**PERTEMUAN 13 LATIHAN SOAL**

**ALGORITMA DAN STRUKTUR DATA**



**Nama : Khulika Malkan**

**NIM : 2311110057**

**Kelas : S1SD04-B**

**PROGRAM STUDI SAINS DATA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

**2023**

1. Jelaskan secara detail konsep heap dan bagaimana cara kerjanya dalam struktur data. Jelaskan pula perbedaan antara heap min dan heap max

Jawab :

* Heap adalah struktur data berbasis pohon biner yang memenuhi sifat heap. Sifat heap ini mendefinisikan hubungan antara nilai-nilai dalam pohon.
* Heap Maks: Nilai pada setiap node lebih besar atau sama dengan nilai pada semua node turunannya. Nilai terbesar selalu berada di akar pohon. Sedangkan,
* Heap Min: Nilai pada setiap node lebih kecil atau sama dengan nilai pada semua node turunannya. Nilai terkecil selalu berada di akar pohon.
* Cara Kerja Heap: Heap diimplementasikan dengan cara memanipulasi pohon biner untuk menjaga sifat heap terpenuhi. Operasi dasar pada heap meliputi:
* Insert: Menambahkan elemen baru ke dalam heap.
* Extract: Menghapus elemen dari heap, biasanya elemen dengan nilai terbesar (heap maks) atau terkecil (heap min).
* Decrease Key: Menurunkan nilai suatu elemen dalam heap.
* Increase Key: Menaikkan nilai suatu elemen dalam heap.

Operasi-operasi ini dapat dilakukan dengan kompleksitas waktu O(log n), di mana n adalah jumlah elemen dalam heap. Hal ini membuat heap menjadi struktur data yang sangat efisien untuk berbagai aplikasi.

1. Jelaskan bagaimana cara mengimplementasikan heap menggunakan array. Jelaskan pula kelebihan dan kekurangan implementasi heap menggunakan array

Jawab :

* Langkah-langkah meng**implementasi**kan heap menggunakan array :

1. Representasi Pohon:

* Akar pohon terletak pada indeks 0.
* Anak kiri dari node pada indeks i terletak pada indeks 2i + 1.
* Anak kanan dari node pada indeks i terletak pada indeks 2i + 2.
* Indeks terakhir array adalah n-1, di mana n adalah jumlah elemen dalam heap.

1. Operasi Heap:

* Insert: Menambahkan elemen baru ke akhir array kemudian lakukan bubble-up untuk menjaga sifat heap terpenuhi.
* Extract: Menukar elemen akar dengan elemen terakhir array dan hapus elemen terakhir array. Kemudian lakukan bubble-down untuk menjaga sifat heap terpenuhi.
* Decrease Key: Menemukan elemen yang ingin diturunkan nilainya. Kemudian lakukan bubble-down untuk memindahkan elemen ke posisi yang sesuai.
* Increase Key: Temukan elemen yang ingin dinaikkan nilainya. Dan lakukan bubble-up untuk memindahkan elemen ke posisi yang sesuai.

1. **Kelebihan** Implementasi Heap Menggunakan Array

* Sederhana: Implementasi heap menggunakan array mudah dipahami dan diimplementasikan.
* Efisiensi Memori: Heap hanya membutuhkan ruang memori O(n), di mana n adalah jumlah elemen dalam heap.
* Cache-friendly: Akses elemen heap berdekatan dalam memori, yang meningkatkan kinerja cache.

1. **Kekurangan** Implementasi Heap Menggunakan Array

* Memboroskan ruang: Heap membutuhkan ruang memori yang sama meskipun elemennya tidak terisi penuh.
* Operasi Sisipkan dan Hapus Lambat: Operasi sisipkan dan hapus membutuhkan waktu O(log n), yang lebih lambat dibandingkan dengan struktur data lain seperti AVL tree.
* Sulit untuk dimodifikasi: Modifikasi struktur heap secara langsung dapat merusak sifat heap dan membutuhkan operasi tambahan untuk memperbaikinya.

1. Jelaskan bagaimana cara mengimplementasikan heap menggunakan pohon biner. Jelaskan pula kelebihan dan kekurangan implementasi heap menggunakan pohon biner

Jawab :

## Langkah-langkah meng**implementasi**kan heap menggunakan pohon biner:

## **Representasi Pohon:**

* Setiap node dalam pohon biner mewakili elemen dalam heap.
* Sifat heap harus dijaga dalam struktur pohon:
  + **Heap Maks:** Nilai pada setiap node **lebih besar** atau **sama** dengan nilai pada semua node turunannya.
  + **Heap Min:** Nilai pada setiap node **lebih kecil** atau **sama** dengan nilai pada semua node turunannya.

1. **Operasi Heap:**

* **Insert:**
  + Sisipkan elemen baru ke dalam pohon biner sebagai node baru.
  + Lakukan bubble-up untuk menjaga sifat heap terpenuhi.
* **Extract:**
  + Hapus elemen dari node akar.
  + Ganti node akar dengan salah satu node anaknya (biasanya node dengan nilai terbesar/terkecil).
  + Lakukan bubble-down untuk menjaga sifat heap terpenuhi.
* **Decrease Key:**
  + Temukan node yang ingin diturunkan nilainya.
  + Lakukan bubble-down untuk memindahkan node ke posisi yang sesuai.
* **Increase Key:**
  + Temukan node yang ingin dinaikkan nilainya.
  + Lakukan bubble-up untuk memindahkan node ke posisi yang sesuai.

1. **Kelebihan:**

* **Lebih fleksibel:** Struktur pohon biner dapat dimodifikasi dengan mudah untuk memenuhi kebutuhan aplikasi yang spesifik.
* **Lebih efisien untuk operasi tertentu:** Operasi seperti find dan merge dapat dilakukan dengan lebih efisien pada pohon biner dibandingkan dengan implementasi array.
* **Memungkinkan visualisasi:** Struktur pohon biner dapat divisualisasikan dengan mudah untuk membantu memahami cara kerja heap.

1. **Kekurangan:**

* **Lebih kompleks:** Implementasi heap menggunakan pohon biner lebih kompleks dibandingkan dengan implementasi array.
* **Membutuhkan lebih banyak memori:** Struktur pohon biner membutuhkan lebih banyak memori dibandingkan dengan implementasi array, terutama untuk pohon biner yang tidak seimbang.
* **Kinerja cache lebih rendah:** Akses elemen heap pada pohon biner mungkin tidak berdekatan dalam memori, yang dapat menurunkan kinerja cache.

1. Jelaskan konsep rekursif dan berikan contoh penerapannya dalam algoritma.

Jawab :

**Konsep Rekursif:**

Rekursif adalah teknik pemrograman di mana sebuah fungsi memanggil dirinya sendiri untuk menyelesaikan suatu masalah. Hal ini memungkinkan untuk memecahkan masalah yang kompleks menjadi sub-masalah yang lebih kecil dan serupa.

Ciri-ciri fungsi rekursif:

1. Kasus dasar: Sebuah kondisi yang menghentikan proses rekursi dan mengembalikan nilai secara langsung.
2. Memanggil diri sendiri: Fungsi rekursif harus memanggil dirinya sendiri dengan argumen yang berbeda.
3. Menghasilkan solusi: Setiap pemanggilan rekursif harus menghasilkan solusi parsial dari masalah, dan solusi-solusi parsial ini harus digabungkan untuk menghasilkan solusi akhir.

Kelebihan rekursif:

* Sederhana: Rekursif dapat membuat kode lebih mudah dibaca dan dipahami untuk beberapa masalah.
* Elegan: Rekursif dapat menghasilkan solusi yang lebih elegan dan ringkas dibandingkan dengan iterasi.

Kekurangan rekursif:

* Konsumsi memori: Rekursif dapat menggunakan memori yang lebih banyak, terutama untuk masalah yang kompleks.
* Kemungkinan stack overflow: Jika rekursi tidak dikontrol dengan baik, dapat terjadi stack overflow, di mana memori stack tidak cukup untuk menyimpan semua pemanggilan rekursif.

**Contoh Penerapan Rekursif:**

1. **Faktorial:**

* Kode program

int factorial(int n) {

if (n == 0) {

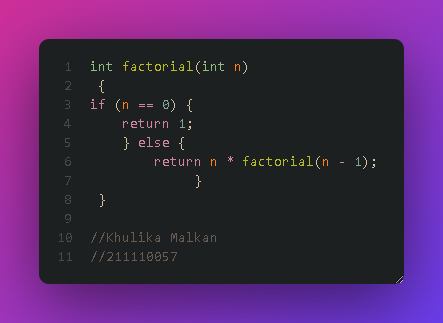
return 1;

} else {

return n \* factorial(n - 1);

}

}



**Langkah-langkah atau Algoritma:**

1. **Kasus Dasar (n = 0):**
   * Jika n sama dengan 0, maka faktorialnya adalah 1. Hal ini karena faktorial 0 didefinisikan sebagai 1.
   * Fungsi mengembalikan nilai 1 dan keluar dari proses rekursi.
2. **Kasus Rekursif (n > 0):**
   * Jika n lebih besar dari 0, maka faktorialnya dapat dihitung dengan mengalikan n dengan faktorial dari n-1.
   * Fungsi melakukan panggilan rekursif ke dirinya sendiri dengan meneruskan nilai n-1 sebagai argumen.
   * Nilai faktorial yang dikembalikan dari panggilan rekursif (factorial(n-1)) dikalikan dengan n dan dikembalikan sebagai hasil faktorial n.

**Penjelasan:**

* Fungsi pertama kali memeriksa apakah n sama dengan 0. Jika ya, fungsi langsung mengembalikan 1 sebagai faktorial 0.
* Jika n lebih besar dari 0, fungsi melakukan panggilan rekursif ke dirinya sendiri dengan meneruskan nilai n-1.
  + Panggilan rekursif ini menghitung faktorial dari n-1.
* Hasil dari panggilan rekursif (factorial(n-1)) dikalikan dengan n dan dikembalikan sebagai faktorial n.

**Contoh yang di tulis dalam program:**

* Menghitung faktorial 5:
  1. Fungsi dipanggil dengan n = 5.
  2. Karena n > 0, kasus rekursif dijalankan.
  3. Fungsi melakukan panggilan rekursif ke dirinya sendiri dengan n-1 = 4.
  4. Panggilan rekursif ini menghitung faktorial 4 (24).
  5. Hasil rekursif (24) dikalikan dengan n (5) dan dikembalikan sebagai faktorial 5 (120).

1. **Mencari Elemen dalam Daftar:**

* Kode program

bool findElement(vector<int> list, int element) {

if (list.empty()) {

return false;

} else if (list[0] == element) {

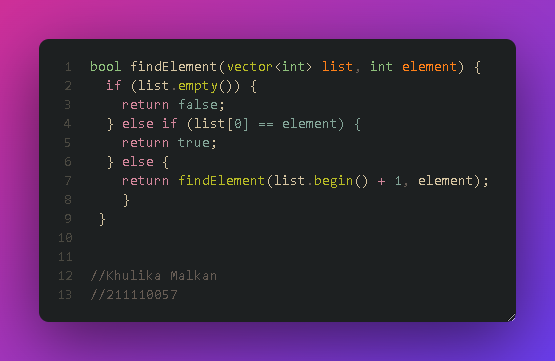
return true;

} else {

return findElement(list.begin() + 1, element);

}

}



* Langkah-langkah atau Algoritma :

1. **Kasus Dasar (Daftar Kosong):**
   * Jika daftar (list) kosong (list.empty()), elemen tidak mungkin ada. Fungsi mengembalikan false untuk menunjukkan hal ini.
2. **Kecocokan pada Elemen Pertama:**
   * Jika elemen pertama dari daftar (list[0]) sama dengan elemen target (element), pencarian berhasil. Fungsi mengembalikan true untuk menandakan hal ini.
3. **Panggilan Rekursif (Sisa Pencarian):**
   * Jika tidak ada satupun kasus dasar yang terpenuhi, fungsi perlu mencari elemen pada elemen selanjutnya dalam daftar.
   * Fungsi melakukan panggilan rekursif ke dirinya sendiri (findElement(list.begin() + 1, element)). Panggilan ini meneruskan daftar baru yang tidak termasuk elemen pertama (diperoleh menggunakan list.begin() + 1). Panggilan rekursif pada dasarnya mencari elemen target pada sub-himpunan daftar yang tersisa.
4. **Nilai Kembali dari Panggilan Rekursif:**
   * Pernyataan return dalam panggilan rekursif menentukan apakah elemen target ditemukan di sisa daftar. Nilai true atau false ini kemudian disebarkan kembali melalui rantai rekursif ke panggilan fungsi awal.

**Penjelasan:**

* Fungsi terlebih dahulu memeriksa apakah daftar kosong. Jika ya, tidak ada gunanya mencari lebih lanjut, jadi fungsi mengembalikan false.
* Jika daftar tidak kosong, fungsi memeriksa apakah elemen pertama dari daftar sama dengan elemen target. Jika ya, pencarian berhasil, dan fungsi mengembalikan true.
* Jika tidak ada satupun kasus dasar yang benar, fungsi memanggil dirinya sendiri secara rekursif, meneruskan daftar baru yang tidak termasuk elemen pertama. Proses ini berlanjut hingga daftar menjadi kosong (mengarah ke kasus dasar pertama) atau elemen target ditemukan.
* Panggilan rekursif pada akhirnya menentukan apakah elemen target ada dalam daftar, dan nilai true atau false terakhir dikembalikan ke pemanggil.

1. Berikan contoh algoritma yang menggunakan heap dan rekursif untuk menyelesaikan suatu masalah.

Jawab :

* **Langkah- langkah manusia (Algoritma):**

1. **Membangun Heap:**
   * Buat fungsi heapify yang menerima array, indeks awal, dan indeks akhir sebagai argumen.
   * Hitung indeks child dan bandingkan nilai elemen dengan child-nya.
   * Jika nilai elemen lebih kecil dari child, tukar dengan child.
   * Lakukan rekursi untuk child yang lebih besar.
   * Ulangi proses ini dari indeks awal ke indeks akhir untuk membangun heap.
2. **Pengurutan:**
   * Buat fungsi heapSort yang menerima array sebagai argumen.
   * Panggil heapify untuk seluruh array untuk membangun heap awal.
   * Ulangi langkah-langkah berikut hingga semua elemen terurut:
     + Tukar elemen pertama (akar) dengan elemen terakhir array.
     + Kurangi indeks akhir (ukuran array - 1).
     + Panggil heapify untuk elemen pertama hingga indeks akhir yang baru untuk memperbaiki heap.
3. **Fungsi Utama:**
   * Buat fungsi main untuk mengeksekusi program.
   * Deklarasikan dan inisialisasi array yang ingin diurutkan.
   * Panggil heapSort untuk mengurutkan array.
   * Cetak array yang sudah terurut.

* **Langkah-langkah Pembuatan Kode:**

1. **Membuat File dan Menulis Header:**
   * Buat file C++ baru (misalnya heap\_sort\_recursive.cpp).
   * Sertakan header C++ standar (iostream dan vector) di awal file.
   * Deklarasikan namespace std untuk memudahkan akses ke fungsi-fungsi standar.
2. **Menulis Fungsi heapify:**
   * Definisikan fungsi heapify dengan parameter array, start, dan end.
   * Hitung indeks child (child = root \* 2 + 1).
   * Periksa apakah child kedua (child + 1) ada dan nilainya lebih kecil dari child pertama.
   * Jika child kedua ada dan lebih kecil, atur child ke indeks child kedua.
   * Bandingkan nilai elemen pada indeks root dengan nilai pada indeks child.
   * Jika nilai elemen lebih kecil dari child, tukar nilai elemen dengan child.
   * Lakukan rekursi dengan memanggil heapify untuk child yang lebih besar.
   * Ulangi proses ini dari indeks start ke end untuk membangun heap.
3. **Menulis Fungsi heapSort:**
   * Definisikan fungsi heapSort dengan parameter array.
   * Hitung panjang array (n = array.size()).
   * Panggil heapify untuk seluruh array (heapify(array, 0, n - 1)) untuk membangun heap awal.
   * Ulangi langkah-langkah berikut hingga semua elemen terurut:
     + Tukar elemen pertama (akar) dengan elemen terakhir array (swap(array[0], array[n - 1])).
     + Kurangi indeks akhir (end -= 1).
     + Panggil heapify untuk elemen pertama hingga indeks akhir yang baru (heapify(array, 0, end)) untuk memperbaiki heap.
4. **Menulis Fungsi main:**
   * Definisikan fungsi main.
   * Deklarasikan dan inisialisasi array yang ingin diurutkan (misalnya vector<int> array = {7, 6, 5, 4, 3, 2, 1, 0};).
   * Panggil heapSort untuk mengurutkan array (heapSort(array)).
   * Gunakan loop for untuk mencetak array yang sudah terurut.
   * Kembalikan nilai 0 dari fungsi main.
5. **Kompilasi dan Eksekusi:**
   * Simpan file C++.
   * Gunakan compiler C++ untuk mengkompilasi file (misalnya g++ heap\_sort\_recursive.cpp -o heap\_sort\_recursive).
   * Jalankan program yang dihasilkan (misalnya ./heap\_sort\_recursive).

* **Kode program**

#include <iostream>

#include <vector>

using namespace std;

void heapify(vector<int>& array, int start, int end) {

int root = start;

while (root \* 2 + 1 <= end) {

int child = root \* 2 + 1;

if (child + 1 <= end && array[child] < array[child + 1]) {

child++;

}

if (child <= end && array[root] < array[child]) {

swap(array[root], array[child]);

root = child;

} else {

return;

}

}

}

void heapSort(vector<int>& array) {

int n = array.size();

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(array, i, n - 1);

}

for (int i = n - 1; i > 0; i--) {

swap(array[0], array[i]);

heapify(array, 0, i - 1);

}

}

int main() {

vector<int> array = {7, 6, 5, 4, 3, 2, 1, 0};

heapSort(array);

for (int i = 0; i < array.size(); i++) {

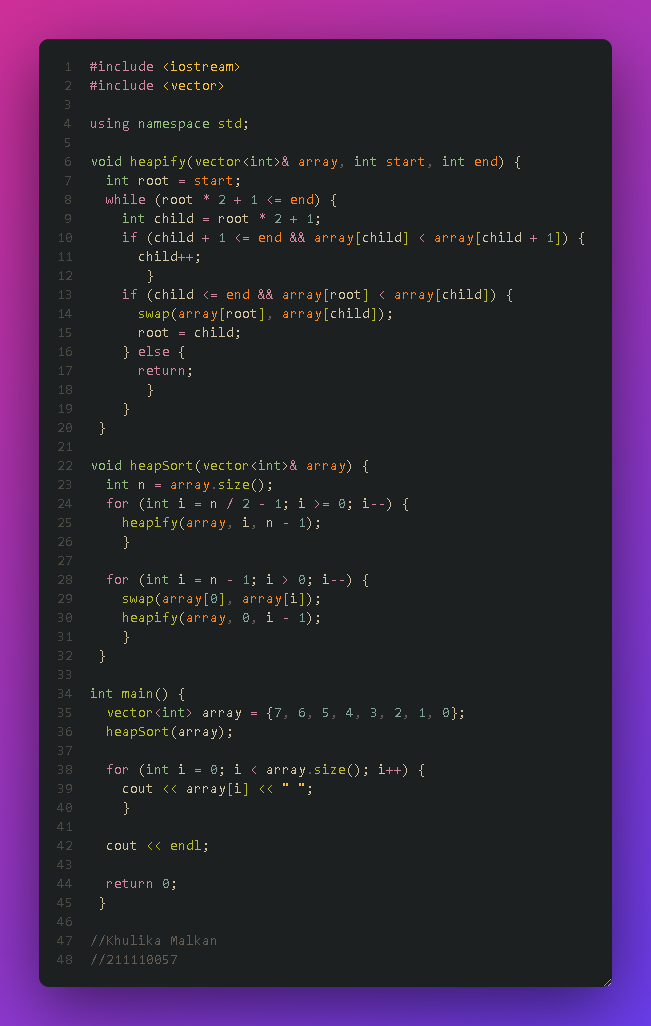
cout << array[i] << " ";

}

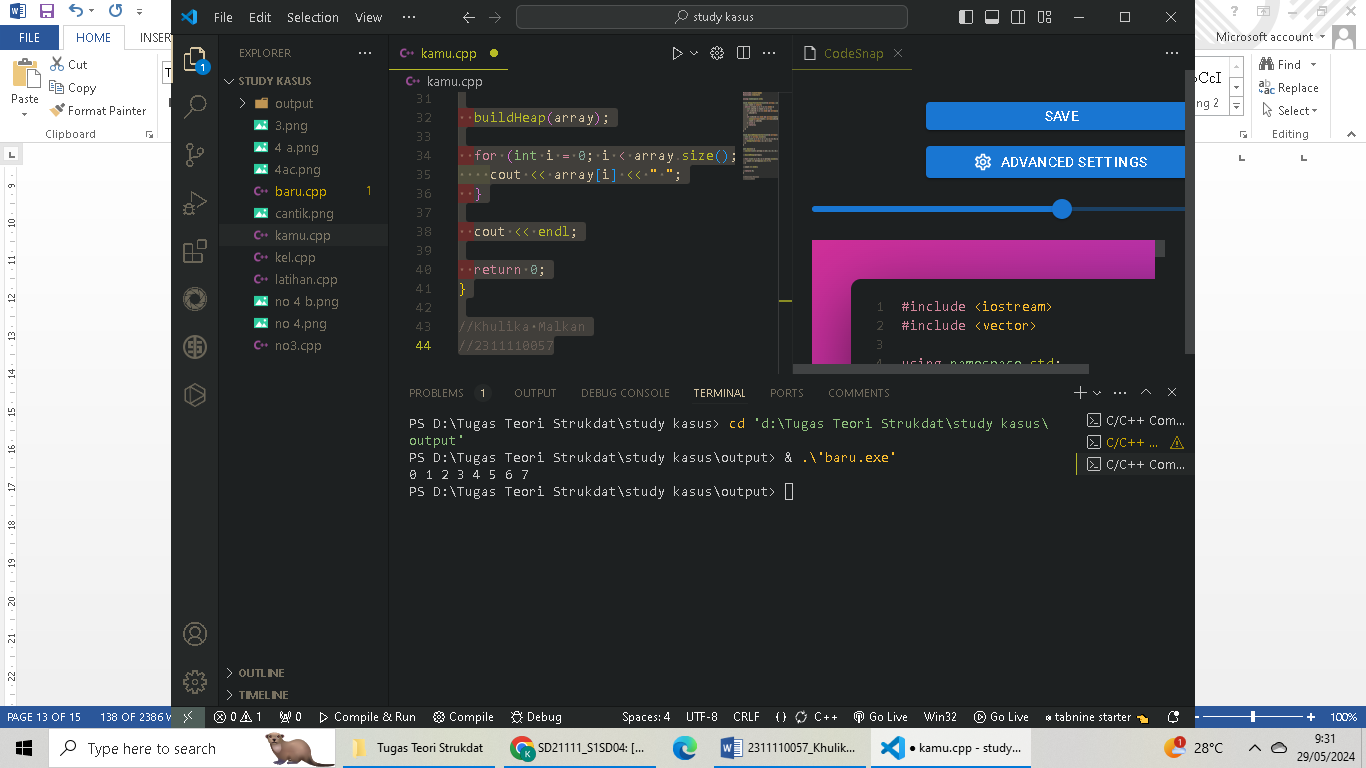
cout << endl;

return 0;

}



* **Output :**



* **Penjelasan:**
* **Fungsi heapify:**
  + Mirip dengan versi Python, fungsi ini membangun heap dari array dengan cara:
    - Memulai dari elemen tengah dan bergerak ke bawah ("shift down") untuk memastikan bahwa setiap elemen lebih besar dari atau sama dengan anak-anaknya.
    - Proses ini diulang untuk semua elemen pada level yang lebih tinggi.
* **Fungsi heapSort:**
  + Mirip dengan versi Python, fungsi ini mengurutkan array dengan cara:
    - Mengambil elemen terkecil dari heap (yang berada di akar) dan menukarnya dengan elemen terakhir dalam array.
    - Memanggil heapify untuk memperbaiki heap setelah penukaran.
    - Mengulangi proses ini sampai semua elemen dalam array terurut.
* **Fungsi main:**
  + Membuat vector array dan memasukkan beberapa elemen.
  + Memanggil heapSort untuk mengurutkan array.
  + Mencetak array yang sudah terurut.

1. Buatlah program C++ untuk membangun heap dari array bilangan bulat. Program harus dapat menangani array dengan berbagai ukuran dan elemen.

**Input**: Array bilangan bulat

**Output**: Array yang sama dengan input, tetapi telah tersusun dalam struktur heap

**Contoh**:

Input: {5, 2, 4, 6, 1, 3}

Output: {6, 5, 4, 2, 1, 3}

**Petunjuk**: Anda dapat menggunakan algoritma heapify untuk membangun heap dari array. Algoritma heapify bekerja dengan cara menelusuri array dari bawah ke atas dan memastikan bahwa setiap simpul memiliki sifat heap.

Jawab :

* **Contoh membangun heap dari array bilangan bulat:**

#include <iostream>

#include <vector>

using namespace std;

void heapify(vector<int>& array, int start, int end) {

int root = start;

while (root \* 2 + 1 <= end) {

int child = root \* 2 + 1;

if (child + 1 <= end && array[child] < array[child + 1]) {

child++;

}

if (child <= end && array[root] < array[child]) {

swap(array[root], array[child]);

root = child;

} else {

return;

}

}

}

void buildHeap(vector<int>& array) {

int n = array.size();

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(array, i, n - 1);

}

}

int main() {

vector<int> array = {5, 2, 4, 6, 1, 3};

buildHeap(array);

for (int i = 0; i < array.size(); i++) {

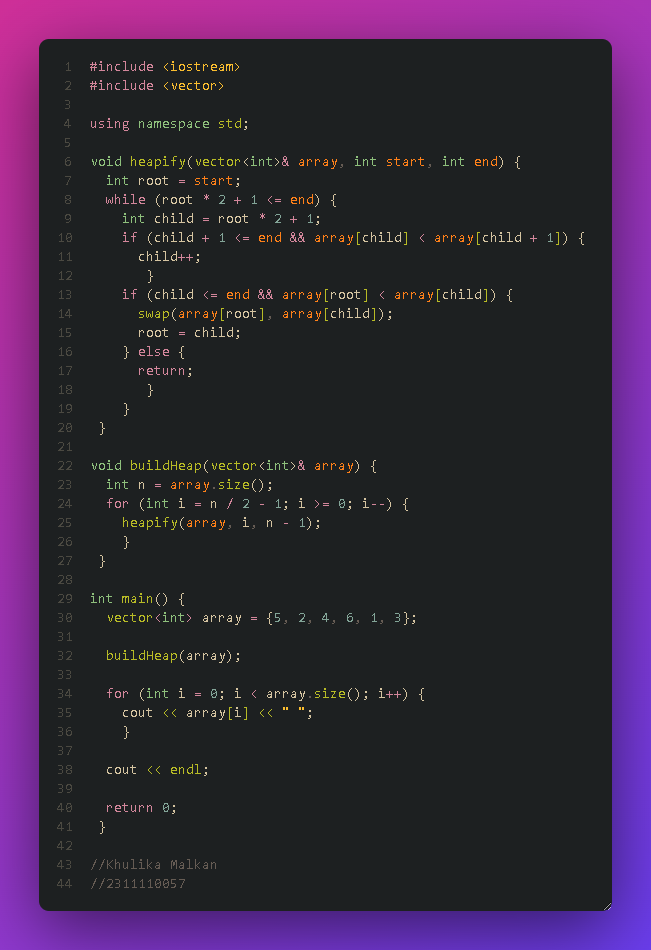
cout << array[i] << " ";

}

cout << endl;

return 0;

}



Atau bisa juga menggunakan kode program untuk input

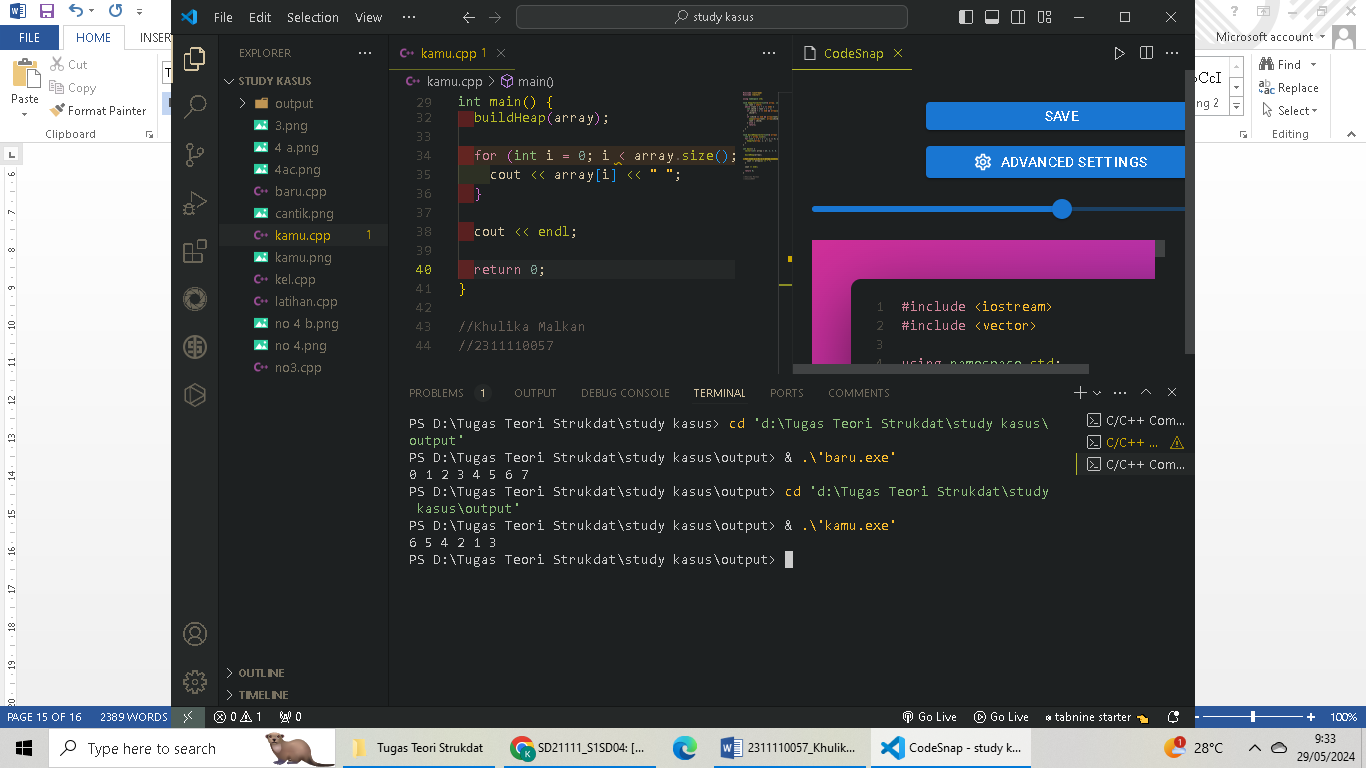


**Penjelasan:**

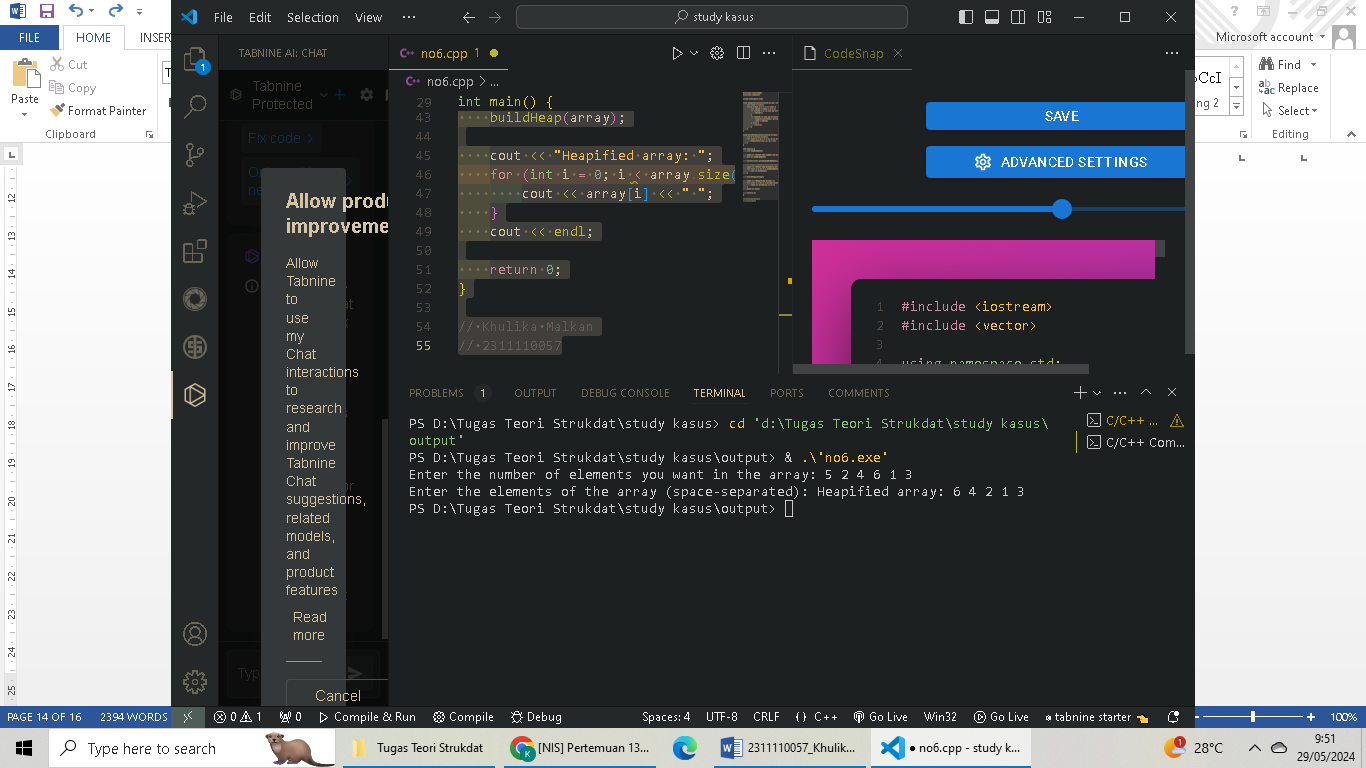
* **Fungsi heapify:**
  + Mirip dengan versi sebelumnya, fungsi ini membangun heap dari bagian array tertentu.
  + Memulai dari elemen tengah dan bergerak ke bawah ("shift down") untuk memastikan bahwa setiap elemen lebih besar dari atau sama dengan anak-anaknya.
  + Proses ini diulang untuk semua elemen pada level yang lebih tinggi.
* **Fungsi buildHeap:**
  + Memanggil heapify untuk seluruh array untuk membangun heap awal.
* **Fungsi main:**
  + Deklarasikan dan inisialisasi array yang ingin diurutkan.
  + Panggil buildHeap untuk membangun heap dari array.
  + Cetak array yang sudah terstruktur dalam heap.

**Output:**

6 5 4 2 1 3



Atau dengan fungsi input seperti gambar dibawah ini:



1. Buatlah program C++ untuk menghitung faktorial suatu bilangan menggunakan rekursif.

**Input**: Bilangan bulat positif

**Output**: Faktorial dari bilangan input

Contoh: Input: 5 Output: 120

**Petunjuk**: Faktorial dari suatu bilangan dapat dihitung dengan cara mengalikan bilangan tersebut dengan faktorial dari bilangan sebelumnya. Faktorial dari bilangan 1 didefinisikan sebagai 1.

Jawab :

* Kode program menghitung faktorial suatu bilangan menggunakan rekursif:

#include <iostream>

using namespace std;

int factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

int main() {

int number;

cout << "Masukkan bilangan: ";

cin >> number;

if (number < 0) {

cout << "Faktorial tidak didefinisikan untuk bilangan negatif." << endl;

} else {

int result = factorial(number);

cout << "Faktorial dari " << number << " adalah: " << result << endl;

}

return 0;

}



**Penjelasan:**

* **Fungsi factorial:**
  + Merupakan fungsi rekursif yang menerima bilangan bulat n sebagai argumen.
  + Memiliki kasus dasar: Jika n sama dengan 0, kembalikan nilai 1.
  + Jika n lebih besar dari 0, kembalikan hasil perkalian n dengan faktorial dari n-1.
* **Fungsi main:**
  + Meminta input bilangan dari pengguna.
  + Memeriksa apakah bilangan negatif. Jika ya, tampilkan pesan error.
  + Jika bilangan non-negatif, panggil factorial untuk menghitung faktorial.
  + Tampilkan hasil faktorial.

**Contoh Penggunaan:**

Masukkan bilangan: 5

Faktorial dari 5 adalah: 120

