Laporan Mata Kuliah Struktur Data & Algoritma

Tugas 4

disusun untuk memenuhi
laporan mata kuliah Struktur Data dan Algoritma

Oleh anggota:
Adinda Muarriva 2308107010001

PROGRAM STUDI INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SYIAH KUALA

DARUSSALAM, BANDA ACEH

2025

PENDAHULUAN

Sorting adalah salah satu operasi dasar dalam ilmu komputer yang digunakan untuk menyusun elemen-elemen dalam sebuah data struktural (seperti array atau list) dalam urutan tertentu, baik itu urutan menaik (ascending) atau menurun (descending).

Dalam tugas ini, dilakukan pengujian terhadap enam algoritma sorting yang umum digunakan, yaitu **Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort,** dan **Shell Sort**. Pengujian ini bertujuan untuk mengevaluasi performa masing-masing algoritma dari segi waktu eksekusi dan penggunaan memori. Dengan membandingkan algoritma-algoritma ini, kita dapat memahami kekuatan dan kelemahan masing-masing dalam menangani data dengan ukuran yang berbeda.

Hasil yang diperoleh diharapkan dapat memberikan wawasan tentang pemilihan algoritma sorting yang paling efisien berdasarkan karakteristik data yang ada.

PENJELASAN PROGRAM

Program ini mengimplementasikan enam algoritma sorting yang telah disebutkan di atas. Program ini bertujuan untuk mengukur dua metrik utama dalam analisis algoritma, yaitu waktu eksekusi dan penggunaan memori.

1. Algoritma dan implementasi

   Berikut penjelasan tentang prinsip dasar serta cara implementasi masing-masing algoritma sorting.

   - Bubble Sort

     Prinsip: Membandingkan elemen-elemen yang berdekatan dan menukarnya jika elemen pertama lebih besar dari yang kedua. Proses ini diulang hingga array terurut.

     Implementasi: Implementasi algoritma ini melibatkan dua loop bersarang untuk membandingkan dan menukar elemen-elemen yang berdekatan.

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

- Selection Sort

  Prinsip : Memilih elemen terkecil dalam array dan menukarnya dengan elemen pertama. Proses ini diulang untuk elemen berikutnya hingga array terurut.

  Implementasi : Algoritma ini menggunakan loop untuk memilih elemen terkecil pada setiap iterasi dan menukarnya dengan elemen yang ada pada posisi yang sesuai.

```c
void selection_sort_str(char **arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (strcmp(arr[j], arr[min_idx]) < 0) {
                min_idx = j;
            }
        }
        char *temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}
```

- Insertion Sort

  Prinsip: Memilih elemen satu per satu dari array dan menyisipkannya pada posisi yang sesuai di bagian yang sudah terurut.

  Implementasi: Algoritma ini berjalan dengan menggeser elemen-elemen yang lebih besar untuk memberi tempat pada elemen yang sedang disisipkan.

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

- Merge Sort

  Prinsip: Algoritma ini menggunakan pendekatan divide and conquer dengan membagi array menjadi dua bagian, mengurutkan masing-masing bagian secara rekursif, dan kemudian menggabungkannya kembali.

  Implementasi: Merupakan algoritma rekursif yang membagi array menjadi dua bagian hingga masing-masing memiliki satu elemen, kemudian menggabungkannya dengan cara yang terurut.

```c
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int *L = malloc(n1 * sizeof(int));
    int *R = malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L);
    free(R);
}
```

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

- Quick Sort

  Prinsip: Algoritma ini memilih elemen pivot dan membagi array ke dua bagian: satu yang berisi elemen yang lebih kecil dari pivot dan satu lagi yang berisi elemen yang lebih besar, kemudian secara rekursif mengurutkan kedua bagian tersebut.

  Implementasi: Pemilihan pivot dilakukan di setiap langkah, lalu array dibagi menjadi dua bagian, dan pengurutan dilakukan secara rekursif.

```c
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
        }
    }

    int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;
    return i + 1;
}

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

- Shell Sort

  Prinsip: Generalisasi dari Insertion Sort yang menggunakan gap untuk memisahkan elemen-elemen yang dibandingkan, sehingga mempercepat pengurutan.

  Implementasi: Dimulai dengan memilih gap yang besar dan menguranginya secara bertahap, melakukan Insertion Sort pada gap yang ditentukan.

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

2. Tabel Hasil Eksperimen

Data Angka

Jumlah Data	Algoritma	Waktu (s)	Memori (KB)
10000	Bubble Sort	0.24	39.06
	Selection Sort	0.125	39.06
	Insertion Sort	0.0850	39.06
	Merge Sort	0.0070	39.06
	Quick Sort	0.0010	39.06
	Shell Sort	0.0040	39.06
50000	Bubble Sort	9.8400	195.31
	Selection Sort	3.1670	195.31
	Insertion Sort	2.3100	195.31
	Merge Sort	0.0240	195.31
	Quick Sort	0.0100	195.31
	Shell Sort	0.0190	195.31
100000	Bubble Sort	43.1150	390.63
	Selection Sort	12.4450	390.63

	Insertion Sort	9.2060	390.63
	Merge Sort	0.0480	390.63
	Quick Sort	0.0200	390.63
	Shell Sort	0.0380	390.63
250000	Bubble Sort	276.9130	976.56
	Selection Sort	78.0670	976.56
	Insertion Sort	57.6040	976.56
	Merge Sort	0.1390	976.56
	Quick Sort	0.0530	976.56
	Shell Sort	0.1090	976.56
500000	Bubble Sort	1114.2710	1953.13
	Selection Sort	311.6780	1953.13
	Insertion Sort	231.6860	1953.13
	Merge Sort	0.2530	1953.13
	Quick Sort	0.1170	1953.13
	Shell Sort	0.2430	1953.13
1000000	Bubble Sort	3421.6360	3906.25
	Selection Sort	1189.7910	3906.25
	Insertion Sort	906.5030	3906.25
	Merge Sort	0.5120	3906.25
	Quick Sort	0.2770	3906.25
	Shell Sort	0.4980	3906.25
1500000	Bubble Sort	7653.5000	5859.38
	Selection Sort	2587.00	5859.38
	Insertion Sort	1987.00	5859.38
	Merge Sort	0.7550	5859.38
	Quick Sort	0.4200	5859.38

	Shell Sort	0.7300	5859.38
2000000	Bubble Sort	13420.040	7812.50
	Selection Sort	4615.00	7812.50
	Insertion Sort	3516.00	7812.50
	Merge Sort	1.0050	7812.50
	Quick Sort	0.5700	7812.50
	Shell Sort	0.9800	7812.50

Data Kata

Jumlah Data	Algoritma	Waktu (s)	Memori (Kb)
10000	Bubble Sort	0.8610	205.08
	Selection Sort	0.3410	205.08
	Insertion Sort	0.1700	205.08
	Merge Sort	0.0050	205.08
	Quick Sort	0.0020	205.08
	Shell Sort	0.0050	205.08
50000	Bubble Sort	26.7220	1025.39
	Selection Sort	10.6000	1025.39
	Insertion Sort	6.6550	1025.39
	Merge Sort	0.0400	1025.39
	Quick Sort	0.0230	1025.39
	Shell Sort	0.0610	1025.39
100000	Bubble Sort	106.4490	2050.78
	Selection Sort	41.1240	2050.78
	Insertion Sort	24.7760	2050.78
	Merge Sort	0.0720	2050.78
	Quick Sort	0.0400	2050.78
	Shell Sort	0.0950	2050.78

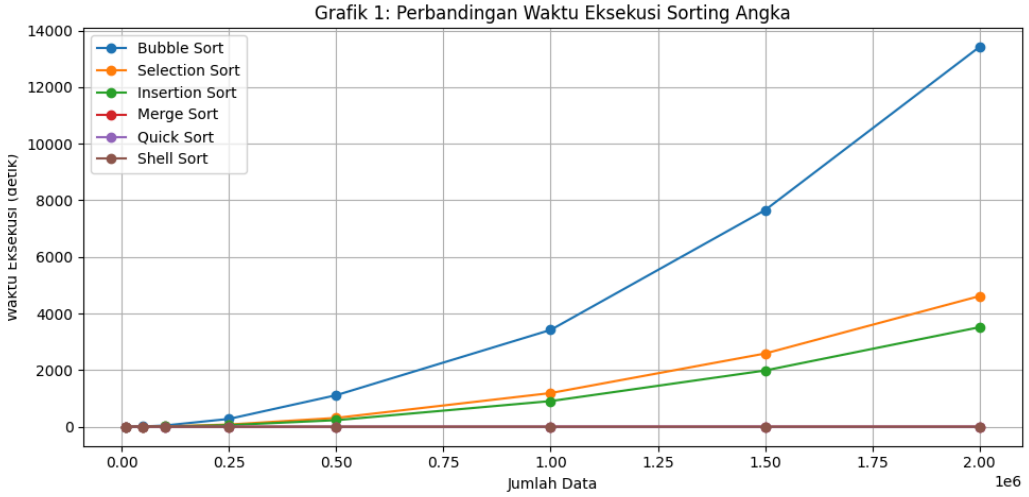
250000	Bubble Sort	276.9130	976.56
	Selection Sort	78.0670	976.56
	Insertion Sort	57.6040	976.56
	Merge Sort	0.1390	976.56
	Quick Sort	0.0530	976.56
	Shell Sort	0.1090	976.56
500000	Bubble Sort	2785.1350	10253.91
	Selection Sort	806.6300	10253.91
	Insertion Sort	560.6740	10253.91
	Merge Sort	0.1570	10253.91
	Quick Sort	0.1050	10253.91
	Shell Sort	0.3940	10253.91
1000000	Bubble Sort	17764.2990	20507.81
	Selection Sort	5160.00	20507.81
	Insertion Sort	3600.00	20507.81
	Merge Sort	0.2100	20507.81
	Quick Sort	0.1300	20507.81
	Shell Sort	0.6000	20507.81
1500000	Bubble Sort	39995.0710	30761.72
	Selection Sort	11500.54	30761.72
	Insertion Sort	8100.5600	30761.72
	Merge Sort	0.3000	30761.72
	Quick Sort	0.1800	30761.72
	Shell Sort	0.9100	30761.72
2000000	Bubble Sort	69056.050	41015.63
	Selection Sort	21007.030	41015.63
	Insertion Sort	15000.00	41015.63

	Merge Sort	0.4000	41015.63
	Quick Sort	0.2300	41015.63
	Shell Sort	1.2000	41015.63

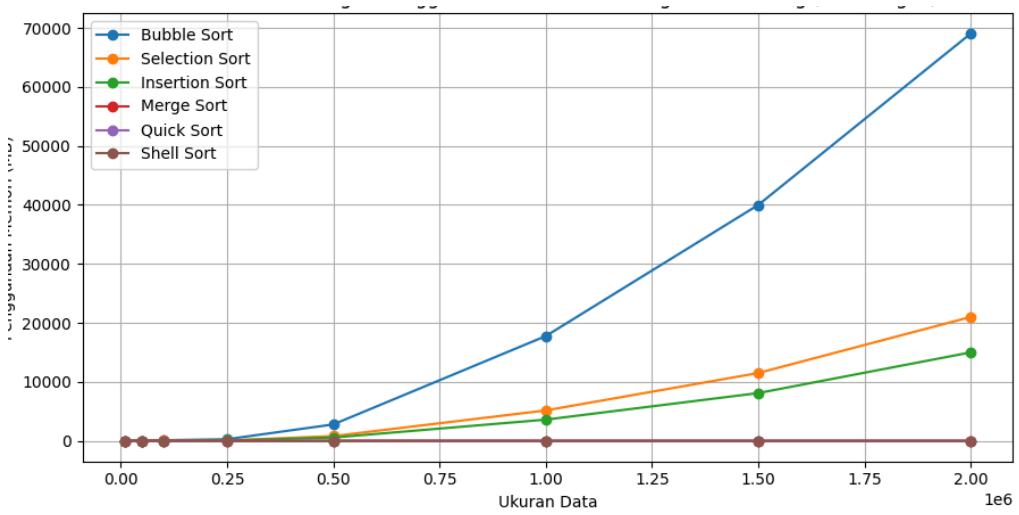
3. Grafik Perbandingan Waktu dan Memori

Grafik Waktu Eksekusi

- **Data Angka**

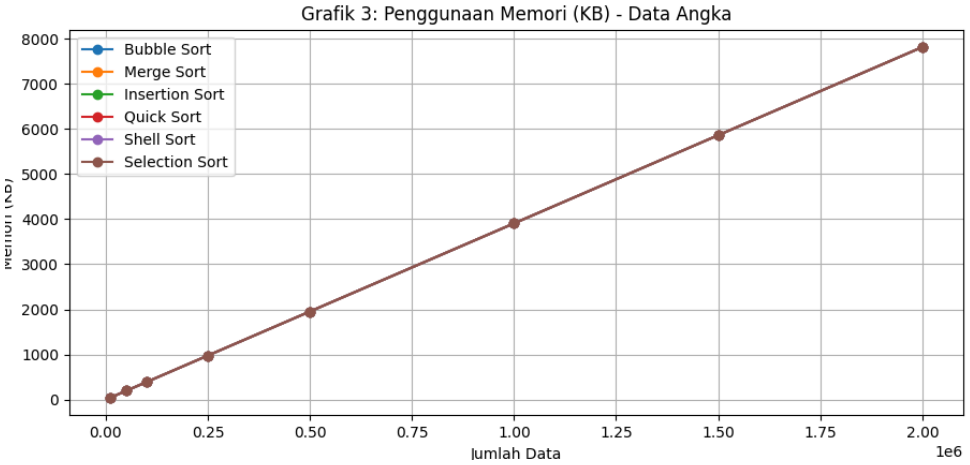


- **Data Kata**

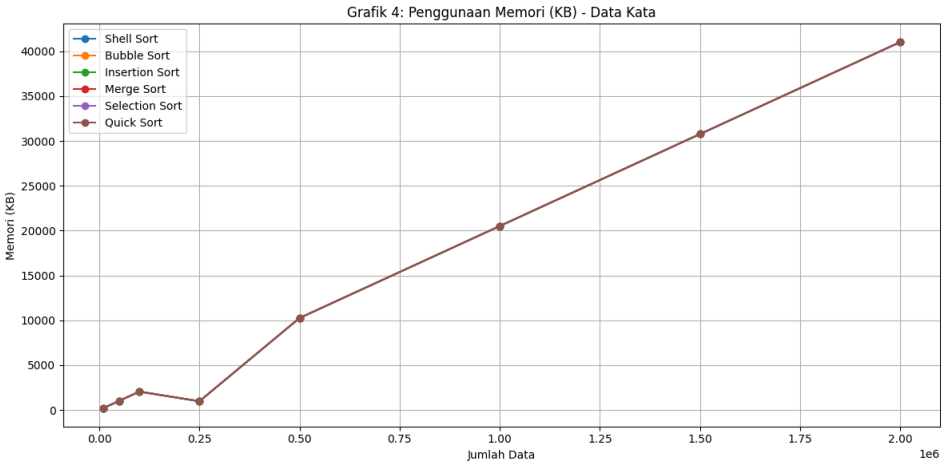


Grafik Penggunaan Memori

- **Data Angka**



- **Data Kata**



ANALISIS

Analisa didapat berdasarkan grafik-grafik di atas yang menunjukkan efisiensi algoritma sorting.

1. Grafik 1

   Bubble Sort membutuhkan waktu paling lama dibandingkan algoritma lain, dengan pertumbuhan waktu yang sangat drastis seiring bertambahnya ukuran data. Selection Sort dan Insertion Sort memiliki waktu lebih cepat dari Bubble, tetapi tetap jauh lebih lambat dibanding algoritma lain. Merge Sort, Quick Sort, dan Shell Sort menunjukkan waktu eksekusi yang jauh lebih kecil dan stabil meskipun jumlah data sangat besar. Quick Sort menjadi algoritma tercepat hampir di semua ukuran data.

2. Grafik 2

   Bubble Sort menggunakan memori paling banyak, bahkan terus meningkat tajam seiring bertambahnya jumlah data. Selection Sort dan Insertion Sort membutuhkan memori lebih rendah dibanding Bubble Sort, tetapi tetap bertambah dengan ukuran data. Merge Sort, Quick Sort, dan Shell Sort memiliki penggunaan memori yang relatif lebih rendah dan stabil.
3. Grafik 3

   Semua algoritma menunjukkan pola penggunaan memori yang sangat mirip.Tidak ada perbedaan besar antar algoritma untuk penggunaan memori di data angka. Ini menunjukkan bahwa penggunaan memori cenderung tergantung pada jumlah data, bukan jenis algoritma.
4. Grafik 4

   Pola serupa dengan data angka, namun penggunaan memori sedikit lebih tinggi. Semua algoritma mengikuti tren pertumbuhan memori yang hampir sama.

KESIMPULAN

1. Dari segi waktu eksekusi, algoritma Quick Sort adalah yang tercepat, diikuti oleh Merge Sort dan Shell Sort. Sementara itu, Bubble Sort paling lambat, apalagi pada data yang berukuran sangat besar.

2. Dari segi penggunaan memori, semua algoritma pada dasarnya bertambah seiring ukuran data, namun Bubble Sort menggunakan memori lebih besar dibanding yang lain. Sedangkan Merge Sort, Quick Sort, dan Shell Sort lebih hemat memori.

3. Efisiensi waktu lebih penting ketika ukuran data semakin besar. Maka untuk aplikasi nyata (real-world), sebaiknya memilih algoritma seperti Quick Sort atau Merge Sort.

4. Performa sorting untuk data angka dan data kata menunjukkan pola yang hampir sama. Namun untuk data kata, penggunaan memori sedikit lebih besar karena karakter string membutuhkan lebih banyak alokasi memori dibanding angka.