Laporan Praktikum

Struktur Data



Disusun Oleh :

**MUHAMMAD FARREL GIOVANNI (2311533006)**

Dosen Pengampu : Dr. Wahyudi, S.T, M.T

Departemen Informatika

Fakultas Teknologi Informasi

Universitas Andalas

Tahun 2024

**Algoritma Sorting 2**

1. **Tujuan Praktikum**
2. Memahami algoritma Shell Sort pada struktur data.
3. Memahami algoritma Quick Sort pada struktur data.
4. Memahami algoritma Merge Sort pada struktur data.
5. **Pendahuluan**

Algoritma pengurutan (sorting) meletakkan elemen data kedalam kumpulan data urutan tertentu, proses pengurutan yang sebelumnya data disusun acak menjadi tersusun teratur menurut aturan tertentu. Pengurutan merupakan hal yang tidak bisa dipisahkan dari dunia komputer. Sekarang ini Google dikenal sebagai mesin pencari terbesar di dunia. Dalam hitungan detik dapat diperoleh informasi yang diinginkan. Adanya kebutuhan terhadap proses pengurutan memunculkan bermacam-macam metode pengurutan yang bertujuan untuk memperoleh metode pengurutan yang optimal.

Pengurutan data (sorting) didefinisikan sebagai proses untuk menyusun kembali himpunan objek dengan menggunakan aturan tertentu. Tujuannya adalah untuk mendapatkan kemudahan dalam pencarian dari suatu himpunan. Kelebihan suatu data yang terurut adalah mudah untuk dicek apabila ada data yang hilang.

Data terkadang berada dalam bentuk yang tidak berpola ataupun dengan pola tertentu yang diinginkan. Tidak ada algoritma terbaik untuk semua keadaan, kadang kala sebuah algortitma sangat efisien ketika jumlah datanya sedikit, namun kinerjanya menjadi berkurang ketika jumlah data ditambahkan atau meningkat. Meskipun memiliki kemampuan komputasi yang lebih tinggi, namun jika menggunakan algoritma yang kurang efisien, maka akan membutuhkan waktu lebih lama. Sehingga untuk memecahkan permasalahan diperlukan sebuah algoritma yang efektif dan efisien agar persoalan komputasi serta terbatasnya alokasi memori dapat diatasi.

Secara umum ada dua jenis pengurutan data yaitu: 1) Model urut naik (ascending) yang mengurutkan data dari yang mempunyai nilai terkecil sampai terbesar. 2) Model urut turun (descending) yang mengurutkan data dari yang mempunyai nilai terbesar sampai terkecil.

Algoritma pengurutan data yang sering ditemukan dalam literatur komputer antara lain bubble sort, selection sort, insertion sort, heap sort, shell sort, quick sort, merge sort, radix sort, dan tree sort. Semua algoritma pengurutan selalu melakukan operasi perbandingan data untuk menemukan posisi urutan yang tepat. Berdasarkan tempat penyimpanan data, sorting dibedakan antara external sort dan internal sort. External sort bila datanya berada dalam media external, atau external storage seperti hardisk. Internal sort bila datanya ada dalam internal storage atau memory komputer.

1. **Metode Praktikum**
2. **Shell Sort**

Cara kerja shell sort yaitu dengan cara membandingkan suatu data lain yang memiliki jarak tertentu sehingga membentuk sebuah sub-list. Langkah – langkah pengurutan algoritma shell sort sebagai berikut:

1. Untuk jarak (N/2)+1: Data pertama (i=0) dibandingkan dengan data jarak (N/2)+1. Apabila data pertama lebih besar dari data ke (N/2)+1 tersebut maka kedua data tersebut ditukar. Kemudian data kedua (i=1) dibandingkan dengan jarak yang sama yaitu (N/2)+1) = elemen ke-(i+N/2)+1. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-i selalu lebih kecil dari pada data ke-(i+N/2)+1.
2. Ulangi langkah-langkah diatas untuk jarak = (N/4)+1 kemudian lakukan perbandingan dan pengurutan sehingga semua data ke-i lebih kecil daripada data ke-(i+N/4)+1.
3. Ulangi langkah-langkah diatas untuk jarak =(N/8)+1 kemudian lakukan perbandingan dan pengurutan sehingga semua data ke-i lebih kecil daripada data ke-(i+N/8)+1
4. Demikian seterusnya samapi jarak yang digunakan adalah I atau data sudah terurut.

Berikut kelas yang menggunakan algoritma shell sort dalam pengurutan data :

public class ShellSort {

static void printArray(int arr[]) {

int n = arr.length;

for (int i=0; i<n; i++)

System.***out***.print(arr[i]+" ");

System.***out***.println();

}

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;

}

}

return 0;

}

public static void main(String[] args) {

int arr[] = {12,34,54,2,3};

System.***out***.print("Data sebeum diurutkan : ");

*printArray*(arr);

ShellSort ob = new ShellSort();

ob.sort(arr);

System.***out***.print("Data setelah diurutkan : ");

*printArray*(arr);

}

}

Output dari program tersebut menampilkan elemen-elemen array sebelum dan sesudah diurutkan menggunakan algoritma shell sort :

Data sebeum diurutkan : 12 34 54 2 3

Data setelah diurutkan : 2 3 12 34 54

1. **Quick Sort**

Algoritma Quick Sort berdasar pada pola divide-and-conquer. Pola divide-and-conquer mengimplementasikan konsep rekursi untuk menyelesaikan permasalahan. Permasalahan utama kemudian dipecah menjadi sub-masalah, kemudian solusi dari sub-masalah akan membimbing menuju solusi permasalahan utama. algoritma ini mengikuti langkah – langkah sebagai berikut :

1. Divide

Memilah rangkaian data menjadi dua sub-rangkaian A[p…q-1] dan A[q+1…r] dimana setiap elemen A[p…q-1] adalah kurang dari atau sama dengan A[q] dan setiap elemen pada A[q+1…r] adalah lebih besar atau sama dengan elemen pada A[q]. A[q] disebut sebagai elemen pivot. Perhitungan pada elemen q merupakan salah satu bagian dari prosedur pemisahan.

1. Conquer

Mengurutkan elemen pada sub-rangkaian secara rekursif

Pada algoritma quicksort, langkah ”kombinasi” tidak di lakukan karena telah terjadi pengurutan elemen – elemen pada sub-array.

Berikut kelas yang menggunakan algoritma quick sort dalam pengurutan data :

public class QuickSort {

static void swap(int[] arr, int i, int j) {

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = (low-1);

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);

return (i+1);

}

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);

}

}

public static void printArr (int[] arr) {

for (int i = 0; i< arr.length; i++) {

System.***out***.print(arr[i]+" ");

}

System.***out***.println();

}

public static void main(String[] args) {

int[] arr = {10,7,8,9,1,5};

int N=arr.length;

System.***out***.print("Data sebelum diurutkan : ");

*printArr*(arr);

*quickSort*(arr, 0, N-1);

System.***out***.print("Data terurut quicksort : ");

*printArr*(arr);

}

}

Output dari program tersebut menampilkan elemen-elemen array sebelum dan sesudah diurutkan menggunakan algoritma quick sort :

Data sebelum diurutkan : 10 7 8 9 1 5

Data terurut quicksort : 1 5 7 8 9 10

1. **Merge Sort**

Merge sort menggunakan pola divide and conquer. Dengan hal ini deskripsi dari algoritma dirumuskan dalam 3 langkah berpola divide-and-conquer. Berikut menjelaskan langkah kerja dari Merge sort :

1. Divide

Memilah elemen – elemen dari rangkaian data menjadi dua bagian.

1. Conquer

Conquer setiap bagian dengan memanggil prosedur merge sort secara rekursif

1. Kombinasi

Mengkombinasikan dua bagian tersebut secara rekursif untuk mendapatkan rangkaian data berurutan.

Proses rekursi berhenti jika mencapai elemen dasar. Hal ini terjadi bilamana bagian yang akan diurutkan menyisakan tepat satu elemen. Sisa pengurutan satu elemen tersebut menandakan bahwa bagian tersebut telah terurut sesuai rangkaian.

Berikut kelas yang menggunakan algoritma shell sort dalam pengurutan data :

public class MergeSort {

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

//find sizes of two subarrays to be merged

int n1=m-l+1;

int n2=r-m;

//create temp array

int L[]=new int[n1];

int R[]=new int[n2];

//copy data to temp array

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;

//initial index of merged subarray array

int k=l;

while (i<n1&&j<n2) {

if (L[i] <= R[j]) {

arr[k]=L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

//copy remaining elements of L[] if any

while (i<n1) {

arr[k] = L[i];

i++;

k++;

}

//copy remaining elements of R[] if any

while (j<n2) {

arr[k] = R[j];

j++;

k++;

}

}

void sort(int arr[], int l, int r) {

if (l < r) {

//find the middle point

int m = (l+r)/2;

//sort first and second halves

sort(arr,l,m);

sort(arr,m+1,r);

//merge sorted halves

merge(arr,l,m,r);

}

}

//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();

}

public static void main(String[] args) {

int arr[] = {12,11,13,5,6,7};

System.***out***.println("Sebelum terurut");

*printArray*(arr);

MergeSort ob = new MergeSort();

ob.sort(arr, 0, arr.length-1);

System.***out***.println("Sesudah terurut menggunakan merge sort");

*printArray*(arr);

}

}

Output dari program tersebut menampilkan elemen-elemen array sebelum dan sesudah diurutkan menggunakan algoritma merge sort :

Sebelum terurut

12 11 13 5 6 7

Sesudah terurut menggunakan merge sort

5 6 7 11 12 13

1. **Kesimpulan Praktikum**

Pengurutan data (sorting) didefinisikan sebagai proses untuk menyusun kembali himpunan objek dengan menggunakan aturan tertentu. Tujuannya adalah untuk mendapatkan kemudahan dalam pencarian dari suatu himpunan. Beberapa contoh algoritma sorting yang populer dalam literatur komputer diantaranya shell sort, quick sort, merge sort.

Algoritma *insertion sort* meengurutkan data dengan cara menyisipkan elemen data pada posisi yang tepat. Sedangkan algoritma *selection sort*, konsepnya adalah dengan memilih elemen minimum kemudian mempertukarkan elemen minimum dengan elemen pertama untuk urutan *ascending* dan elemen paling akhir untuk urutan *descending*. Selain itu, algoritma *Bubble sort* adalah algoritma pengurutan sederhana yang bekerja dengan cara membandingkan dan menukar elemen-elemen bersebelahan jika berada dalam urutan yang salah. Proses ini diulangi hingga tidak ada lagi elemen yang perlu ditukar.

Pemilihan algoritma pengurutan yang tepat tergantung pada berbagai faktor seperti ukuran data, kebutuhan stabilitas pengurutan, dan kompleksitas implementasi. Inilah mengapa banyak algoritma sorting dikembangkan, masing-masing dengan keunggulan dan kelemahan yang berbeda untuk memenuhi berbagai kebutuhan pengurutan data. Dengan algoritma yang baik dapat dihasilkan sebuah program yang efisien dari segi waktu dan hasil yang dicapai. Kompleksitas waktu atau kecepatan eksekusi dari algoritma sorting menjadi faktor penting dalam memilih algoritma yang tepat untuk aplikasi tertentu. Selain itu, sebuah algoritma juga menjadi sangat efisien ketika jumlah datanya sedikit, yang kinerjanya akan berkurang ketika jumlah data ditambahkan atau meningkat.