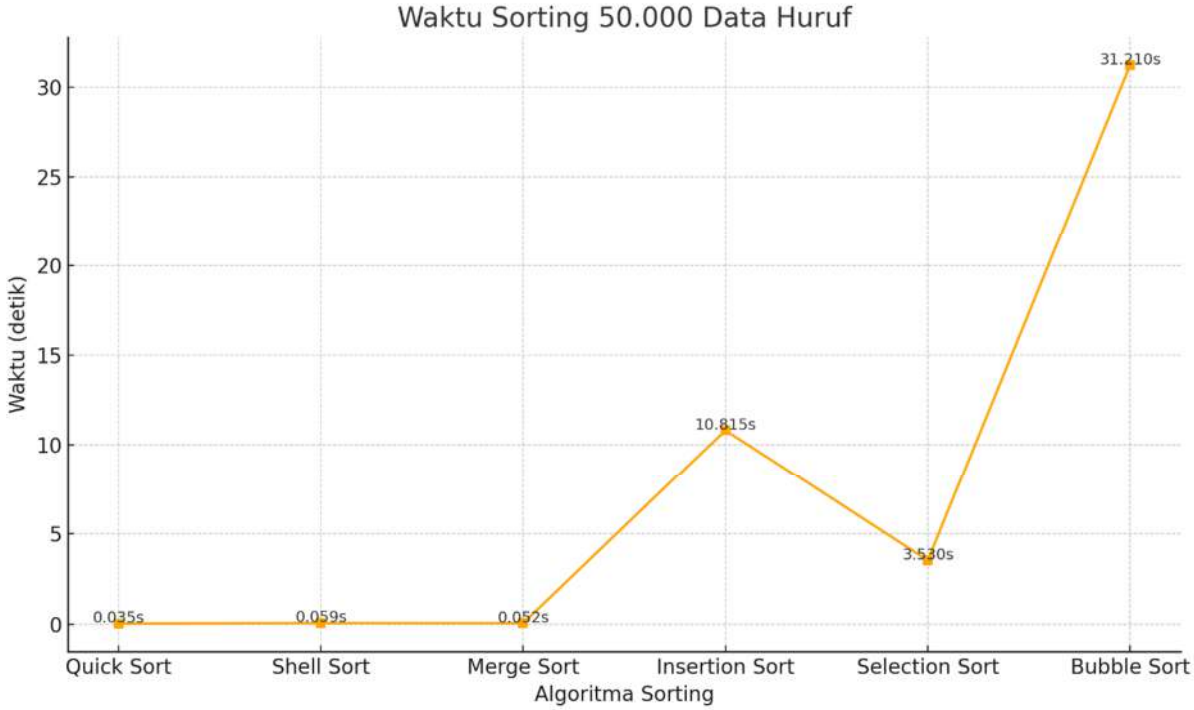


b. Grafik Pengujian data 50000

- Angka

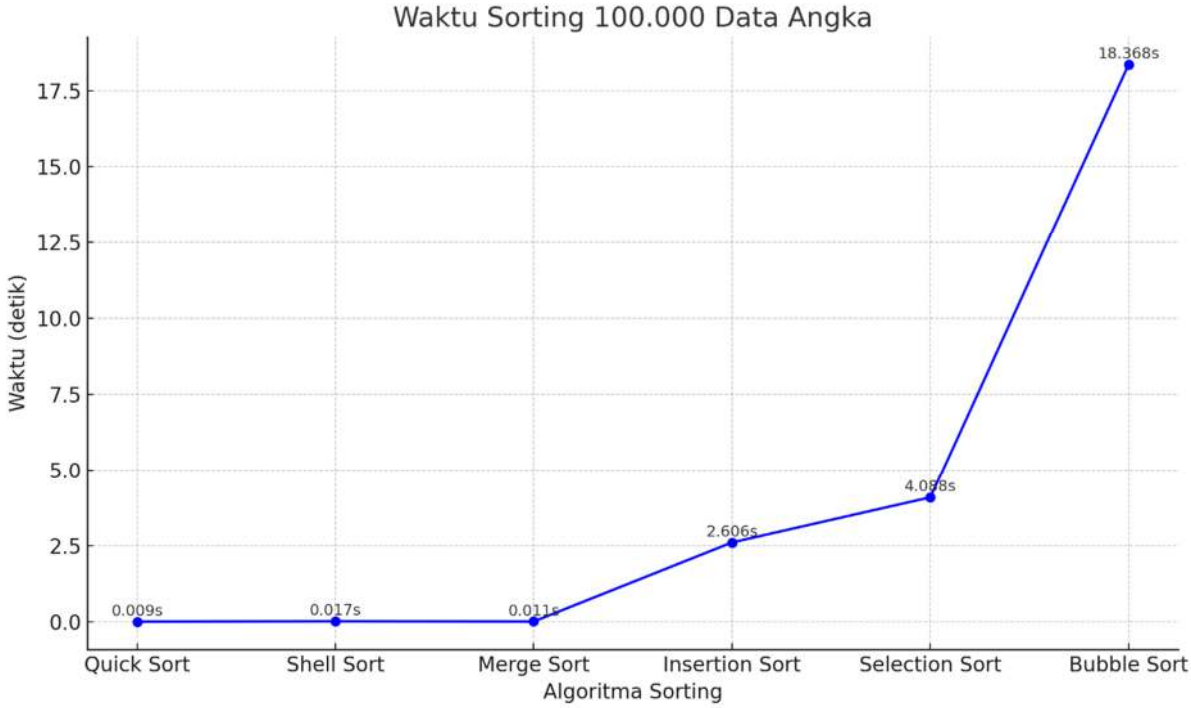


- Kata

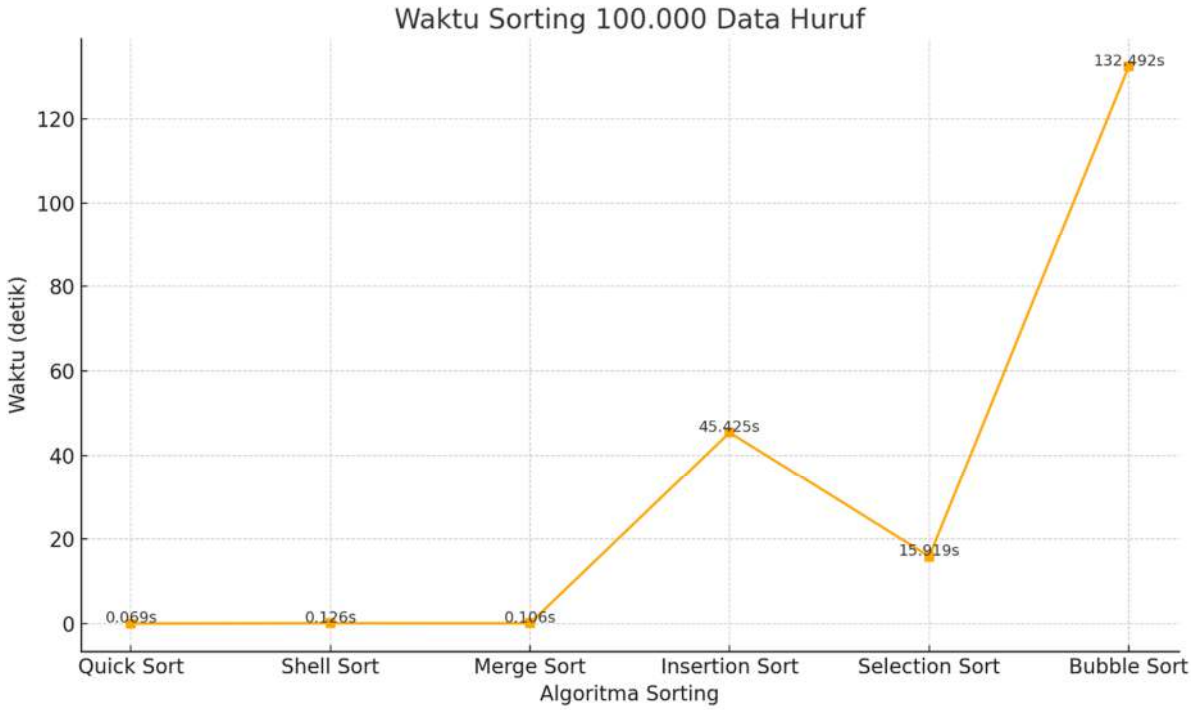


c. Grafik Pengujian data 100000

- Angka

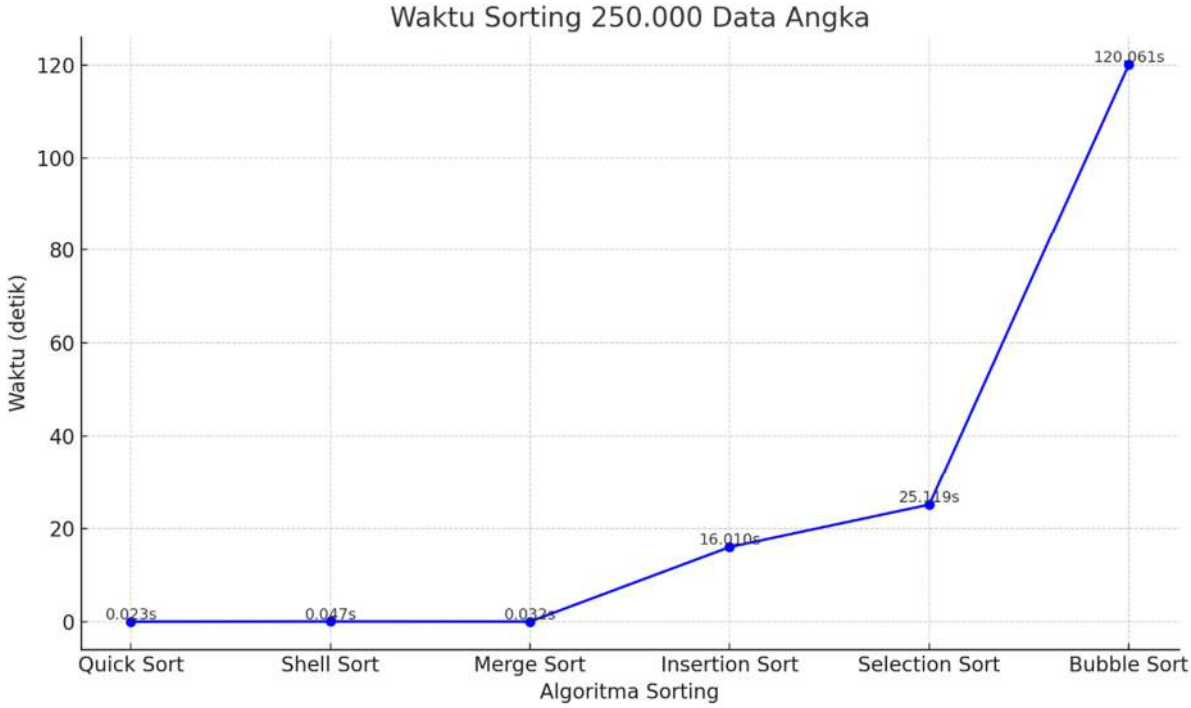


- Kata

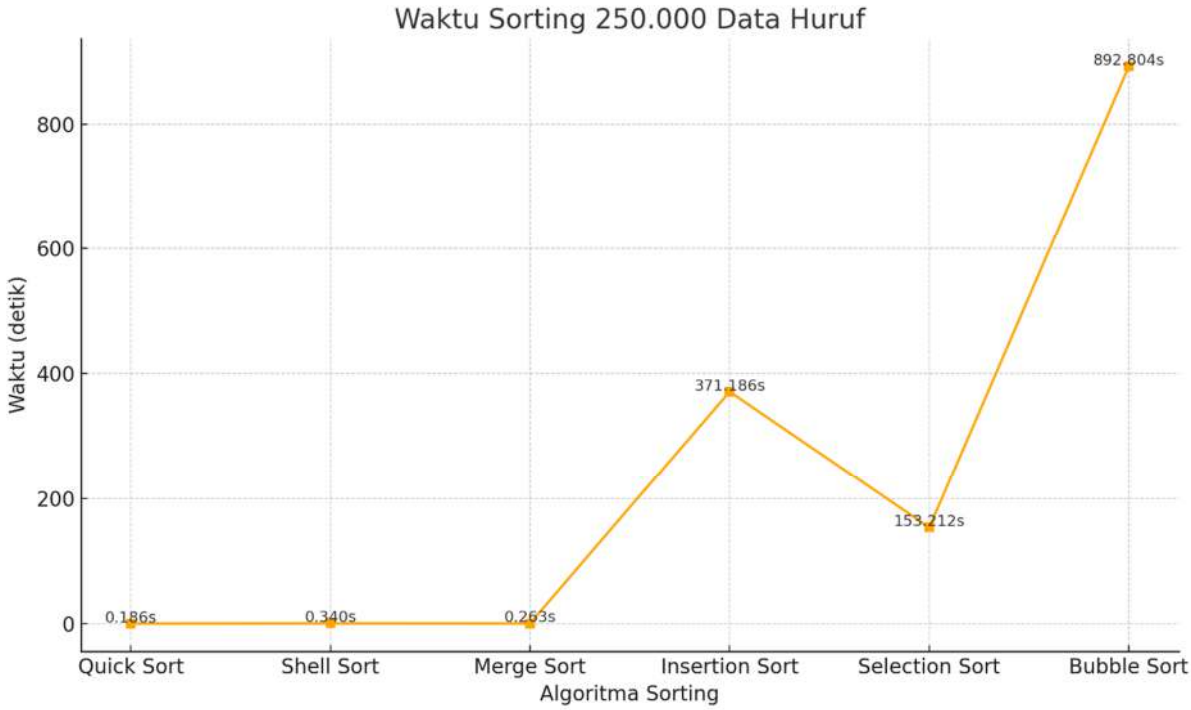


d. Grafik Pengujian data 250000

- Angka

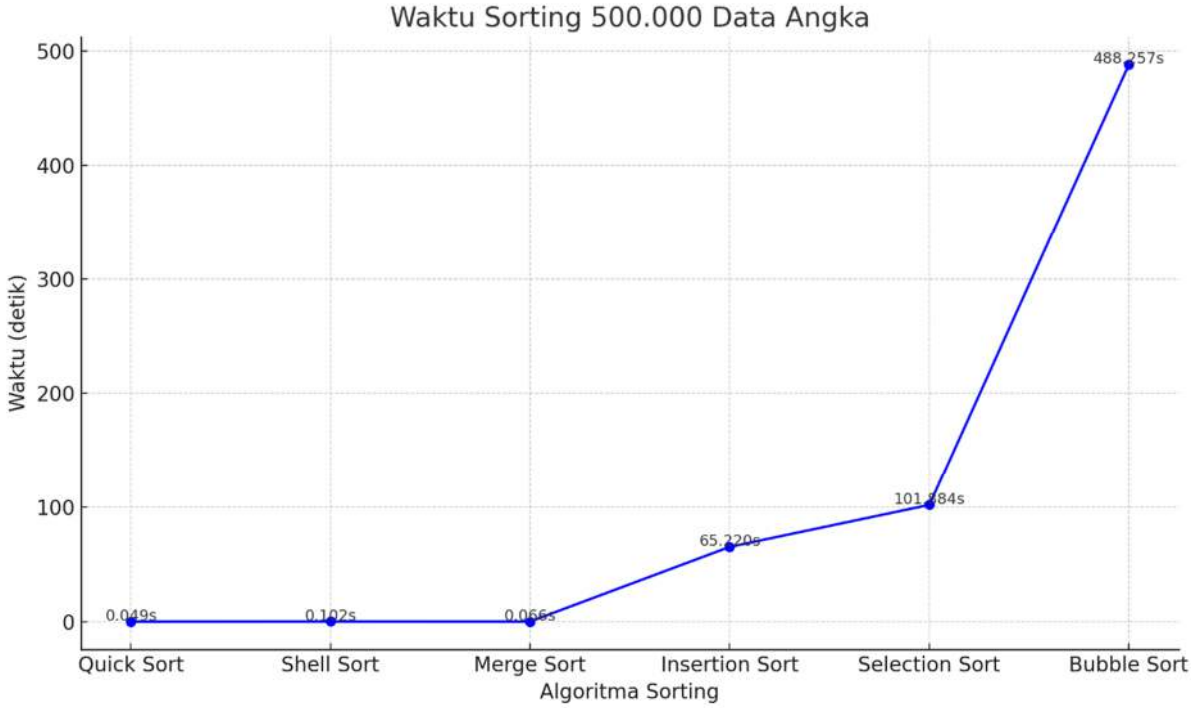


- Kata

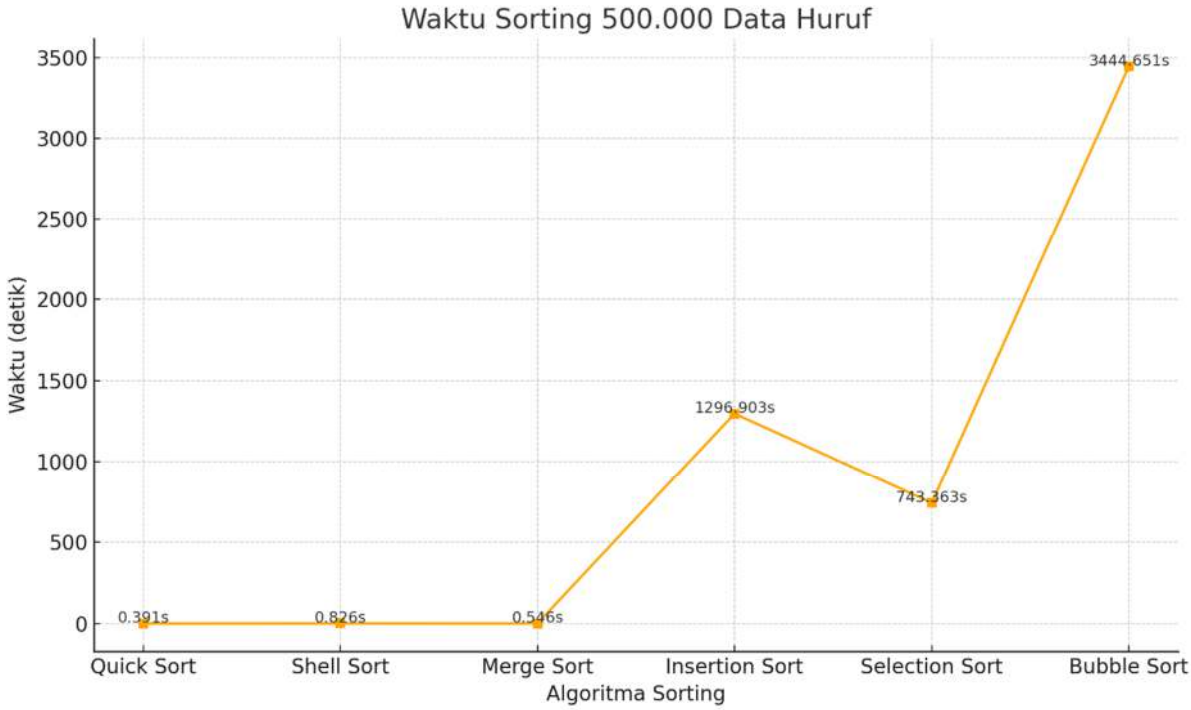


e. Grafik Pengujian data 500000

- Angka

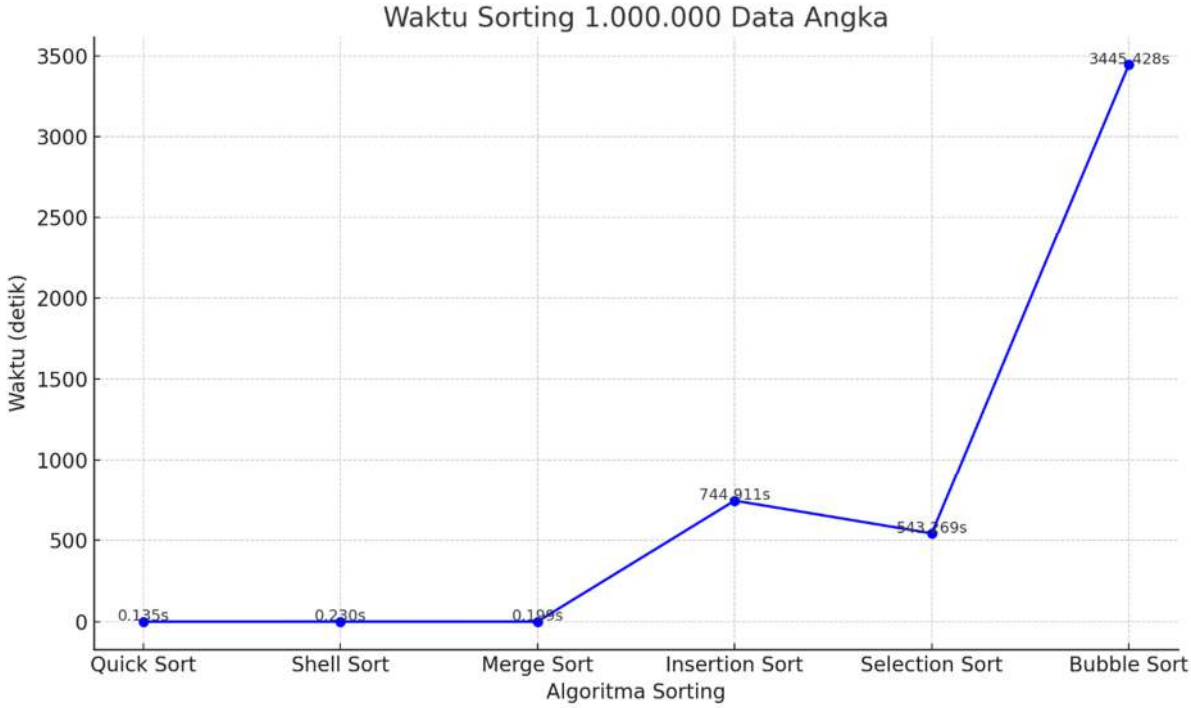


- Kata

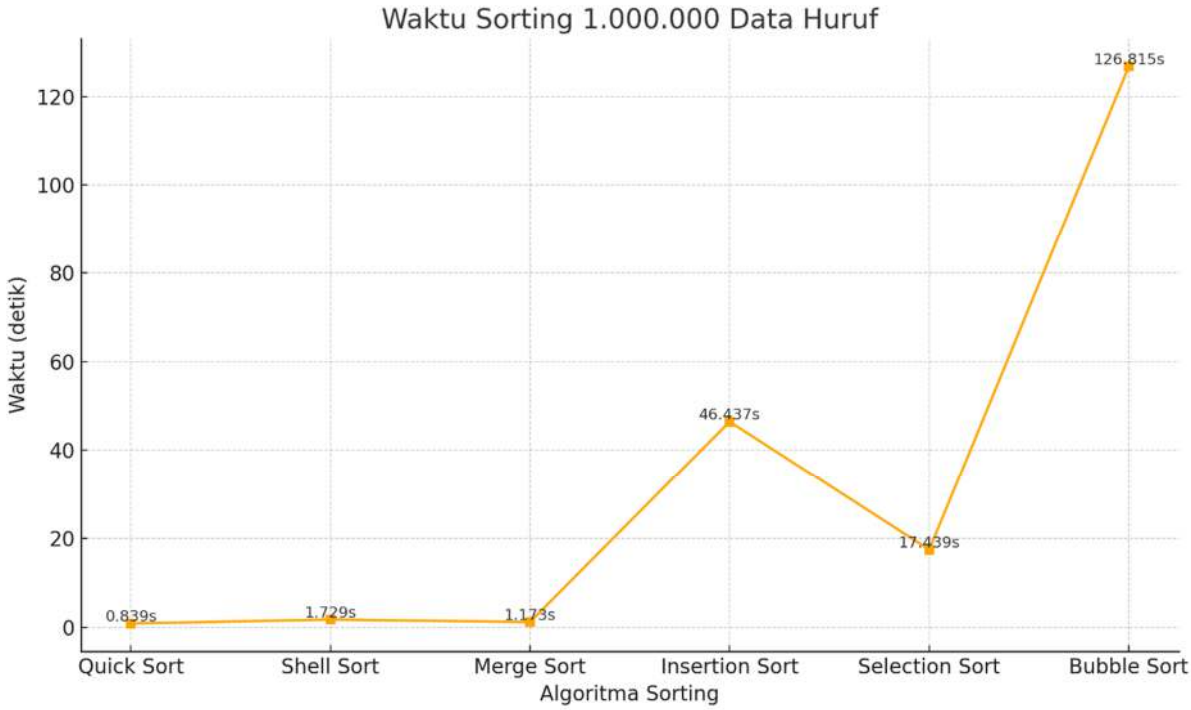


f. Grafik Pengujian data 1000000

- Angka

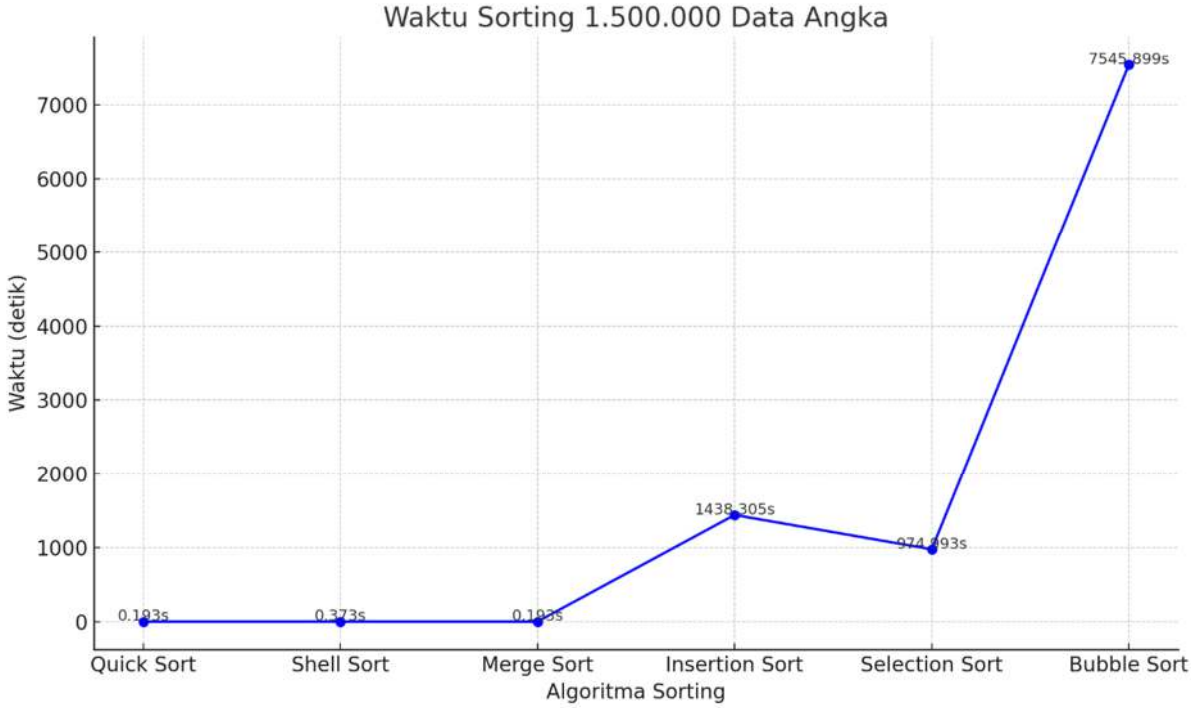


- Kata

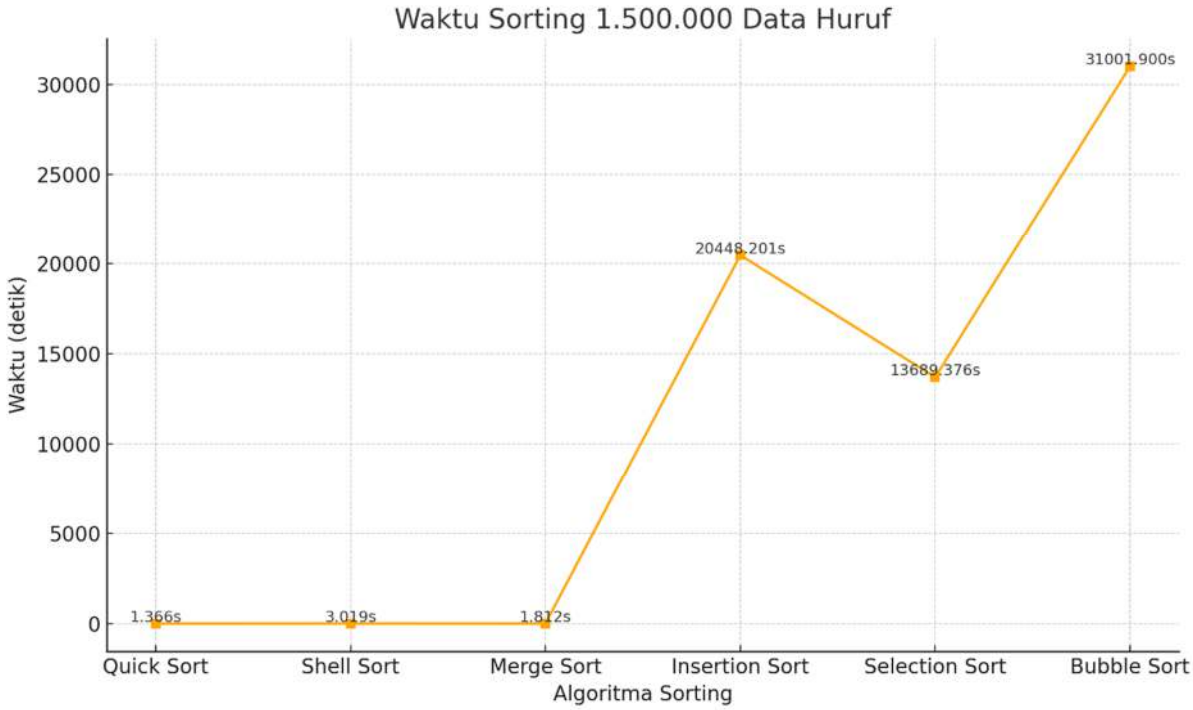


g. Grafik Pengujian data 1500000

- Angka

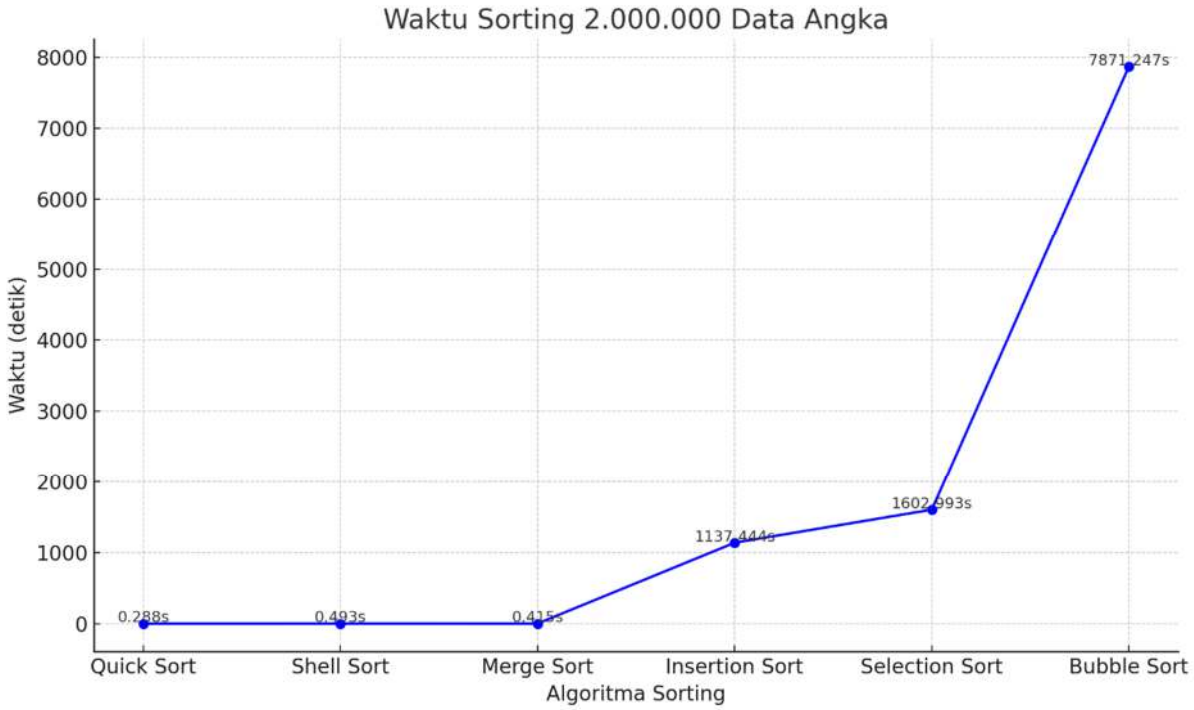


- Kata

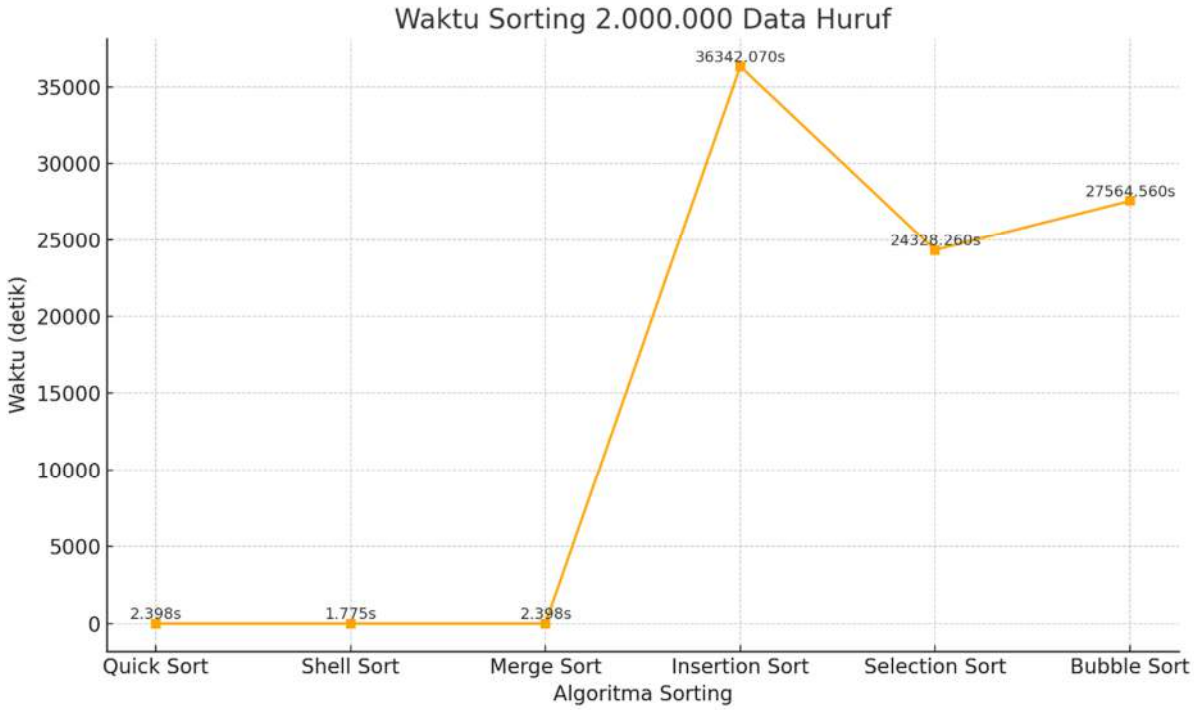


h. Grafik Pengujian data 2000000

- Angka



- Kata



D. Analisa dan Kesimpulan

a. Analisa

Berdasarkan hasil pengujian sorting pada data angka dan huruf dengan jumlah data bervariasi mulai dari 10.000 hingga 2.000.000, didapatkan beberapa temuan penting yang ditunjukkan pada grafik-grafik sebelumnya:

1. Algoritma Quick Sort, Merge Sort, dan Shell Sort menunjukkan performa yang sangat baik untuk data dalam skala besar, baik angka maupun huruf.

- Quick Sort konsisten menjadi algoritma tercepat untuk data dalam jumlah besar karena kompleksitas rata-ratanya O(n log n).
- Shell Sort juga menunjukkan performa stabil dan efisien, lebih unggul dari Insertion, Selection, dan Bubble Sort.
- Merge Sort performanya baik, meskipun butuh lebih banyak memori karena proses pembagian dan penggabungan data.

2. Algoritma Bubble Sort, Selection Sort, dan Insertion Sort sangat tidak efisien untuk jumlah data besar.
   - Pada 500.000 data ke atas, waktu eksekusi ketiga algoritma ini meningkat drastis, terutama Bubble Sort yang mencapai puluhan ribu detik saat mengolah 1,5 juta hingga 2 juta huruf.
   - Hal ini sejalan dengan kompleksitas algoritma $O(n^2)$ yang menyebabkan waktu eksekusi membengkak seiring bertambahnya jumlah data.
3. Perbandingan sorting angka vs huruf
   - Sorting huruf membutuhkan waktu lebih lama dibanding angka untuk algoritma yang sama dan jumlah data yang sama.
   - Ini karena operasi perbandingan string (pakai `strcmp()`) lebih mahal dibanding operasi integer biasa, yang cukup sekali banding nilai.
4. Data string jauh lebih berat untuk algoritma $O(n^2)$
   - Pada jumlah data besar, sorting string seperti Bubble, Selection, dan Insertion benar-benar tidak praktis untuk dipakai karena waktu proses bisa mencapai puluhan ribu detik..

b. Kesimpulan

- Quick Sort menjadi algoritma paling efisien dan konsisten untuk mengurutkan data dalam skala besar, baik angka maupun huruf.
- Merge Sort dan Shell Sort juga sangat cocok untuk pengurutan data besar dengan hasil waktu eksekusi yang stabil dan mendekati Quick Sort.
- Bubble Sort, Selection Sort, dan Insertion Sort sebaiknya dihindari untuk jumlah data besar karena performanya sangat buruk, terutama pada sorting string yang membutuhkan waktu eksekusi berkali-kali lipat lebih lama dibanding sorting angka.
- Operasi sorting string lebih berat dibanding angka karena proses perbandingan string lebih kompleks.
- Performa sorting sangat dipengaruhi oleh kompleksitas algoritma. Algoritma dengan kompleksitas O(n log n) jauh lebih efisien dibanding $O(n^2)$ saat jumlah data besar.