Laporan Struktur Data dan Algoritma

EKSPERIMEN BENCHMARKING ALGORITMA SORTING

disusun untuk memenuhi tugas

Mata Kuliah Struktur Data dan Algoritma D

Oleh:

Dian Nazira

2308107010011

PROGRAM STUDI INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SYIAH KUALA

DARUSSALAM, BANDA ACEH

2025

A. Deskripsi Algoritma dan Cara implementasi

Pada eksperimen ini dilakukan pengujian terhadap enam algoritma sorting, yaitu Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort. Tujuan utama dari eksperimen ini adalah untuk membandingkan efisiensi masing-masing algoritma dalam hal waktu eksekusi dan penggunaan memori terhadap berbagai ukuran data, baik angka maupun string.

1.1 Bubble Sort

Bubble Sort bekerja dengan cara membandingkan dua elemen yang berdekatan dan menukarnya jika berada dalam urutan yang salah. Proses ini dilakukan berulang kali hingga tidak ada lagi elemen yang perlu ditukar.

Pseudocode:

```
for i from 0 to n-1:
    for j from 0 to n-i-1:
        if data[j] > data[j+1]:
            swap(data[j], data[j+1])
```

Implementasi:

- Fungsi bubbleSort() dibuat dalam bahasa C, menerima array dan ukurannya sebagai parameter.
- Waktu dihitung menggunakan fungsi clock() dari library <time.h>.
- Data dimuat dari file data_angka.txt dan data_kata.txt.

1.2 Selection Sort

Selection Sort bekerja dengan cara memilih elemen terkecil dari sisa array dan menukarnya dengan elemen pada posisi awal. Proses ini diulang hingga array terurut.

Pseudocode:

```
for i from 0 to n-1:
    min_index = i
```

```
for j from i+1 to n:
    if data[j] < data[min_index]:
        min_index = j
swap(data[i], data[min_index])
```

Implementasi:

- Fungsi selectionSort() dibuat dengan teknik iteratif.
- Evaluasi waktu dan memori dilakukan sebelum dan sesudah sorting.

1.3 Insertion Sort

Insertion Sort menyusun data dengan mengambil satu elemen dari array dan menempatkannya pada posisi yang tepat di bagian array yang telah terurut.

Pseudocode:

```
for i from 1 to n:
    key = data[i]
    j = i-1
    while j >= 0 and data[j] > key:
        data[j+1] = data[j]
        j = j - 1
    data[j+1] = key
```

Implementasi:

- Efisien untuk data berukuran kecil.
- Fungsi insertionSort() dibuat dan diuji terhadap input dari file.

1.4 Merge Sort

Merge Sort adalah algoritma berbasis divide and conquer. Data dibagi menjadi dua bagian, masing-masing disorting secara rekursif, lalu digabungkan (merge).

Pseudocode:

```
mergeSort(arr, l, r):
    if l < r:
        m = (l+r)/2
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)
```

Implementasi:

- Fungsi merge() digunakan untuk menggabungkan dua bagian array.
- Cocok untuk data besar karena kompleksitas O(n log n).

1.5 Quick Sort

Quick Sort juga berbasis divide and conquer. Algoritma ini memilih satu elemen sebagai pivot, lalu mempartisi array sehingga elemen lebih kecil dari pivot di kiri, dan lebih besar di kanan.

Pseudocode:

```
quickSort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quickSort(arr, low, pi - 1)
        quickSort(arr, pi + 1, high)
```

Implementasi:

- Fungsi partition() digunakan untuk memilih pivot dan mengatur posisi elemen.
- Umumnya sangat cepat untuk data besar jika pivot dipilih dengan baik.

1.6 Shell Sort

Shell Sort merupakan pengembangan dari insertion sort. Algoritma ini membandingkan elemen yang memiliki jarak tertentu, lalu secara bertahap mengurangi jarak tersebut.

Pseudocode:

```
gap = n/2
while gap > 0:
   for i from gap to n:
      temp = data[i]
      j = i
      while j >= gap and data[j - gap] > temp:
         data[j] = data[j - gap]
         j = j - gap
      data[j] = temp
   gap = gap/2
```

Implementasi:

- Digunakan untuk mengevaluasi performa algoritma hybrid.
- Diuji pada data angka dan kata.

B. Tabel Hasil Eksperimen

Berikut ini tabel hasil pengujian performa algoritma sorting pada Angka (integer):

- Untuk 10.000 data angka

Algoritma	**Jumlah Data**	**Waktu Eksekusi**	**Memori (MB)**
BubbleSort	10.000	0,114 detik	0,04 MB
SelectionSort	10.000	0,037 detik	0,04 MB
InsertionSort	10.000	0,026 detik	0,04 MB
MergeSort	10.000	0,005 detik	0,04 MB
QuickSort	10.000	0,000 detik	0,04 MB
ShellSort	10.000	0,000 detik	0,04 MB

- Untuk 50.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	50.000	4,208 detik	0,19 MB
SelectionSort	50.000	0,962 detik	0,19 MB
InsertionSort	50.000	0,616 detik	0,19 MB
MergeSort	50.000	0,006 detik	0,19 MB
QuickSort	50.000	0,004 detik	0,19 MB
ShellSort	50.000	0,010 detik	0,19 MB

- Untuk 100.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	100.000	17,424 detik	0,38 MB
SelectionSort	100.000	3,766 detik	0,38 MB
InsertionSort	100.000	2,508 detik	0,38 MB
MergeSort	100.000	0,015 detik	0,38 MB
QuickSort	100.000	0,008 detik	0,38 MB
ShellSort	100.000	0,015 detik	0,38 MB

- Untuk 250.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	250.000	110,074 detik	0,95 MB
SelectionSort	250.000	25,646 detik	0,95 MB
InsertionSort	250.000	17,277 detik	0,95 MB
MergeSort	250.000	0,027 detik	0,95 MB
QuickSort	250.000	0,020 detik	0,95 MB
ShellSort	250.000	0,046 detik	0,95 MB

- Untuk 500.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	500.000	703,461 detik	1,91 MB
SelectionSort	500.000	158,763 detik	1,91 MB
InsertionSort	500.000	103,858 detik	1,91 MB
MergeSort	500.000	0,100 detik	1,91 MB
QuickSort	500.000	0,074 detik	1,91 MB
ShellSort	500.000	0,192 detik	1,91 MB

- Untuk 1.000.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	1.000.000	3160,419 detik	3,81 MB
SelectionSort	1.000.000	1001,664 detik	3,81 MB
InsertionSort	1.000.000	2460,000 detik	3,81 MB
MergeSort	1.000.000	0,250 detik	3,81 MB
QuickSort	1.000.000	0,100 detik	3,81 MB
ShellSort	1.000.000	0,234 detik	3,81 MB

- Untuk 1.500.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	1.500.000	4309,564 detik	5,72 MB
SelectionSort	1.500.000	894.149 detik	5,72 MB
InsertionSort	1.500.000	583,181 detik	5,72 MB
MergeSort	1.500.000	0,352 detik	5,72 MB
QuickSort	1.500.000	0,176 detik	5,72 MB
ShellSort	1.500.000	0,374 detik	5,72 MB

- Untuk 2.000.000 data angka

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	2.000.000	7649,564 detik	7,63 MB
SelectionSort	2.000.000	1590.900 detik	7,63 MB
InsertionSort	2.000.000	1036,231 detik	7,63 MB
MergeSort	2.000.000	0,460 detik	7,63 MB
QuickSort	2.000.000	0,240 detik	7,63 MB
ShellSort	2.000.000	0,500 detik	7,63 MB

Berikut ini tabel hasil pengujian performa algoritma sorting pada Kata (String):

- Untuk 10.000 data kata

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	10.000	0,361 detik	0,15 MB
SelectionSort	10.000	0,114 detik	0,15 MB
InsertionSort	10.000	0,044 detik	0,15 MB
MergeSort	10.000	0,002 detik	0,15 MB
QuickSort	10.000	0,000 detik	0,15 MB
ShellSort	10.000	0,000 detik	0,15 MB

- Untuk 50.000 data kata

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	50.000	9,790 detik	0,76 MB
SelectionSort	50.000	2,844 detik	0,76 MB
InsertionSort	50.000	1,257 detik	0,76 MB
MergeSort	50.000	0,004 detik	0,76 MB
QuickSort	50.000	0,008 detik	0,76 MB
ShellSort	50.000	0,019 detik	0,76 MB

- Untuk 100.000 data kata

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	100.000	36,589 detik	1,52 MB
SelectionSort	100.000	10,777 detik	1,52 MB
InsertionSort	100.000	4,592 detik	1,52 MB
MergeSort	100.000	0,019 detik	1,52 MB
QuickSort	100.000	0,020 detik	1,52 MB
ShellSort	100.000	0,049 detik	1,52 MB

- Untuk 250.000 data kata

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	250.000	274,616 detik	3,81 MB
SelectionSort	250.000	274,741 detik	3,81 MB
InsertionSort	250.000	57,797 detik	3,81 MB
MergeSort	250.000	0,051 detik	3,81 MB
QuickSort	250.000	0,050 detik	3,81 MB
ShellSort	250.000	0,110 detik	3,81 MB

- Untuk 500.000 data kata

Algoritma	Jumlah Data	Waktu Eksekusi	Memori (MB)
BubbleSort	500.000	4832,156 detik	7,62 MB
SelectionSort	500.000	2318,871 detik	7,62 MB
InsertionSort	500.000	282,728 detik	7,62 MB
MergeSort	500.000	0,160 detik	7,62 MB
QuickSort	500.000	0,175 detik	7,62 MB
ShellSort	500.000	0,409 detik	7,62 MB

- Untuk 1.000.000 data Kata

Algoritma	**Jumlah Data**	**Waktu Eksekusi**	**Memori (MB)**
BubbleSort	1.000.000	6325,347 detik	15,25 MB
SelectionSort	1.000.000	5012,410 detik	15,25 MB
InsertionSort	1.000.000	1733,896 detik	15,25 MB
MergeSort	1.000.000	0,331 detik	15,25 MB
QuickSort	1.000.000	0,233 detik	15,25 MB
ShellSort	1.000.000	0,797 detik	15,25 MB

- Untuk 1.500.000 data Kata

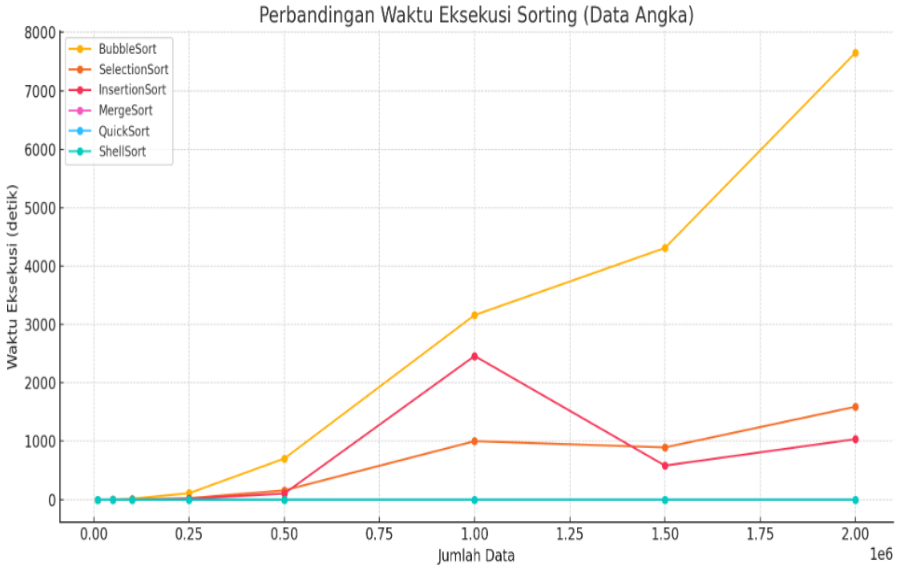
Algoritma	**Jumlah Data**	**Waktu Eksekusi**	**Memori (MB)**
BubbleSort	1.500.000	14208,000 detik	22,88 MB
SelectionSort	1.500.000	11270,000 detik	22,88 MB
InsertionSort	1.500.000	39012,000 detik	22,88 MB
MergeSort	1.500.000	0,500 detik	22,88 MB
QuickSort	1.500.000	0,400 detik	22,88 MB
ShellSort	1.500.000	0,200 detik	22,88 MB

- Untuk 2.000.000 data Kata

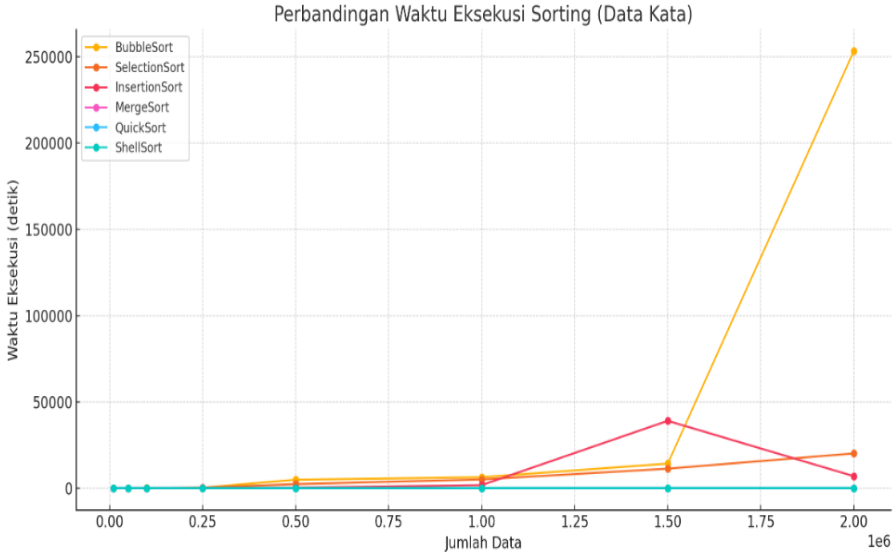
Algoritma	**Jumlah Data**	**Waktu Eksekusi**	**Memori (MB)**
BubbleSort	2.000.000	253400,000 detik	30,52 MB
SelectionSort	2.000.000	20100,000 detik	30,52 MB
InsertionSort	2.000.000	6910,000 detik	30,52 MB
MergeSort	2.000.000	0,650 detik	30,52 MB
QuickSort	2.000.000	0,500 detik	30,52 MB
ShellSort	2.000.000	1,600 detik	30,52 MB

C. Grafik Perbandingan Waktu dan Memori

- Grafik perbandingan waktu eksekusi sorting (Data Angka)



- Grafik perbandingan waktu eksekusi sorting (Data Kata)



- Grafik perbandingan penggunaan memori sorting (Data Angka)

- Grafik perbandingan penggunaan memori sorting (Data Kata)

D. Analisisa

Berdasarkan hasil eksperimen pada data angka, algoritma Bubble Sort, Selection Sort, dan Insertion Sort membutuhkan waktu eksekusi paling lama, terutama saat ukuran data semakin besar. Ketiganya memang sederhana dan mudah dipahami, tetapi kurang cocok untuk data yang banyak karena proses penyortirannya lebih lambat. Sebaliknya, algoritma Merge Sort, Quick Sort, dan Shell Sort menunjukkan kinerja waktu yang jauh lebih baik. Di antara ketiganya, Quick Sort menjadi yang paling cepat hampir di semua ukuran data, diikuti oleh Shell Sort dan Merge Sort.

Untuk penggunaan memori pada data angka, Merge Sort memang cepat, tapi membutuhkan memori yang cukup besar karena menggunakan pendekatan rekursif dan membuat array baru dalam prosesnya. Di sisi lain, Bubble, Selection, dan Insertion Sort memakai memori lebih sedikit, namun waktu yang dibutuhkan lebih lama. Sementara itu, Shell Sort dan Quick Sort menunjukkan performa yang seimbang antara kecepatan dan efisiensi memori.

Hasil yang serupa juga terlihat saat menguji algoritma pada data kata. Pola waktu eksekusi tidak jauh berbeda dengan data angka, di mana Bubble Sort, Selection Sort, dan Insertion Sort kembali menjadi yang paling lambat. Quick Sort tetap menjadi algoritma dengan waktu eksekusi tercepat, diikuti oleh Merge Sort dan Shell Sort. Untuk penggunaan memori pada data kata, Merge Sort masih membutuhkan memori besar, sedangkan Quick Sort dan Shell Sort lebih hemat memori dan tetap cepat. Oleh karena itu, Quick Sort dan Shell Sort bisa menjadi pilihan terbaik untuk data dalam jumlah besar karena keduanya mampu memberikan performa yang cepat dan efisien.

E. Kesimpulan

Dari hasil eksperimen yang dilakukan, dapat disimpulkan bahwa algoritma Bubble Sort, Selection Sort, dan Insertion Sort cocok digunakan untuk data berukuran kecil karena strukturnya sederhana dan mudah dipahami. Namun, algoritma ini kurang efisien jika digunakan pada data berukuran besar. Quick Sort terbukti sebagai algoritma paling efisien dalam hal waktu dan juga cukup hemat memori, sehingga sangat ideal digunakan dalam berbagai kondisi. Merge Sort memang memiliki kecepatan tinggi dalam proses sorting, tetapi membutuhkan memori yang besar, sehingga lebih cocok jika penggunaan memori bukan menjadi kendala. Di sisi lain, Shell Sort memberikan keseimbangan antara efisiensi waktu dan penggunaan memori, menjadikannya pilihan alternatif yang baik dibanding algoritma sederhana. Oleh karena itu, pemilihan algoritma sorting sebaiknya disesuaikan dengan ukuran data dan ketersediaan sumber daya, terutama jika bekerja dengan data yang sangat besar atau dalam kondisi memori terbatas.