SOAL SORTING ARRAY

Mata Kuliah	:	Struktur Data
Dosen Pengampu	:	Andi Moch Januriana, ST., M.Kom

NAME : Mustopa

NIM : 3337220023

KELAS : C

Program Studi Informatika

Fakultas Teknik

Universitas Sultan Ageng Tirtayasa

Soal

Carilah 3 metode sorting lainnya dan tuliskan dalam paper beserta source code, cara dan analisis dan tiap-tiap metode sorting yang ada!

Buatlah semua procedure-procedure yang ada di atas dalam program utuh!

1. EXCHANGE SORT

Akan Melakukan perbandingan dengan elemen di sebelahnya apabila nilai nya tidak sesuai maka kedua elemen tersebut akan bertukar posisi

Algoritma ini hamper sama dengan buble sort

Perbedaan :

Buble sort akan membandingkan sampai ahir elemen

Sedangkan exchange sort akan akan berhenti apabila nilai sudah terurut

Contoh code exchange sort :

```cpp
#include <iostream>
using namespace std;
void exchange_sort(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        // terakhir i elemen sudah terurut, tidak perlu dibandingkan lagi
        for (int j = 0; j < n-i-1; j++) {
            // jika elemen ke-j lebih besar dari elemen ke-(j+1), tukar posisinya
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }

        }

    }
}
int main() {
    int arr[] = {5, 2, 7, 1, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

    exchange_sort(arr, n);

    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output

```
1 2 5 7 9


...Program finished with exit code 0
Press ENTER to exit console.
```

2. HEAP SORT

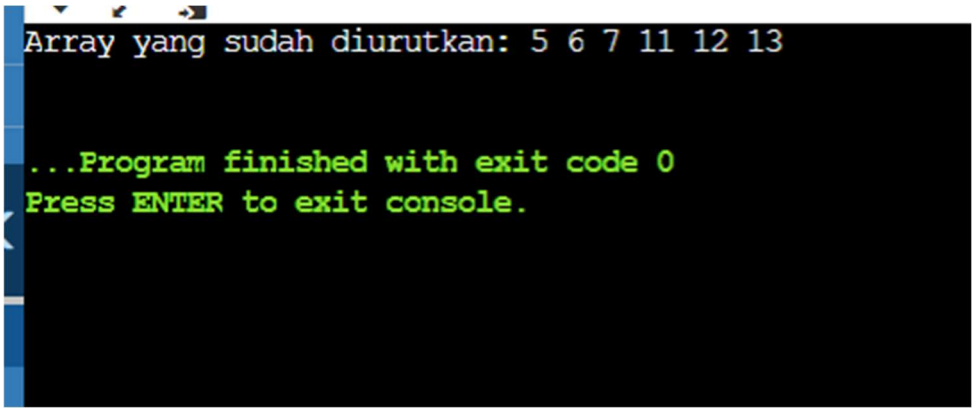
Akan melakukan perbandingan antara dua buah elamen dan menempatkan nilai terbesar ke elemen ahir . Proses ini diulangi sampai semua elemen telah diekstraksi, menghasilkan array yang terurut.

Contoh Code :

```
using namespace std;

void heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest]) {
        largest = l;
    }

    if (r < n && arr[r] > arr[largest]) {
        largest = r;
    }

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
```

```
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Array yang sudah diurutkan: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT



Penjelasan singkat:

- Baris pertama (**#include <iostream>**) adalah header file yang diperlukan untuk menggunakan input/output stream di C++.
- Baris kedua (**using namespace std;**) menghilangkan kebutuhan untuk mengetikkan **std::** sebelum menggunakan setiap fungsi di namespace **std**.
- Fungsi **heapify** digunakan untuk menjaga sifat heap dari sebuah array.
- Fungsi **heapSort** melakukan pengurutan array menggunakan algoritma heap sort.
- Fungsi **main** dimulai pada baris ke-22 dan diakhiri pada baris ke-34.
- Array **arr** dideklarasikan pada baris ke-24, dan ukuran array **n** dihitung menggunakan **sizeof(arr) / sizeof(arr[0])**.
- Baris ke-26 melakukan pemanggilan fungsi **heapSort** untuk mengurutkan array **arr**.
- Baris ke-28 hingga 33 digunakan untuk menampilkan array yang sudah diurutkan menggunakan fungsi **cout**.
- Baris ke-35 adalah akhir fungsi **main** dan mengembalikan nilai 0 untuk menandakan bahwa program berjalan dengan sukses.

3. TREE SORT

Tree sort adalah algoritma pengurutan yang menggunakan struktur data tree untuk menyimpan elemen yang akan diurutkan. Berikut adalah contoh kode program Tree sort:

CONTOH CODE

```cpp
#include <iostream>

using namespace std;

struct Node {
    int key;
    struct Node *left, *right;
};

struct Node* newNode(int item) {
    struct Node* temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

struct Node* insert(struct Node* node, int key) {
    if (node == NULL) return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}
```

```cpp
void storeSorted(struct Node* root, int arr[], int& i) {
    if (root != NULL) {
        storeSorted(root->left, arr, i);
        arr[i++] = root->key;
        storeSorted(root->right, arr, i);
    }
}

void treeSort(int arr[], int n) {
    struct Node* root = NULL;

    for (int i = 0; i < n; i++) {
        root = insert(root, arr[i]);
    }

    int i = 0;
    storeSorted(root, arr, i);
}

int main() {
    int arr[] = {5, 4, 7, 2, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    treeSort(arr, n);

    cout << "Array yang sudah diurutkan: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
```

```cpp
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT

```
Array yang sudah diurutkan: 2 4 5 7 11
```