**Homework week 3**

**Complexity analyses**

1. **Sort the following functions in the ascending order of Big O notation:**

- 210

- 2logn

- 3n+100logn

- 4n

- nlogn

- 4nlogn + 2n

- n2 + 10n

- n3

- 2n

1. **Given an integer number *n*, your task is to write two different algorithms in pseudo-codes to calculate 2n, and evaluate the complexity of the algorithms.**

**Algorithm 1:**

function power\_n\_of\_2(n):

ans ← 1;

for i ← 1 to n do:

ans ← ans \* 2;

return ans;

Time complexity: O(N)

**Algorithm 2:**

function power\_n\_of\_2(n):

if n = 0:

return 1;

res ← power\_n\_of\_2(n/2);

res ← res \* res;

if n % 2 = 0:

return res;

else:

return res\*2;

Time complexity: O(log2(N))

1. **Operations of queue data structure in pseudo-codes using an array**

const int max = 100; // Define the maximum size of the queue

struct Queue {

int count, front, rear;

int element[max];

void Init() {

count = 0;

front = 0;

rear = -1;

}

bool isEmpty() {

return (count == 0);

}

void enQueue(int x) {

if (rear == max - 1)

rear = 0;

else

rear = rear + 1;

element[rear] = x;

count = count + 1;

}

void deQueue() {

if (!isEmpty()) {

if (front == rear) {

front = 0;

rear = -1;

} else if (front == max - 1) {

front = 0;

} else {

front = front + 1;

}

count = count - 1;

}

}

int getFront() {

if (!isEmpty()) {

return element[front];

} else {

return -1;

}

}

};

Complexities:

- isEmpty(): O(1)

- enQueue(): O(1)

- deQueue(): O(1)

- getFront(): O(1)

1. **Queue data structure in pseudo-codes using a linked list, then evaluate the complexities of the operations.**

struct Node

{

Node \*nextNode;

int x;

};

struct Queue

{

Node \*head = NULL;

bool isEmpty()

{

return head == NULL;

}

void enQueue(int x)

{

Node \*newNode = new Node();

newNode->x = x;

if (head == NULL)

{

head = newNode;

return;

}

Node \*cur = head;

while (cur->nextNode != NULL)

{

cur = cur->nextNode;

}

cur->nextNode = newNode;

return;

}

void deQueue()

{

if (!isEmpty())

head = head->nextNode;

return;

}

int getFront()

{

return head->x;

}

};

Complexities:

- isEmpty(): O(1)

- enQueue(): O(1)

- deQueue(): O(1)

- getFront(): O(1)

1. **Write operations of stack data structure in pseudo-codes using an array, then evaluate the complexities of the operations.**

const int max = 100; // Define the maximum size of the stack

struct Stack {

int count;

int element[max];

void Init() {

count = 0;

}

bool isEmpty() {

return (count == 0);

}

void Push(int x) {

element[count] = x;

count++;

}

void Pop() {

if (!isEmpty()) {

count = count - 1;

}

}

int getTop() {

if (!isEmpty()) {

return element[count-1];

} else {

return -1;

}

}

};

Complexities:

- isEmpty(): O(1)

- Push(): O(1)

- Pop(): O(1)

- getTop(): O(1)

1. **Operations of stack data structure in pseudo-codes using a linked list, then evaluate the complexities of the operations.**

struct Node

{

Node \*preNode, \*nextNode;

int x;

Node()

{

preNode = nextNode = NULL;

x = 0;

}

};

struct Stack

{

Node \*head = NULL, \*tail = NULL;

bool isEmpty() {

return head == NULL;

}

void Push(int x)

{

Node \*newNode = new Node();

newNode->x = x;

if (head == NULL)

{

head = newNode;

tail = newNode;

return;

}

Node \*cur = head;

while (cur->nextNode != NULL)

{

cur = cur->nextNode;

}

newNode->preNode = cur;

cur->nextNode = newNode;

tail = newNode;

return;

}

void Pop()

{

if (tail == head)

{

tail = head = NULL;

return;

}

tail = tail->preNode;

tail->nextNode = NULL;

return;

}

int getTop()

{

return tail->x;

}

};

Complexities:

- isEmpty(): O(1)

- Push(): O(1)

- Pop(): O(1)

- getTop(): O(1)

1. **Write the pseudo codes and calculate the complexity of following functions on an array**

array elements[number\_of\_elements];

num = 0 //current number of elements

int element(p)

{

return element[p];

}

void insert(p, x)

{

num++;

for i from num to p:

a[i] = a[i-1];

a[p] = x;

}

void delete(p)

{

for i from p to n-1:

a[i] = a[i+1];

num--;

}

Complexities:

- element(p): O(1)

- insert(p, x): O(n)

- delete(p): O(n)

struct Node

int x;

Node \*nextNode;

};

int element(Node \*head, int p){

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

head = head->nextNode;

return head->x;

}

Node\* insert(Node\* head, int p, int x){

Node \*nnode = new Node();

nnode->x = x;

if (!p)

{

nnode->nextNode = head;

return nnode;

}

Node \*cur = head;

for (int i = 0; i < p-1; i++)

cur = cur->nextNode;

nnode->nextNode = cur->nextNode;

cur->nextNode = nnode;

return head;

}

Node\* delete(Node\* head, int p){

if (!p)

{

head = head->nextNode;

return head;

}

Node \*cur = head;

for (int i = 0; i < p-1; i++)

cur = cur->nextNode;

cur->nextNode = cur->nextNode->nextNode;

return head;

}

Complexities:

- element(p): O(n)

- insert(p, x): O(n)

- delete(p): O(n)