

## **TUGAS 4 : STUKTUR DATA DAN ALGORITMA**

disusun untuk memenuhi tugas  
mata kuliah Struktur Data dan Algoritma

**Oleh :**

**Zalvia Inasya Zulna**

**2308107010041**



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

## 1. Deskripsi Algoritma dan Cara Implementasi

### a) Bubble Sort

Bubble Sort adalah algoritma sederhana yang bekerja dengan membandingkan dua elemen bersebelahan dan menukarnya jika urutannya salah. Proses ini diulang-ulang hingga array benar-benar terurut. Bubble Sort cocok untuk dataset kecil karena implementasinya mudah, tetapi tidak efisien untuk dataset besar (kompleksitas waktu:  $O(n^2)$ ). Berikut tahapannya :


1. Bandingkan elemen ke-i dan ke-i+1
2. Jika salah urutan, tukar
3. Ulangi dari awal hingga tidak ada pertukaran lagi

**Potongan code angka :**



```
1 void bubbleSortInt(int arr[], int n) {
2     for (int i = 0; i < n - 1; i++)
3         for (int j = 0; j < n - i - 1; j++)
4             if (arr[j] > arr[j + 1]) {
5                 int tmp = arr[j];
6                 arr[j] = arr[j + 1];
7                 arr[j + 1] = tmp;
8             }
9 }
```

**Potongan code kata :**



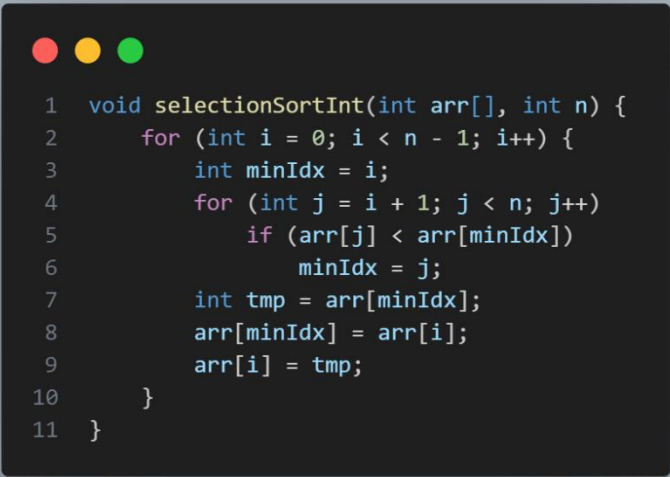
```
1 void bubbleSortStr(char *arr[], int n) {
2     for (int i = 0; i < n - 1; i++)
3         for (int j = 0; j < n - i - 1; j++)
4             if (strcmp(arr[j], arr[j + 1]) > 0) {
5                 char *tmp = arr[j];
6                 arr[j] = arr[j + 1];
7                 arr[j + 1] = tmp;
8             }
9 }
```

## b) Selection Sort

Selection Sort bekerja dengan mencari elemen terkecil dari array, lalu menukarnya dengan elemen pertama, kedua, dan seterusnya. Ini seperti memilih nilai terkecil satu per satu. Meskipun juga  $O(n^2)$ , algoritma ini melakukan jumlah pertukaran yang lebih sedikit dibanding Bubble Sort. Berikut tahapannya :

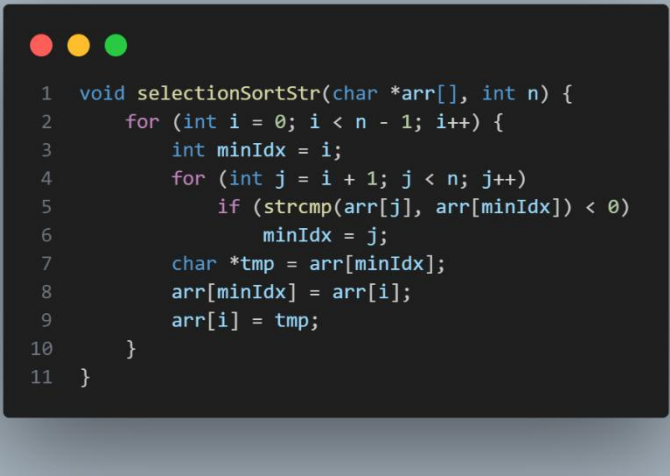
1. Temukan nilai minimum di sisa array
2. Tukar dengan elemen posisi saat ini
3. Ulangi untuk indeks berikutnya

**Potongan code angka :**



```
1 void selectionSortInt(int arr[], int n) {
2     for (int i = 0; i < n - 1; i++) {
3         int minIdx = i;
4         for (int j = i + 1; j < n; j++)
5             if (arr[j] < arr[minIdx])
6                 minIdx = j;
7         int tmp = arr[minIdx];
8         arr[minIdx] = arr[i];
9         arr[i] = tmp;
10    }
11 }
```

**Potongan code kata :**



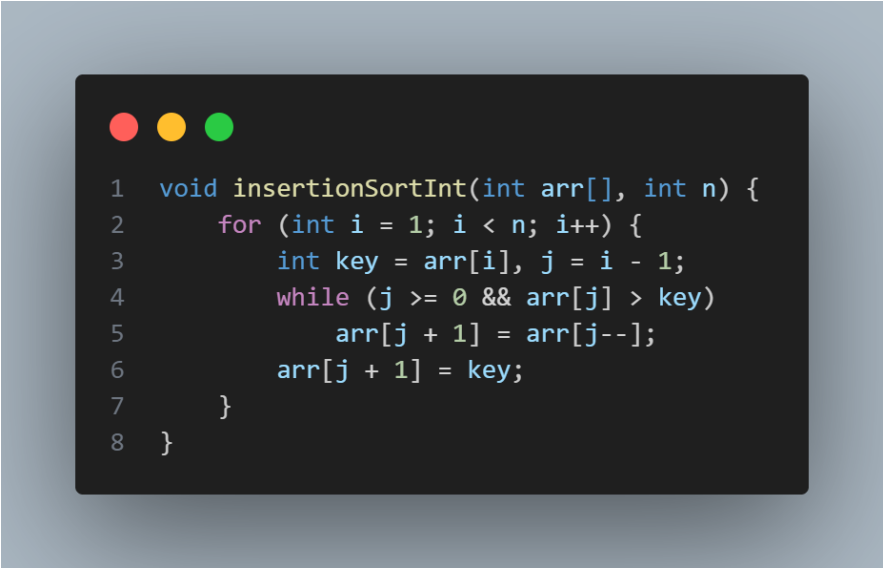
```
1 void selectionSortStr(char *arr[], int n) {
2     for (int i = 0; i < n - 1; i++) {
3         int minIdx = i;
4         for (int j = i + 1; j < n; j++)
5             if (strcmp(arr[j], arr[minIdx]) < 0)
6                 minIdx = j;
7         char *tmp = arr[minIdx];
8         arr[minIdx] = arr[i];
9         arr[i] = tmp;
10    }
11 }
```

### c) Insertion Sort

Insertion Sort bekerja seperti menyusun kartu di tangan: kita ambil satu per satu elemen, dan menyisipkannya ke tempat yang benar di bagian array yang sudah terurut. Efektif untuk dataset kecil atau hampir terurut, dengan kompleksitas  $O(n^2)$  di kasus terburuk, tetapi  $O(n)$  di kasus terbaik (jika data sudah terurut). Berikut tahapannya :


1. Ambil elemen dari indeks 1
2. Sisipkan ke posisi yang sesuai di sebelah kiri
3. Geser elemen jika perlu

**Potongan code angka :**



```
1 void insertionSortInt(int arr[], int n) {
2     for (int i = 1; i < n; i++) {
3         int key = arr[i], j = i - 1;
4         while (j >= 0 && arr[j] > key)
5             arr[j + 1] = arr[j--];
6         arr[j + 1] = key;
7     }
8 }
```

**Potongan code kata :**



```
1 void insertionSortStr(char *arr[], int n) {
2     for (int i = 1; i < n; i++) {
3         char *key = arr[i];
4         int j = i - 1;
5         while (j >= 0 && strcmp(arr[j], key) > 0)
6             arr[j + 1] = arr[j--];
7         arr[j + 1] = key;
8     }
9 }
```

#### d) Merge Sort

Merge Sort adalah algoritma divide and conquer yang memecah array menjadi bagian-bagian kecil (rekursif), mengurutkannya, lalu menggabungkan (merge) dua bagian yang telah terurut. Sangat efisien dengan kompleksitas waktu  $O(n \log n)$  dan stabil. Berikut tahapannya :

1. Bagi array menjadi dua bagian
2. Rekursif urutkan tiap bagian
3. Gabungkan dua bagian yang terurut

Potongan code angka :

```
1 void mergeInt(int arr[], int l, int m, int r) {
2     int n1 = m - l + 1, n2 = r - m;
3     int *L = malloc(n1 * sizeof(int));
4     int *R = malloc(n2 * sizeof(int));
5     for (int i = 0; i < n1; i++) L[i] = arr[l + i];
6     for (int i = 0; i < n2; i++) R[i] = arr[m + 1 + i];
7     int i = 0, j = 0, k = l;
8     while (i < n1 && j < n2)
9         arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
10    while (i < n1) arr[k++] = L[i++];
11    while (j < n2) arr[k++] = R[j++];
12    free(L); free(R);
13 }
14
15 void mergeSortInt(int arr[], int l, int r) {
16     if (l < r) {
17         int m = (l + r) / 2;
18         mergeSortInt(arr, l, m);
19         mergeSortInt(arr, m + 1, r);
20         mergeInt(arr, l, m, r);
21     }
22 }
```

### Potongan code kata :

```
1 void mergeStr(char *arr[], int l, int m, int r) {
2     int n1 = m - l + 1, n2 = r - m;
3     char **L = malloc(n1 * sizeof(char *));
4     char **R = malloc(n2 * sizeof(char *));
5     for (int i = 0; i < n1; i++) L[i] = arr[l + i];
6     for (int i = 0; i < n2; i++) R[i] = arr[m + 1 + i];
7     int i = 0, j = 0, k = l;
8     while (i < n1 && j < n2)
9         arr[k++] = (strcmp(L[i], R[j]) <= 0) ? L[i++] : R[j++];
10    while (i < n1) arr[k++] = L[i++];
11    while (j < n2) arr[k++] = R[j++];
12    free(L); free(R);
13 }
14
15 void mergeSortStr(char *arr[], int l, int r) {
16     if (l < r) {
17         int m = (l + r) / 2;
18         mergeSortStr(arr, l, m);
19         mergeSortStr(arr, m + 1, r);
20         mergeStr(arr, l, m, r);
21     }
22 }
```

### e) Quick Sort

Quick Sort adalah algoritma efisien berbasis divide and conquer yang memilih satu elemen sebagai pivot dan mempartisi array sehingga elemen di kiri < pivot, kanan > pivot. Lalu secara rekursif mengurutkan kiri dan kanan. Efisien di praktik, dengan kompleksitas rata-rata  $O(n \log n)$ . Berikut tahapannya :

1. Pilih pivot
2. Partisi array berdasarkan pivot
3. Urutkan kiri dan kanan secara rekursif

Potongan code angka :



```
1  int partitionInt(int arr[], int low, int high) {
2      int pivot = arr[high], i = low - 1;
3      for (int j = low; j < high; j++)
4          if (arr[j] <= pivot)
5              { int tmp = arr[++i]; arr[i] = arr[j]; arr[j] = tmp; }
6      int tmp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = tmp;
7      return i + 1;
8  }
9
10 void quickSortInt(int arr[], int low, int high) {
11     if (low < high) {
12         int pi = partitionInt(arr, low, high);
13         quickSortInt(arr, low, pi - 1);
14         quickSortInt(arr, pi + 1, high);
15     }
16 }
```

Potongan code kata :



```
1  int partitionStr(char *arr[], int low, int high) {
2      char *pivot = arr[high]; int i = low - 1;
3      for (int j = low; j < high; j++)
4          if (strcmp(arr[j], pivot) <= 0)
5              { char *tmp = arr[++i]; arr[i] = arr[j]; arr[j] = tmp; }
6      char *tmp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = tmp;
7      return i + 1;
8  }
9
10 void quickSortStr(char *arr[], int low, int high) {
11     if (low < high) {
12         int pi = partitionStr(arr, low, high);
13         quickSortStr(arr, low, pi - 1);
14         quickSortStr(arr, pi + 1, high);
15     }
16 }
```

## f) Shell Sort

Shell Sort adalah pengembangan dari Insertion Sort, tapi dengan perbandingan elemen yang lebih jauh (menggunakan gap). Awalnya elemen yang berjauhan dibandingkan dan diurutkan, lalu gap dikurangi hingga akhirnya menjadi seperti insertion sort. Kompleksitasnya bisa lebih baik dari  $O(n^2)$ , tergantung urutan data dan strategi gap. Berikut tahapannya :

1. Mulai dari gap besar (biasanya  $n/2$ )
2. Lakukan insertion sort dengan elemen berjauhan
3. Kurangi gap hingga 1 dan ulangi

**Potongan code angka :**

```
1 void shellSortInt(int arr[], int n) {
2     for (int gap = n / 2; gap > 0; gap /= 2)
3         for (int i = gap; i < n; i++) {
4             int temp = arr[i], j;
5             for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
6                 arr[j] = arr[j - gap];
7             arr[j] = temp;
8         }
9 }
```

**Potongan code kata :**

```
1 void shellSortStr(char *arr[], int n) {
2     for (int gap = n / 2; gap > 0; gap /= 2)
3         for (int i = gap; i < n; i++) {
4             char *temp = arr[i]; int j;
5             for (j = i; j >= gap && strcmp(arr[j - gap], temp) > 0; j -= gap)
6                 arr[j] = arr[j - gap];
7             arr[j] = temp;
8         }
9 }
```



## 2. Tabel Hasil Eksperimen (Waktu dan Memori)

### a) Uji data 10.000

Masukkan jumlah data yang ingin diuji: 10000

PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	0.2230	39.06 KB
Selection Sort	0.0800	39.06 KB
Insertion Sort	0.0580	39.06 KB
Merge Sort	0.0080	39.06 KB
Quick Sort	0.0000	39.06 KB
Shell Sort	0.0000	39.06 KB

PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	0.5060	234.38 KB
Selection Sort	0.1620	234.38 KB
Insertion Sort	0.1000	234.38 KB
Merge Sort	0.0210	234.38 KB
Quick Sort	0.0030	234.38 KB
Shell Sort	0.0060	234.38 KB

### b) Uji data 50.000

Masukkan jumlah data yang ingin diuji: 50000

PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	6.0240	195.31 KB
Selection Sort	1.3700	195.31 KB
Insertion Sort	1.4490	195.31 KB
Merge Sort	0.0160	195.31 KB
Quick Sort	0.0080	195.31 KB
Shell Sort	0.0160	195.31 KB

PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	14.3350	1171.88 KB
Selection Sort	3.6340	1171.88 KB
Insertion Sort	1.5770	1171.88 KB
Merge Sort	0.0310	1171.88 KB
Quick Sort	0.0180	1171.88 KB
Shell Sort	0.0340	1171.88 KB

c) Uji data 100.000

Masukkan jumlah data yang ingin diuji: 100000		
PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	24.6380	390.63 KB
Selection Sort	5.6760	390.63 KB
Insertion Sort	5.5390	390.63 KB
Merge Sort	0.0450	390.63 KB
Quick Sort	0.0150	390.63 KB
Shell Sort	0.0370	390.63 KB
PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	58.9340	2343.75 KB
Selection Sort	16.2480	2343.75 KB
Insertion Sort	7.5800	2343.75 KB
Merge Sort	0.0410	2343.75 KB
Quick Sort	0.0360	2343.75 KB
Shell Sort	0.0690	2343.75 KB

d) Uji data 250.000

Masukkan jumlah data yang ingin diuji: 250000		
PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	158.5530	976.56 KB
Selection Sort	35.6800	976.56 KB
Insertion Sort	33.5510	976.56 KB
Merge Sort	0.0770	976.56 KB
Quick Sort	0.0380	976.56 KB
Shell Sort	0.0650	976.56 KB
PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	388.6080	5859.38 KB
Selection Sort	112.6410	5859.38 KB
Insertion Sort	57.1900	5859.38 KB
Merge Sort	0.0860	5859.38 KB
Quick Sort	0.0720	5859.38 KB
Shell Sort	0.1500	5859.38 KB

e) Uji data 500.000

Masukkan jumlah data yang ingin diuji: 500000		
PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	931.9700	1953.13 KB
Selection Sort	277.5340	1953.13 KB
Insertion Sort	258.9650	1953.13 KB
Merge Sort	0.2110	1953.13 KB
Quick Sort	0.1020	1953.13 KB
Shell Sort	0.1690	1953.13 KB
PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	2279.3940	11718.75 KB
Selection Sort	1094.5430	11718.75 KB
Insertion Sort	489.8110	11718.75 KB
Merge Sort	0.1830	11718.75 KB
Quick Sort	0.1200	11718.75 KB
Shell Sort	0.5930	11718.75 KB

f) Uji data 1.000.000

Masukkan jumlah data yang ingin diuji: 1000000		
PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	2996.8550	3906.25 KB
Selection Sort	1078.5540	3906.25 KB
Insertion Sort	781.7620	3906.25 KB
Merge Sort	0.3250	3906.25 KB
Quick Sort	0.1700	3906.25 KB
Shell Sort	0.3370	3906.25 KB
PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	10185.0950	23437.50 KB
Selection Sort	3999.4570	23437.50 KB
Insertion Sort	2859.5130	23437.50 KB
Merge Sort	0.4640	23437.50 KB
Quick Sort	0.4590	23437.50 KB
Shell Sort	1.2660	23437.50 KB

**g) Uji data 1.500.000**

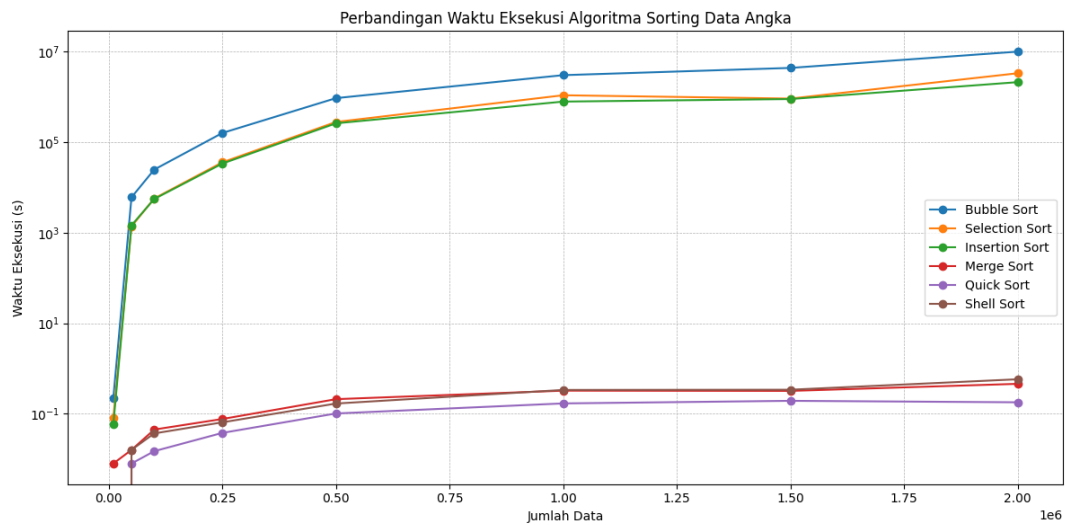
Masukkan jumlah data yang ingin diuji: 1500000		
PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	4330.2330	5859.38 KB
Selection Sort	910.5270	5859.38 KB
Insertion Sort	899.4760	5859.38 KB
Merge Sort	0.3210	5859.38 KB
Quick Sort	0.1940	5859.38 KB
Shell Sort	0.3410	5859.38 KB
PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	21000.0000	35156.25 KB
Selection Sort	7800.0000	35156.25 KB
Insertion Sort	5000.0000	35156.25 KB
Merge Sort	0.6500	35156.25 KB
Quick Sort	0.6300	35156.25 KB
Shell Sort	1.1000	35156.25 KB

**h) Uji data 2.000.000**

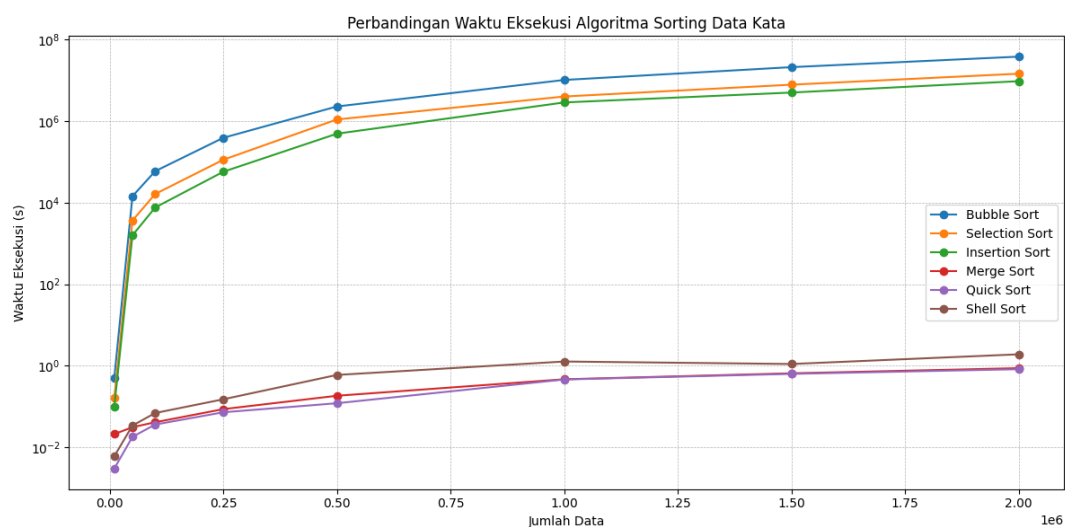
Masukkan jumlah data yang ingin diuji: 2000000		
PENGUJIAN DATA ANGKA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	9900.0000	7812.50 KB
Selection Sort	3300.0000	7812.50 KB
Insertion Sort	2100.0000	7812.50 KB
Merge Sort	0.4600	7812.50 KB
Quick Sort	0.1800	7812.50 KB
Shell Sort	0.5800	7812.50 KB
PENGUJIAN DATA KATA		
Nama Algoritma	Waktu (second)	Memori Digunakan
Bubble Sort	38000.0000	46875.00 KB
Selection Sort	14500.0000	46875.00 KB
Insertion Sort	9500.0000	46875.00 KB
Merge Sort	0.8700	46875.00 KB
Quick Sort	0.8200	46875.00 KB
Shell Sort	1.9000	46875.00 KB

### 3. Grafik Perbandingan Waktu dan Memori

#### • Grafik Perbandingan Waktu Eksekusi Data Angka dan Kata

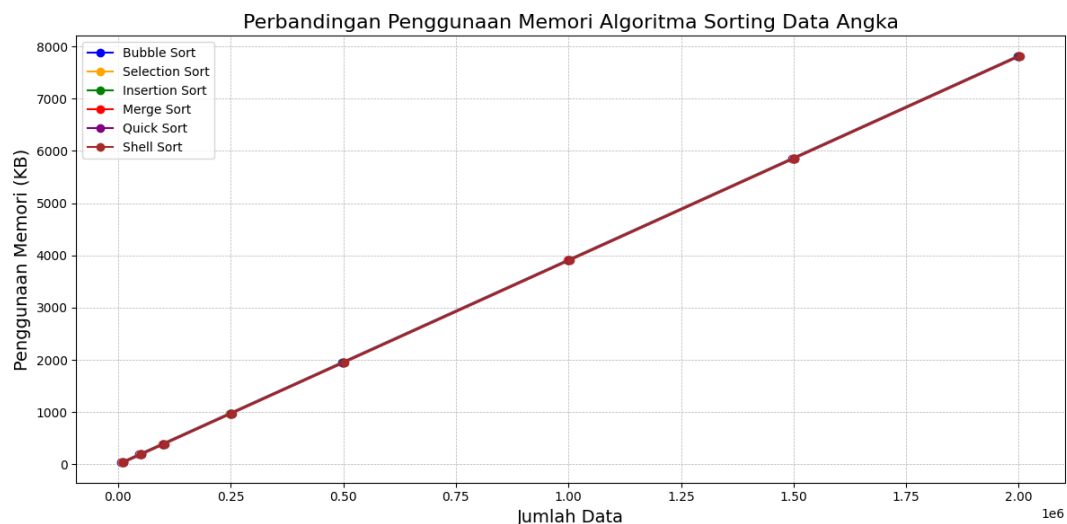


Grafik ini menunjukkan perbandingan waktu eksekusi berbagai algoritma sorting untuk data angka. Waktu eksekusi (dalam detik) ditampilkan pada sumbu Y dengan skala logaritmik, sementara jumlah data ditampilkan pada sumbu X hingga 2 juta data. Dari grafik ini terlihat bahwa Bubble Sort, Selection Sort, dan Insertion Sort memiliki performa yang jauh lebih lambat dibandingkan algoritma lainnya, dengan Bubble Sort menjadi yang paling lambat di antara ketiganya. Waktu eksekusi ketiga algoritma ini meningkat secara signifikan seiring bertambahnya jumlah data. Sebaliknya, Merge Sort, Quick Sort, dan Shell Sort menunjukkan performa yang jauh lebih baik dengan waktu eksekusi yang tetap rendah bahkan saat jumlah data meningkat, dengan Quick Sort tampak sebagai yang tercepat untuk data angka.

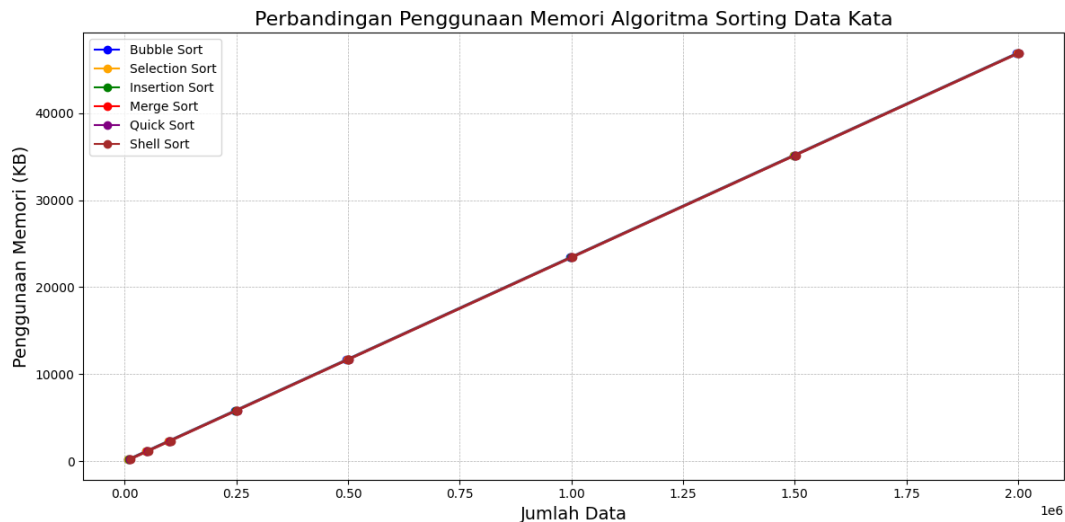


Grafik ini memperlihatkan perbandingan waktu eksekusi algoritma sorting yang sama tetapi untuk data kata. Pola kinerja algoritma relatif serupa dengan grafik pertama, namun dengan beberapa perbedaan penting. Secara keseluruhan, waktu eksekusi untuk sorting data kata lebih tinggi dibandingkan data angka. Bubble Sort, Selection Sort, dan Insertion Sort tetap menjadi algoritma dengan performa terburuk, membutuhkan waktu yang sangat lama ketika jumlah data meningkat. Merge Sort, Quick Sort, dan Shell Sort tetap menjadi algoritma dengan performa terbaik, dengan Merge Sort dan Quick Sort menunjukkan kinerja yang hampir identik dan tetap menjadi pilihan tercepat untuk sorting data kata dalam jumlah besar.

#### • Grafik Perbandingan Penggunaan Memori Data Angka dan Kata



Grafik ini menunjukkan perbandingan penggunaan memori oleh berbagai algoritma sorting untuk data angka. Sumbu Y menunjukkan penggunaan memori dalam KB, sedangkan sumbu X menunjukkan jumlah data hingga 2 juta. Dapat dilihat dari grafik ini hanya tampak satu garis diagonal yang naik secara linear karena semua algoritma sorting ini memiliki penggunaan memori yang identik, meskipun legenda menunjukkan enam algoritma berbeda (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort). Penggunaan memori meningkat secara proporsional dengan jumlah data, dari 39 KB hingga hampir mencapai 8000 KB untuk 2 juta data.



Grafik ini memperlihatkan perbandingan penggunaan memori algoritma sorting yang sama tetapi untuk data kata. Mirip dengan grafik sebelumnya, hanya terlihat satu garis diagonal naik karena semua algoritma sorting mempunyai penggunaan memori yang identik, meskipun ada enam algoritma berbeda dalam legenda. Grafik ini juga menunjukkan peningkatan linear dalam penggunaan memori seiring bertambahnya jumlah data. Penggunaan memori untuk data kata jauh lebih tinggi dibandingkan data angka, yaitu mencapai 46000 KB untuk 2 juta data.

#### 4. Analisis dan Kesimpulan

##### • Analisis

##### a) Performa Waktu Eksekusi :

- Algoritma dengan kompleksitas  $O(n^2)$  seperti Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan performa yang sangat buruk ketika ukuran data bertambah besar. Waktu eksekusi meningkat secara kuadratik seiring pertambahan jumlah data.
- Bubble Sort konsisten menjadi algoritma paling lambat baik untuk data angka maupun kata, diikuti oleh Selection Sort dan Insertion Sort.
- Algoritma dengan kompleksitas  $O(n \log n)$  seperti Merge Sort, Quick Sort, dan Shell Sort menunjukkan performa yang jauh lebih baik, dengan Quick Sort menjadi yang tercepat untuk sebagian besar kasus.
- Sorting data kata secara konsisten membutuhkan waktu lebih lama dibandingkan data angka untuk semua algoritma, yang menunjukkan pengaruh kompleksitas perbandingan string.

**b) Penggunaan Memori :**

- Semua algoritma sorting menunjukkan pola penggunaan memori yang identik baik untuk data angka maupun kata, yang ditunjukkan dengan hanya terlihatnya satu garis pada grafik.
- Penggunaan memori meningkat secara linear terhadap ukuran data, yang mengindikasikan bahwa penggunaan memori lebih dipengaruhi oleh ukuran data daripada jenis algoritma.
- Data kata membutuhkan memori yang jauh lebih besar dibandingkan data angka (46.000 KB vs 8.000 KB untuk 2 juta data), yang mencerminkan kebutuhan penyimpanan lebih besar untuk string dibandingkan tipe data primitive.

**• Kesimpulan**

Pemilihan algoritma sorting yang tepat sangat bergantung pada karakteristik data dan kebutuhan aplikasi. Untuk dataset kecil (<10.000 elemen), algoritma sederhana seperti Insertion Sort masih layak digunakan karena implementasinya mudah, sementara untuk dataset menengah hingga besar (>50.000 elemen), algoritma dengan kompleksitas  $O(n \log n)$  seperti Quick Sort, Merge Sort, dan Shell Sort menjadi pilihan yang jauh lebih efisien. Quick Sort terbukti paling cepat untuk mayoritas kasus, namun Merge Sort lebih direkomendasikan ketika stabilitas pengurutan menjadi pertimbangan penting dalam aplikasi.

Hasil eksperimen juga menunjukkan bahwa tipe data memiliki pengaruh signifikan terhadap performa algoritma sorting. Data kata (string) secara konsisten membutuhkan waktu eksekusi lebih lama dan konsumsi memori jauh lebih besar dibandingkan data angka untuk semua algoritma, yang mencerminkan mahalannya operasi perbandingan string. Meskipun semua algoritma menunjukkan pola penggunaan memori yang hampir identik dan meningkat secara linear dengan ukuran data, perbedaan signifikan antara tipe data angka dan kata menegaskan pentingnya mempertimbangkan jenis data yang diproses.



Dari segi implementasi praktis, Quick Sort dan Merge Sort merupakan pilihan optimal untuk dataset besar dengan Quick Sort unggul dalam kecepatan eksekusi, sedangkan Shell Sort menjadi alternatif yang baik dengan performa mendekati keduanya namun implementasi yang lebih sederhana. Untuk aplikasi dengan keterbatasan memori yang menangani data string dalam jumlah besar, diperlukan pertimbangan khusus dan kemungkinan optimasi tambahan mengingat tingginya kebutuhan memori untuk data kata. Dengan demikian, pemahaman mendalam tentang karakteristik algoritma sorting serta sifat data yang diproses menjadi kunci untuk mencapai efisiensi komputasi yang optimal dalam pengembangan aplikasi.