**Assignment 3 – Complexity**

**Câu 1:**

O(4nlogn + 2n) = O(4nlogn) = O(nlogn)

O(2^10) = O(1)

O(2^logn) = O(n)

O(3n + 100logn) = O(3n) = O(n)

O(4n) = O(n)

O(2^n)

O(n^2 + 10n) = O(n^2)

O(n^3)

O(nlogn)  
Sắp xếp theo thứ tự tăng dần của Big O:

2^10, 2^logn, 3n +100logn, 4n, nlogn, 4nlogn + 2n, n^2 + 10n, n^3, 2^n

**Câu 2:**

Cách 1: Sử dụng vòng lặp

int power(int n) {

int result = 1;

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

result \*= 2;

}

return result;

}

Thuật toán có độ phức tạp là O(n) vì ta sử dụng vòng lặp n lần

Cách 2: Sử dụng hàm logarit

int power(int n) {

return std::pow(2, n);

}

Thuật toán có độ phức tạp là O(logn) vì ta dùng hàm pow(x, n) có sẵn từ thư viện

**Câu 3:**

class Queue {

private:

int size;

int\* a;

int head;

int rear;

public:

Queue(int n){

this->size = n;

a = new int[n];

head = 0;

rear = -1;

}

bool isFull(){

return rear == size-1;

}

bool isEmpty(){

return head > rear;

}

void print(){

for (int i = head; i <= rear; i++)

cout << a[i] << " ";

cout << endl;

}

void enqueue (int x){

if (isFull()) return;

a[++rear] = x;

}

// remove first

void dequeue(){

if (isEmpty()) return;

head++;

}

};

**Đánh giá độ phức tạp thuật toán:**

+ Hàm isFull(): O(1)

+ Hàm isEmpty(): O(1)

+ Hàm print(): O(n)

+ Hàm enqueue(): O(1)

+ Hàm dequeue(): O(1)

**Câu 4:**

// Class to represent a node in the queue

class Node {

public:

int data;

Node\* next;

Node(int data) {

this->data = data;

next = nullptr;

}

};

// Class to represent a queue using a linked list

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() { front = rear = nullptr;}

bool is\_empty() { front == nullptr; }

void enqueue(int element) {

Node\* tmp = new Node(element);

if (front == nullptr) {

front = rear = tmp;

} else {

rear->next = tmp;

rear = tmp;

}

}

void dequeue() {

if (is\_empty()) return;

Node\* tmp = front;

front = front->next;

if (front == nullptr) {

rear = nullptr;

}

delete (tmp);

}

int get\_front() {

if (is\_empty()) {

return -1;

}

return front->data;

}

int get\_rear() {

if (is\_empty()) {

return -1;

}

return rear->data;

}

void print() {

if (is\_empty()) {

std::cout << "Queue is empty" << std::endl;

return;

}

Node\* current = front;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

};

**Đánh giá độ phức tạp:**

+ Hàm is\_empty(): O(1)

+ Hàm enqueue(): O(1)

+ Hàm dequeue(): O(1)

+ Hàm get\_front(): O(1)

+ Hàm get\_rear(): O(1)

+ Hàm print(): O(n)

**Câu 5:**

class Stack{

private:

int\* a;

int size;

int top;

public:

Stack(int n){

this->size = n;

a = new int [n];

top = -1;

}

~Stack(){

delete[] a;

}

bool isFull(){

return top == size-1;

}

bool isEmpty(){

return top == -1;

}

void push(int x){

if (isFull()) return;

a[++top] = x;

}

void pop(){

if (isEmpty()) return;

top--;

}

void print(){

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

cout << a[i] << ' ';

cout << endl;

}

};

**Đánh giá độ phức tạp thuật toán**

+ Hàm isFull(): O(1)

+ Hàm isEmpty(): O(1)

+ Hàm push(): O(1)

+ Hàm pop(): O(1)

+ Hàm print(): O(n)

**Câu 6:**

// node in the stack

class Node {

public:

int data;

Node\* next;

Node(int data) {

this->data = data;

next = nullptr;

}

};

// stack using a linked list

class Stack {

private:

Node\* top;

public:

Stack() {

top = nullptr;

}

// Check if the stack is empty

bool isEmpty() {

return top == nullptr;

}

// Push an element

void push(int element) {

Node\* tmp=new Node(element);

tmp->next = top;

top = new\_node;

}

// Pop an element

int pop() {

if (isEmpty()) { return -1;}

int element = top->data;

Node\* temp = top;

top = top->next;

delete temp;

return element;

}

// Get the top element

int get\_top() {

if (isEmpty()) {

return -1;

}

return top->data;

}

// Print the stack

void print() {

if (isEmpty()) {

std::cout << "Stack is empty" << std::endl;

return;

}

Node\* current = top;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

};

**Đánh giá độ phức tạp thuật toán:**

+ Hàm isEmpty(): O(1)

+ Hàm push(): O(1)

+ Hàm pop(): O(1)

+ Hàm get\_top(): O(1)

+ Hàm print(): O(n)