Tugas Job Sheet 10 Shell and Quick Sort_Praktikum Struktur data
Muhamad Akbar Fauzan_23343075

A. Contoh aplikasi dalam Shell Sort

```
#include <stdio.h>

// Fungsi untuk melakukan Shell Sort
void shellSort(int arr[], int n) {
   // Mulai dengan selisih besar, kemudian kurangi selisihnya
   for (int selisih = n / 2; selisih > 0; selisih /= 2) {
      // Lakukan insertion sort dengan selisih ini.
      // Bagian pertama arr[0..selisih-1] sudah dalam urutan
      // tambahkan satu elemen lagi hingga seluruh array terurut dengan selisih ini
      for (int i = selisih; i < n; i += 1) {
         // tambahkan arr[i] ke elemen yang sudah diurutkan dengan selisih ini
         // simpan arr[i] di temp dan buat lubang di posisi i
         int temp = arr[i];
         // geser elemen yang sudah diurutkan dengan selisih ke atas sampai lokasi yang benar
untuk arr[i] ditemukan
         int j;
         for (j = i; j >= selisih && arr[j - selisih] > temp; j -= selisih)
            arr[j] = arr[j - selisih];
         // letakkan temp (arr[i] asli) di lokasi yang benar
         arr[j] = temp;
      }
   }
}

int main() {
   int arr[] = {12, 34, 54, 2, 3};
   int n = sizeof(arr)/sizeof(arr[0]);

   printf("Array sebelum sorting: \n");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");

   shellSort(arr, n);

   printf("Array setelah sorting: \n");
   for (int i = 0; i < n; i++)
```

```
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

```
Array sebelum sorting:
12 34 54 2 3
Array setelah sorting:
2 3 12 34 54

--------------------------------
Process exited after 0.04967 seconds with return value 0
Press any key to continue . . .
```

Penjelasan :
Shell Sort: Metode ini membagi array menjadi beberapa sub-array yang lebih kecil untuk dilakukan insertion sort. Prinsipnya adalah dengan melakukan insertion sort pada sub-array yang lebih kecil dan kemudian secara bertahap mengurangi ukuran gap (jarak antar elemen yang dibandingkan) sampai gap menjadi 1, yang pada akhirnya menjadikan array hampir terurut. Hal ini membantu mempercepat proses insertion sort karena elemen-elemen yang lebih kecil telah "maju" ke bagian awal array.

B. Contoh aplikasi dalam Quick Sort

```
#include <stdio.h>

// Fungsi untuk menukar nilai dua variabel
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

// Fungsi untuk membagi array dan mengembalikan index pivot
int partition(int arr[], int rendah, int tinggi) {
    int pivot = arr[tinggi];    // pivot
    int i = (rendah - 1);  // Indeks elemen yang lebih kecil

    for (int j = rendah; j <= tinggi - 1; j++) {
        // Jika elemen saat ini lebih kecil dari pivot
        if (arr[j] < pivot) {
            i++;    // tingkatkan indeks elemen yang lebih kecil
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[tinggi]);
    return (i + 1);
```

```
}

// Fungsi untuk Quick Sort
void quickSort(int arr[], int rendah, int tinggi) {
   if (rendah < tinggi) {
      // pi adalah index pembagian, arr[p] saat ini ada di tempat yang benar
      int pi = partition(arr, rendah, tinggi);

      // Urutkan secara terpisah elemen sebelum pembagian dan setelah pembagian
      quickSort(arr, rendah, pi - 1);
      quickSort(arr, pi + 1, tinggi);
   }
}

int main() {
   int arr[] = {10, 7, 8, 9, 1, 5};
   int n = sizeof(arr) / sizeof(arr[0]);

   printf("Array sebelum sorting: \n");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");

   quickSort(arr, 0, n - 1);

   printf("Array setelah sorting: \n");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
   return 0;
}
```

```
Array sebelum sorting:
10 7 8 9 1 5
Array setelah sorting:
1 5 7 8 9 10

--------------------------------
Process exited after 0.05489 seconds with return value 0
Press any key to continue . . .
```

Penjelasan :

Quick Sort: Metode ini menggunakan pendekatan divide and conquer, dimana array dibagi menjadi dua bagian berdasarkan pivot, dan kemudian kedua bagian tersebut diurutkan secara rekursif. Prinsip utamanya adalah dengan memilih elemen pivot, kemudian memindahkan elemen-elemen yang lebih kecil dari pivot ke sebelah kiri pivot, dan elemen-elemen yang lebih

besar ke sebelah kanan pivot. Setelah itu, rekursif memanggil quick sort untuk kedua bagian array yang tersisa. Hal ini dilakukan secara berulang sampai seluruh array terurut. Kecepatan quick sort tergantung pada pemilihan pivot yang efisien, sehingga dalam implementasi yang baik, pivot dipilih agar membagi array menjadi dua bagian yang seimbang.