LAPORAN PRAKTIKUM STRUKTUR DATA

"*Sorting Algorithm 2*"

OLEH :

EZZA ADDINI

2311532001

DOSEN PENGAMPU: Dr. WAHYUDI, M. T.

DEPARTEMEN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

PADANG

2024

A. PENDAHULUAN

Algoritma pengurutan yang lebih dikenal sebagai algoritma sorting merupakan algoritma untuk meletakkan elemen data kedalam kumpulan data urutan tertentu, sehingga proses pengurutan yang sebelumnya masih tersusun acak menjadi tersusun teratur menurut aturan tertentu. Algoritma ini memungkinkan untuk mengurutkan data secara menaik (ascending) ataupun menurun (descending). Tujuannya adalah untuk membuat pencarian lebih mudah bagi himpunan. Kelebihan data yang terurut adalah bahwa data tersebut mudah dicek apabila ada data yang hilang.

Pada praktikum kali ini, algoritma sorting yang digunakan adalah sebagai berikut:

a. Shell Sort

   Shell Sort bekerja dengan cara membandingkan elemen-elemen yang berada pada jarak tertentu (gap) dan mengurutkannya, kemudian mengurangi jarak tersebut secara bertahap hingga jarak menjadi 1, yang mana pada saat itu algoritma akan menjadi Insertion Sort.

b. Quick Sort

   Quick Sort juga menggunakan pendekatan divide-and-conquer. Algoritma ini memilih elemen sebagai pivot dan mempartisi array sehingga elemen yang lebih kecil dari pivot berada di kiri dan yang lebih besar berada di kanan. Proses ini diulangi secara rekursif.

c. Merge Sort

   Merge Sort adalah algoritma yang menggunakan pendekatan divide-and-conquer. Array dibagi menjadi dua bagian, diurutkan secara rekursif, lalu digabungkan kembali.

B. TUJUAN

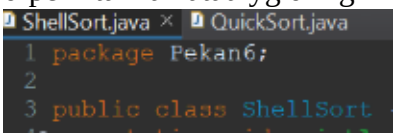
Tujuan dari praktikum ini adalah:

1. Membuat program Shell Sort.
2. Membuat program Quick Sort.
3. Membuat program Merge Sort.

C. LANGKAH KERJA

a. Program Shell Sort

1. Buat package dan class baru dan namakan sesuai dengan yang diperintahkan atau yg diinginkan.



2. Buat sintaks untuk mencetak seluruh elemen array dalam satu baris, dengan setiap elemen dipisahkan oleh spasi.

```
static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
        System.out.println();
```

3. Buat sintaks untuk pengurutan menggunakan algoritma shell sort.

```
} /*function to sort arr using shellSort*/
int sort(int arr[]) {
    int n = arr.length; //start with a big gap, then reduce the gap
    for (int gap = n / 2; gap > 0; gap /= 2) {
    for (int i = gap; i < n; i += 1) {
        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;
    }
```

4. Buat return 0; untuk menandakan bahwa metode tersebut telah selesai.

```
        }
 return 0;
 }
```

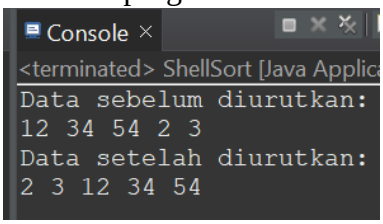
5. Inisialisasi isi array.

```
public static void main(String[] args) {
    int arr[] = {12, 34, 54, 2, 3};
```

6. Buat sintaks untuk mengurutkan data menggunakan algoritma shell dan buat sintaks untuk menampilkan output sebelum dan sesudah diurutkan.

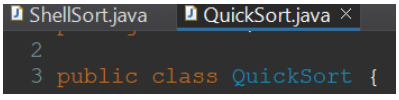
```
System.out.println("Data sebelum diurutkan: ");
printArray(arr);
ShellSort ob = new ShellSort();
ob.sort(arr);
System.out.println("Data setelah diurutkan: ");
printArray(arr);
```

7. Jalankan program.



b. Program Quick Sort

1. Buat class baru dan namakan sesuai yang diinginkan.



2. Buat sintaks baru untuk menukar nilai antara dua elemen dalam sebuah array.

```
static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;}
```

3. Buat algoritma partisi dalam algoritma Quick Sort, di mana pivot dipilih dan indeks pembagi diinisialisasi.

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

4. Buat sintaks iterasi untuk elemen array, kemudian lakukan pengecekan dan pertukaran elemen.

```
for (int j = low; j <= high - 1; j++) {
    //if current element is smaller than the pivot
    if (arr[j] < pivot) {
        //increment index of smaller element
        i++;
        swap(arr, i, j);
    }}
swap(arr, i + 1, high);
```

5. Kemudian, buat sintaks return untuk mengembalikan indeks pivot yang baru diposisikan.

```
return (i + 1);}
```

6. Buat algoritma Quick Sort secara rekursif membagi array menjadi subarray yang lebih kecil, mempartisi dan mengurutkan setiap subarray, dan menggabungkan hasilnya untuk mendapatkan array yang diurutkan secara keseluruhan.

```
static 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);
```

7. Buat metode printArr untuk dengan mudah mencetak seluruh elemen dalam array dalam satu baris dengan setiap elemen dipisahkan oleh spasi.

```
public static void printArr(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();}
```

8. Insialisasi isi array.

```
public static void main(String[] args) {
    int[] arr = {10, 7, 8, 9, 1, 5};
    int N = arr.length;
```

9. Buat sintaks untuk mengurutkan data menggunakan algoritma Quick Sort dan buat sintaks untuk menampilkan output sebelum dan sesudah diurutkan.

```
System.out.print("Data sebelum diurutkan: ");
printArr(arr);
quickSort(arr, 0, N - 1);
System.out.print("Data terurut QuickSort: ");
printArr(arr);
```

10. Jalankan program.

```
Console ×
<terminated> QuickSort [Java Application] C:\Users\USER
Data sebelum diurutkan: 10 7 8 9 1 5
Data terurut QuickSort: 1 5 7 8 9 10
```

c. Program Merge Sort

1. Buat class baru dan namakan sesuai perintah atau yang diinginkan.

```
ShellSort.java  QuickSort.java
1 package Pekan6;
2
3 public class MergeSort {
```

2. Buat sintaks menemukan ukuran dua subarray yang akan digabungkan.

```
void merge(int arr[], int l, int m, int r) {
```

3. Buat temp array.

```
int n1 = m - l + 1;
int n2 = r - m; //create temp arrays
```

4. Salin data ke temp array.

```
int L[] = new int[n1];
int R[] = new int[n2]; //copy data to temp arrays
```

5. Buat sintaks untuk mengisi dua array sementara dengan nilai-nilai dari dua bagian terpisah dari array yang lebih besar.

```
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];
```

6. Buat indeks awal dari gabungan subarray array.

```
int i = 0, j = 0; //initial index of merged subarray array
int k = l;
```

7. Buat sintaks untuk menggabungkan dua bagian terurut dalam algoritma Merge Sort.

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

8. Salin elemen-elemen yang tersisa dari L[] jika ada.

```
}//copy remaining elements of L[] if any
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
```

9. Salin elemen-elemen yang tersisa dari R[] jika ada.

```
}//copy remaining elements of R[] if any
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
```

10. Buat sintaks untuk menemukan titik tengah.

```
void sort(int arr[], int l, int r) {
    if (l < r) {
        //find the middle point
        int m = (l + r) / 2;
```

11. Buat sintaks untuk mengurutkan bagian pertama dan kedua.

```
//sort first and second halves
sort(arr, l, m);
```

12. Buat sintaks untuk menggabungkan bagian yang diurutkan.

```
 sort(arr, m + 1, r);//merge the sorted halves
 merge(arr, l, m, r);
```

13. Buat fungsi utilitas untuk mencetak array berukuran n.

```
}// a utility function to print array of size n
static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
```

14. Insialisasi isi array.

```
public static void main(String[] args) {
    int arr[] = {12, 11, 13, 5, 6, 7};
```

15. Buat sintaks untuk mengurutkan data menggunakan algoritma Merge Sort dan buat sintaks untuk menampilkan output sebelum dan sesudah diurutkan.

```
System.out.println("Sebelum terurut: ");
printArray(arr);
MergeSort ob = new MergeSort();
ob.sort(arr, 0, arr.length - 1);
System.out.println("Sesudah terurut: ");
printArray(arr);
```

16. Jalankan program.

```
Console ×
<terminated> MergeSort [Java
Sebelum terurut:
12 11 13 5 6 7
Sesudah terurut:
5 6 7 11 12 13
```

D. KESIMPULAN

Pada praktikum ini, kita telah mempelajari dan mengimplementasikan tiga jenis algoritma sorting: Shell Sort, Quick Sort, dan Merge Sort. Masing-masing algoritma memiliki pendekatan dan karakteristik yang berbeda dalam mengurutkan data:

1. Shell Sort mempercepat proses pengurutan dengan membandingkan elemen-elemen yang berjauhan terlebih dahulu sebelum beralih ke elemen yang berdekatan, yang pada akhirnya menyerupai Insertion Sort saat jarak pengurutan menjadi 1.
2. Quick Sort menggunakan pendekatan divide-and-conquer dengan memilih pivot untuk mempartisi array ke dalam dua sub-array, yang kemudian diurutkan secara rekursif. Algoritma ini terkenal karena efisiensinya dalam rata-rata kasus dengan kompleksitas waktu O(n log n).
3. Merge Sort juga menggunakan pendekatan divide-and-conquer, di mana array dibagi menjadi dua bagian yang lebih kecil, diurutkan secara rekursif, dan kemudian digabungkan kembali. Merge Sort menjamin kompleksitas waktu O(n log n) dalam semua kasus, namun membutuhkan ruang tambahan untuk proses penggabungan.

Dengan memahami dan mengimplementasikan ketiga algoritma ini, kita dapat memilih algoritma yang paling sesuai untuk berbagai jenis data dan kebutuhan spesifik dalam pengurutan data.