JOBSHEET 10

Shell Sort and Quick Sort

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Teknik Informatika	INF1.62.2014	2 x 50 Menit

TUJUAN PRAKTIKUM

1. Mahasiswa memahami konsep pengurutan Shell dan Quick dalam pemrograman C.
2. Mahasiswa memahami operasi yang ada dalam pencarian pengurutan Shell dan Quick.
3. Mahasiswa mampu mengimplementasikan pencarian struktur data pengurutan Shell dan Quick menggunakan pemrograman C dengan IDE.

HARDWARE & SOFTWARE

1. Personal Computer
2. DevC++ IDE

TEORI SINGKAT

1. Shell Sort

Shell sort adalah sebuah algoritma dengan pengurutan elemen pertama memiliki jarak interval dengan elemen yang akan dibandingkan. Pada prinsipnya sama dengan Insertion sort pada pokok bahasan sebelumnya.

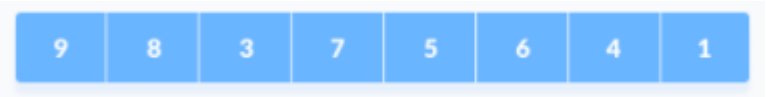
Dalam pengurutan Shell Sort, elemen/data diurutkan dengan interval yang terukur. Jarak interval diantara elemen semakin lama semakin mengecil berdasarkan berapa kali pengulangan pengurutan dilakukan. Kemampuan dari shell sort ini tergantung pada tipe dari tipe ururtan yang digunakan dalam sebuah array yang disediakan.

Beberapa macam formula urutan yang digunakan :

- Shell's original sequence: N/2 , N/4 , …, 1
- Knuth's increments: 1, 4, 13, …, (3k − 1) / 2
- Sedgewick's increments: 1, 8, 23, 77, 281, 1073, 4193, 16577...4j+1+ 3·2j+ 1.

Cara Kerja Shell Sort :

1. Urutkan array yang ada

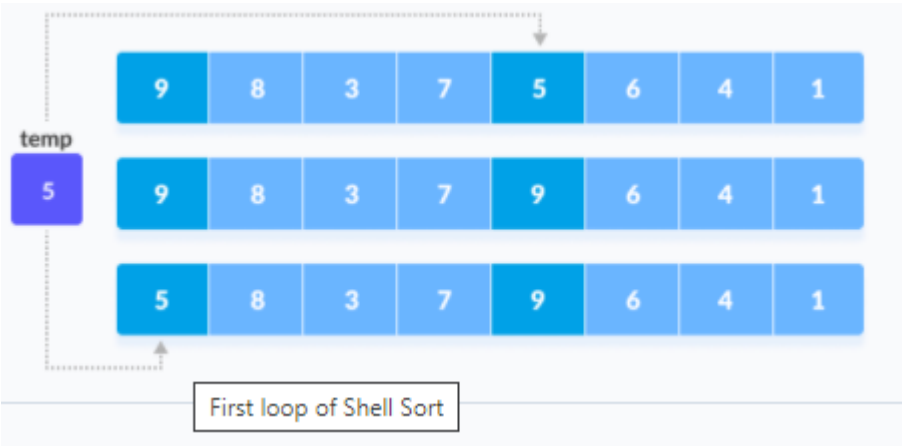


2. Menggunakan formula original shell (N/2, N/4, ...1) sebagai interval yang akan digunakan dalam algoritma.

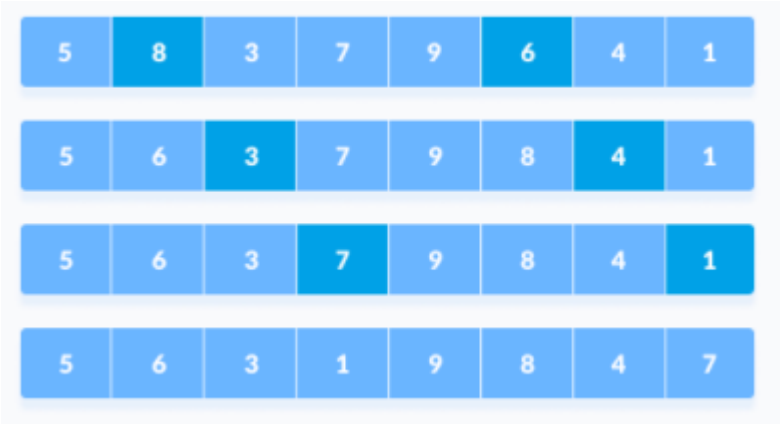
   Pada fase pertama, jika besar array N=8 maka jarak interval elemen menggunakan rumus N/2 = 4, akan dibandingkan dan ditukar jika belum sesuai urutan.

   a. Elemen indeks ke 0 dibandingkan dengan indeks elemen ke 4.

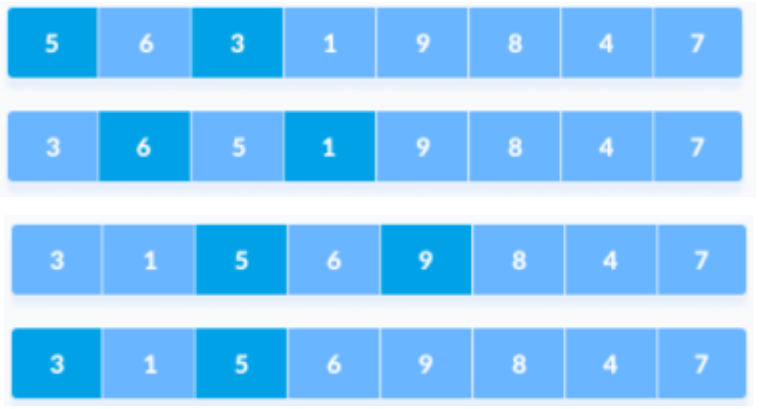
   b. Jika elemen ke 0 lebih besar daripada ke 4 lalu elemen ke 4 disimpan ke dalam variabel ***temp*** dan elemen ke 0 (elemen 0 nilai lebih besar) dipindahkan ke posisi elemen ke 4 dan elemen yang disimpan dalam variabel ***temp*** pindahkan ke posisi ke 0.



   Proses pengulangan pengurutan ini berlangsung untuk keseluran elemen yang ada

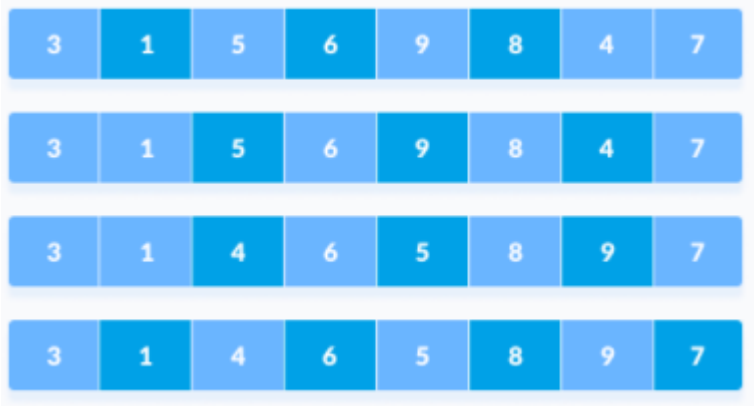


3. Pada pengulangan kedua rumus formula interval yang digunakan N/4 = 8/4 = 2

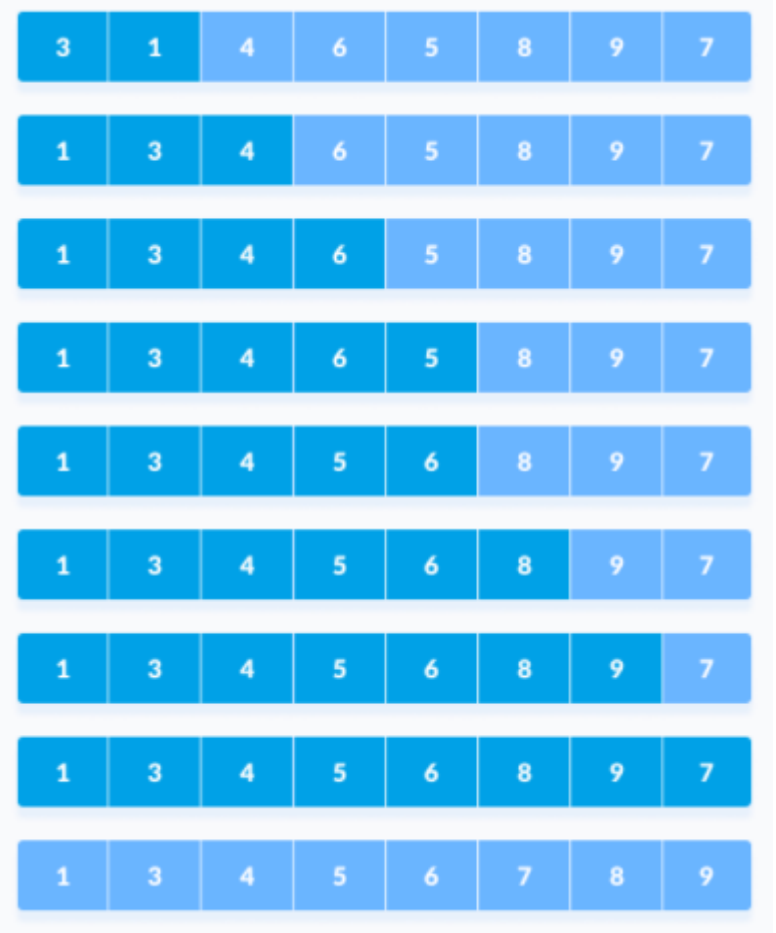


Elemen ke 2 dan ke 4 dibandingkan sesuai rumus interval = 2, elemen ke 0 dan 2 juga dibandingkan.

4. Proses yang sama berlaku untuk elemen yang tersisa.



5. Pada akhirnya samapai pada fase akhir di saat interval N/8 = 8/8 = 1, elemen dengan interval 1 diurutkan



Implementasi program Shell Sort

Shell sort digunakan saat:

- uClibc library menggunakan pengurutan ini.
- bzip2 compressor menggunakan algoritma ini juga.
- Pengurutan Insertion kurang baik saat elemen yang dekat berjauhan. Shell sort membantu dalam mengurangi jarak pertukaran elemen data sehingga terjadi pengurangan data yang akan di tukar.

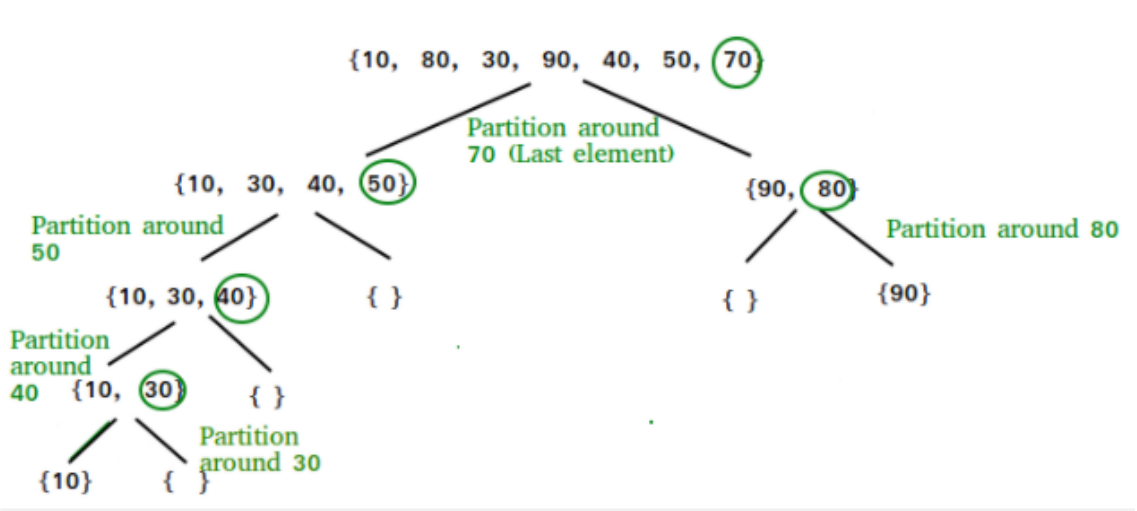
2. Quick Sort

Seperti halnya Merge sort, Quick sort menggunakan algoritma divide dan conquer. Algoritma ini mengambil sebuah elemen sebagai pivot dan memisahkan array yang ada disekitar pivot. Terdapat banyak versi dari Quick sort yang mengambil pivot dengan cara yang berbeda-beda. Diantaranya :

1. Selalu mengambil elemen pertama sebagai pivot.
2. Selalu mengambil eleme terakhir sebagai pivot
3. Elemen acak sebagai pivot
4. Elemen yang di tengah sebagai pivot

Kunci dari proses quick sort adalah partisi(). Target dari partisi adalah array sebuah array dan sebuah elemen x dari array sebagai pivot, letakkan x pada posisi yang benar dalam array yang terurut dan

letakkan semua elemen yang lebih kecil (lebih kecil dari x) sebelum elemen x, dan tempatkan semua elemen yang lebih besar dari x setelah x. Semua proses ini seharusnya seleai dalam waktu yang sama.



PERCOBAAN

1. **Shell sort 1**

```
#include <stdio.h>

void shellSort(int array[], int n){
  for (int gap = n/2; gap > 0; gap /= 2){
    for (int i = gap; i < n; i += 1) {
      int temp = array[i];
      int j;
      for (j = i; j >= gap && array[j - gap] > temp; j -= gap){
        array[j] = array[j - gap];
      }
      array[j] = temp;
    }
  }
}
void printArray(int array[], int size){
  for(int i=0; i<size; ++i){
     printf("%d  ", array[i]);
  }
  printf("\n");
}
int main(){
  int data[]={9, 8, 3, 7, 5, 6, 4, 1};
  int size=sizeof(data) / sizeof(data[0]);
  shellSort(data, size);
  printf("Sorted array: \n");
  printArray(data, size);
}
```

2. **Shell Sort 2**

```
#include "stdio.h"
```

```
int main()
{
    int L[20],temp,i,j,n=6,m;
    printf("pengurutan berdasarkan Shell sort \nmasukkan %d
elements: \n",n);
    for(i=0;i<n;i++) {
        scanf("%d",&L[i]);
      }
    printf("\nsebelum sorting: ");
    for(i=0;i<n;i++) {
            printf("%d ",L[i]);
      }
    for(m = n/2;m>0;m/=2) {
    /*6 7 2 1 ===> 2 7 6 1, 2 1 6 7 // 1 2 6 7, 1 2 6 7, 1 2 6 7*/
       for(j=m;j<n;j++) {
           for(i=j-m;i>=0;i-=m) {
               if(L[i+m]>=L[i]) break;
               else{
                   temp = L[i];
                   L[i] = L[i+m];
                   L[i+m] = temp;
               }
           }
       }
    }
    printf("\nsetelah sorting: ");
    for(i=0;i<n;i++){printf("%d ",L[i]);}
    printf("\n");
}
```

3. **Shell sort 3**

```
#include<stdio.h>
void ShellSort(int a[], int n)
{
    int i, j, increment, tmp;
    for(increment = n/2; increment > 0; increment /= 2) {
        for(i = increment; i < n; i++) {
            tmp = a[i];
            for(j = i; j >= increment; j -= increment) {
                if(tmp < a[j-increment])
                    a[j] = a[j-increment];
                else
                    break;
            }
            a[j] = tmp;
        }
    }
}

int main()
{
    int i, n, a[10];
    printf("Masukan jumlah elemen : ");
    scanf("%d",&n);
    printf("Masukan %d bilangan : \n",n);
    for(i = 0; i < n; i++) {
        scanf("%d",&a[i]);
    }
    ShellSort(a,n);
```

```
      printf("Elemen yang telah diurutkan adalah:\n");
      for(i = 0; i < n; i++)
          printf("%d\n",a[i]);
      printf("\n");
      return 0;
  }
```

4. **Quick Sort 1**

```
/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int arr[], int low, int high) {
    int pivot = arr[high];    // pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j <= high- 1; j++) {
        // If current element is smaller than the pivot
        if (arr[j] < pivot) {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
/* Function to print an array */
void printArray(int arr[], int size) {
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: \n");
```

```
    printArray(arr, n);
    return 0;
}
```

5. **Quick Sort 2**

```
// Quick sort in C
#include <stdio.h>

void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

int partition(int array[], int low, int high) {

  int pivot = array[high];
  int i = (low - 1);

  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {
      i++;
      swap(&array[i], &array[j]);
    }
  }
  swap(&array[i + 1], &array[high]);
  return (i + 1);
}
void quickSort(int array[], int low, int high) {
  if (low < high) {
    int pi = partition(array, low, high);
    quickSort(array, low, pi - 1);
    quickSort(array, pi + 1, high);
  }
}

void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

int main() {
  int data[] = {8, 7, 2, 1, 0, 9, 6};
  int n = sizeof(data) / sizeof(data[0]);
  quickSort(data, 0, n - 1);
  printf("Sorted array in ascending order: \n");
  printArray(data, n);
}
```

6. **Quick Sort 3**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
#define MaxStack 10
```

```
int Data[MAX]; // = {12,35,9,11,3,17,23,15,31,20};

// Prosedur menukar data
void Tukar (int *a, int *b)
{
 int temp;
 temp = *a;
 *a = *b;
 *b = temp;
}
// Prosedur pengurutan metode Quick Sort
void QuickSortNonRekursif()
{
      struct tump
      {
  int Kiri;
  int Kanan;
      }Tumpukan[MaxStack];
      int i, j, L, R, x, ujung = 1;
      Tumpukan[1].Kiri = 0;
      Tumpukan[1].Kanan = MAX-1;
      while (ujung!=0){
  L = Tumpukan[ujung].Kiri;
  R = Tumpukan[ujung].Kanan;
  ujung--;
  while(R > L)
      {
   i = L;
   j = R;
   x = Data[(L+R)/2];
   while(i <= j){
    while(Data[i] < x)
     i++;
    while(x < Data[j])
     j--;
    if(i <= j){
     Tukar(&Data[i], &Data[j]);
     i++;

     j--;
       }
     }
   if(L < i){
    ujung++;
    Tumpukan[ujung].Kiri = i;
    Tumpukan[ujung].Kanan = R;
     }
   R = j;
    }
 }
}
int main()
{
   int i;
   srand(0);

// Membangkitkan bilangan acak
   printf("DATA SEBELUM TERURUT");
   for(i=0; i<MAX; i++) {
```

```
    Data[i] = (int) rand()/1000+1;
    printf("\nData ke %d : %d ", i, Data[i]);
    }
    QuickSortNonRekursif();
    // Data setelah terurut
    printf("\nDATA SETELAH TERURUT");
    for(i=0; i<MAX; i++){
       printf("\nData ke %d : %d ", i, Data[i]);
    }
}
```

TUGAS

1. Carilah contoh aplikasi yang mengimplementasikan shell sort dan quick sort serta jelaskan bagaimana aplikasi tersebut bekerja sesuai dengan prinsip kedua metode sorting tersebut!

DAFTAR PUSTAKA

1. Kernighan, Brian W, & Ritchie, Dennis M. 1988. The Ansi C Programming Language Second Edition, Prentice-Hall.
2. Cipta Ramadhani. 2015. Dasar Algoritma & Struktur Data. Yogyakarta: ANDI.