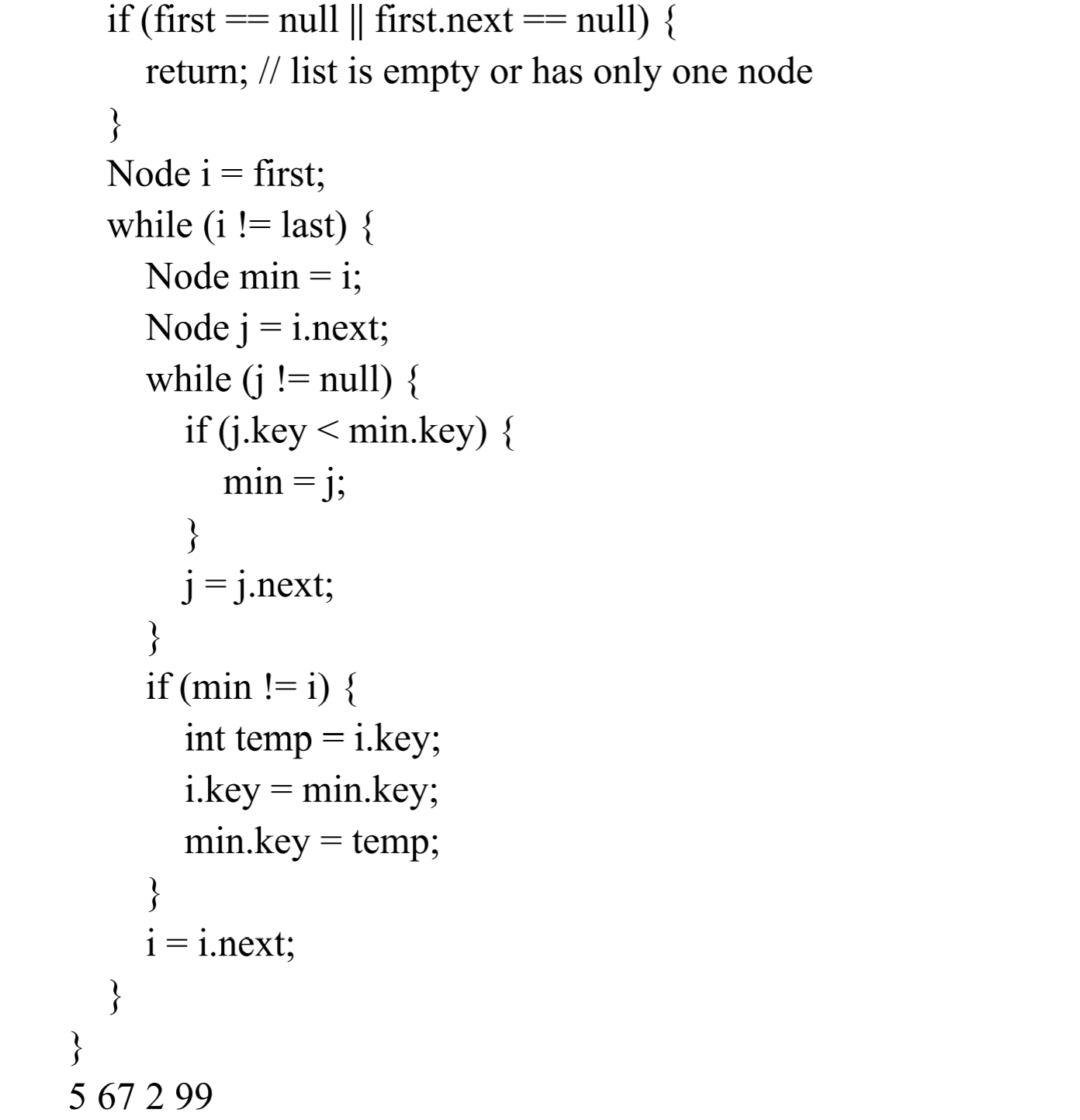
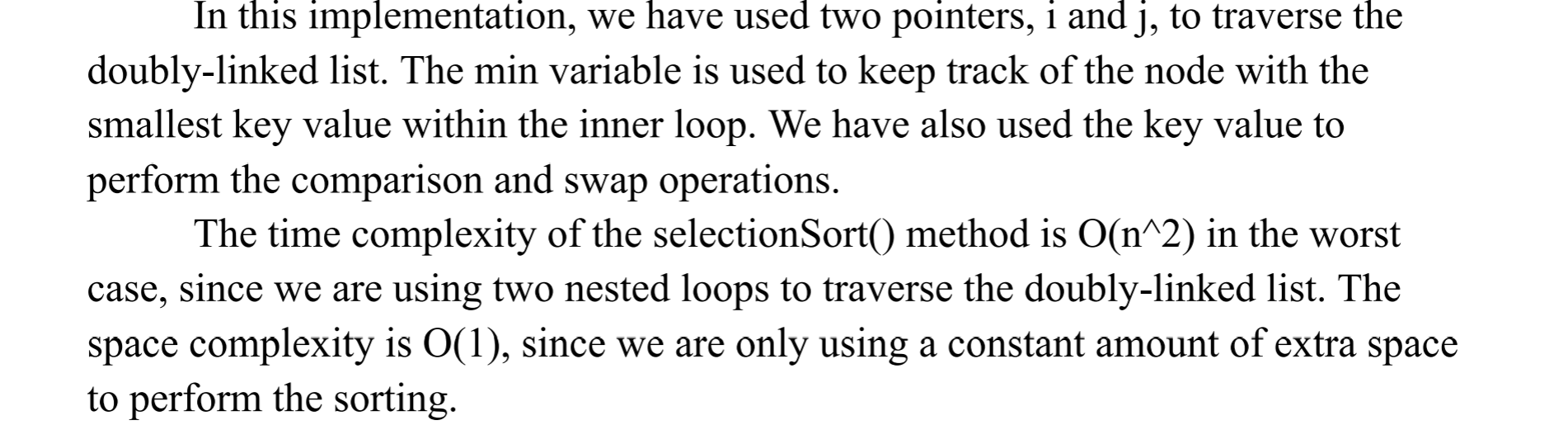
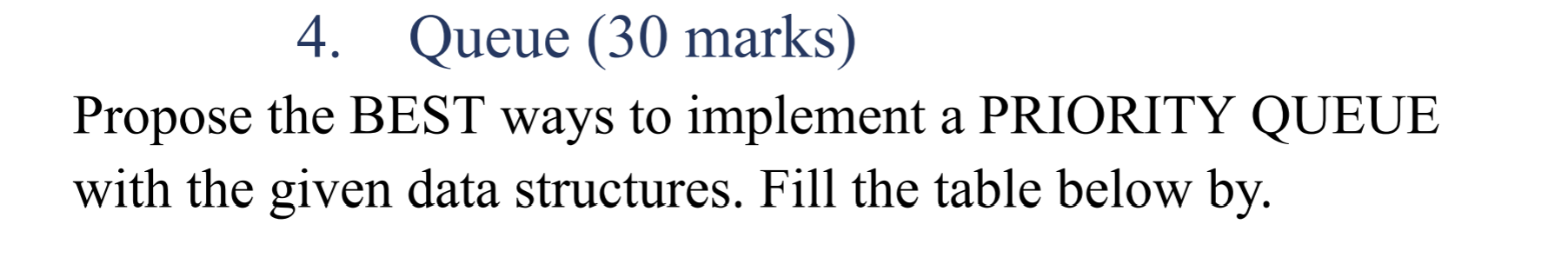
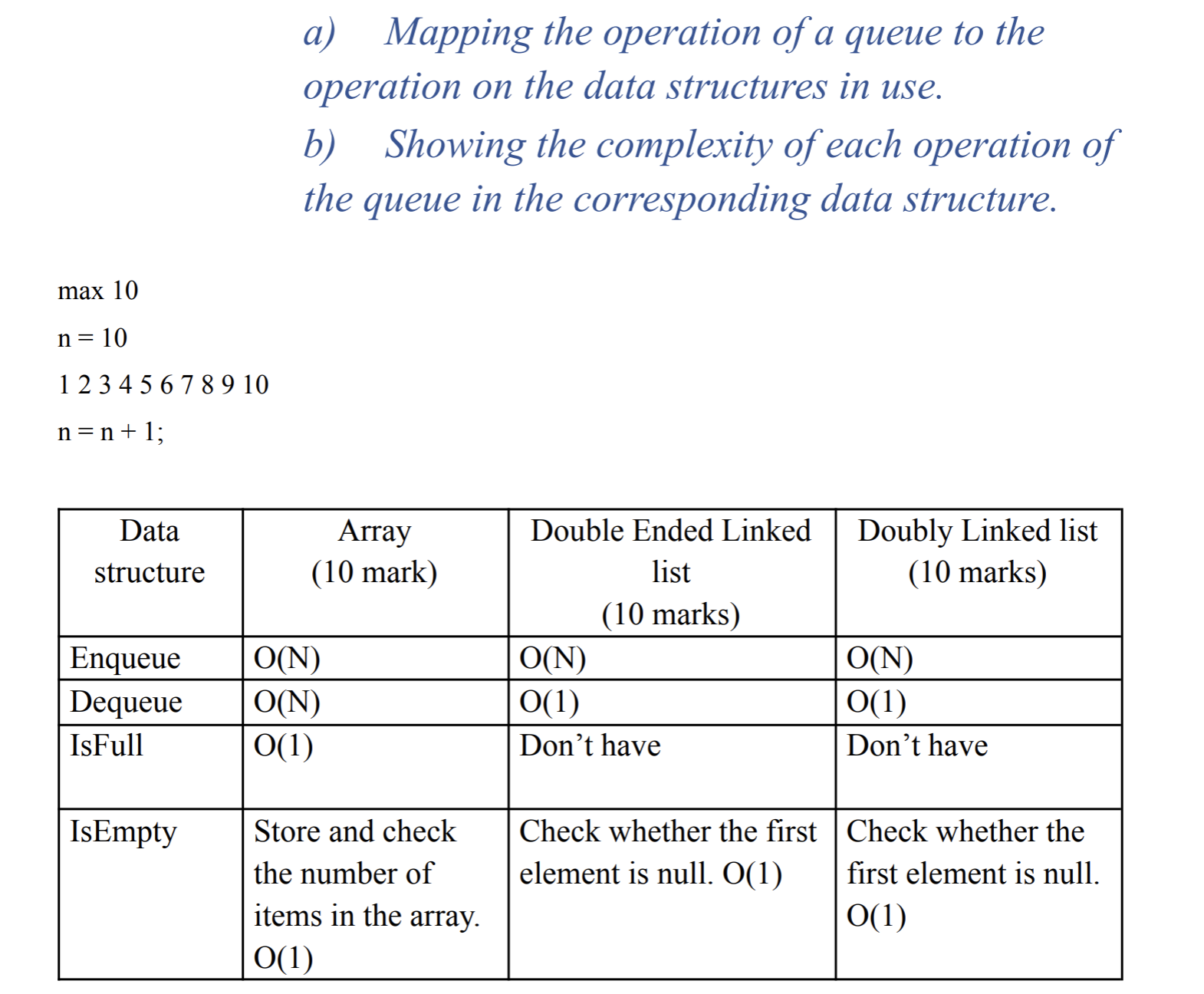
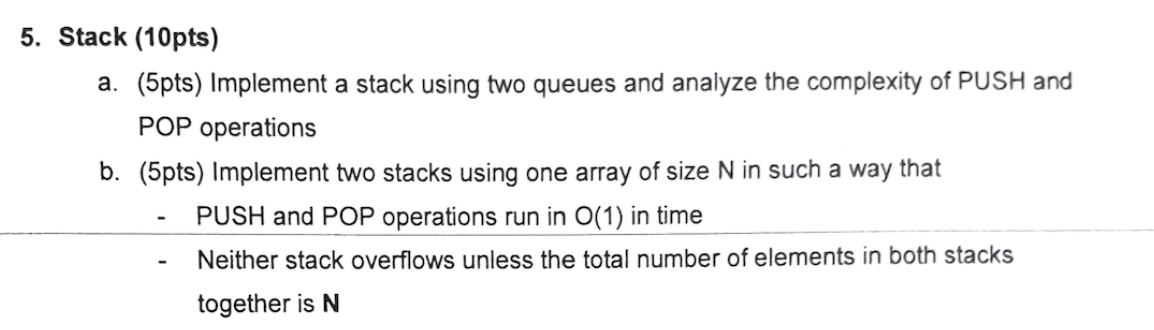


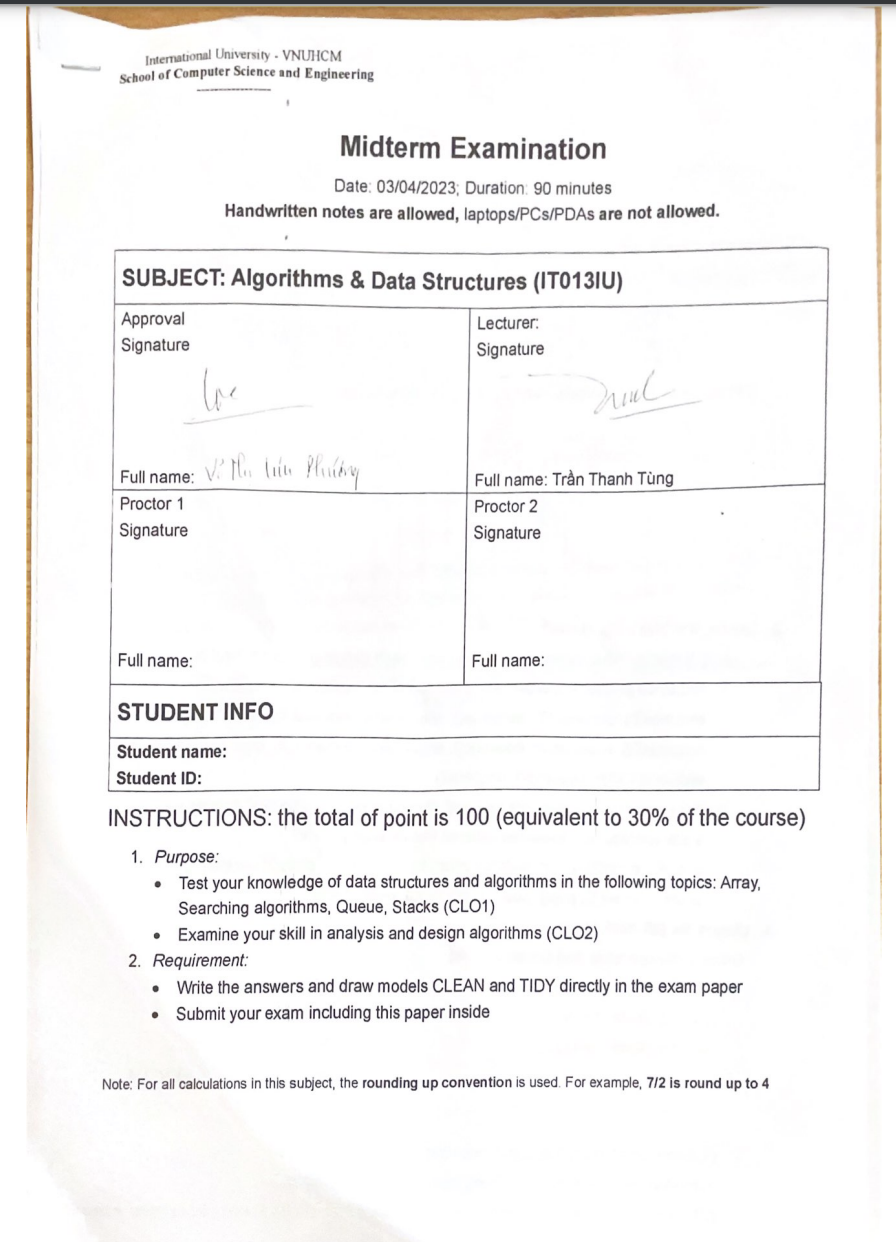
Em 

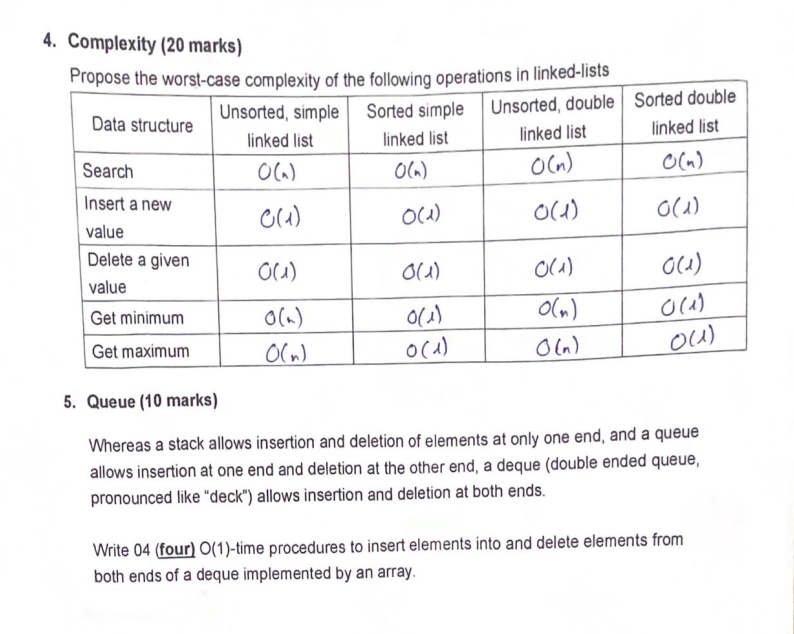
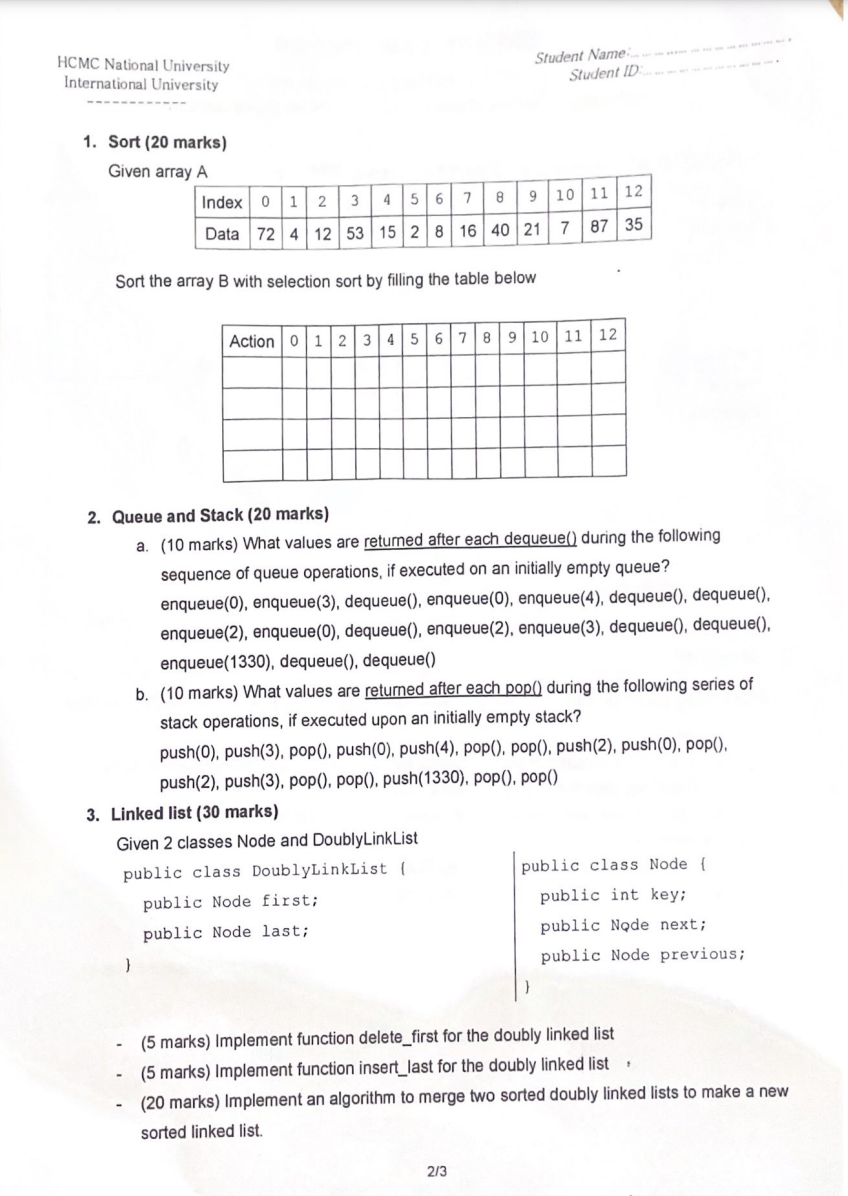












Q3.

### **Implementation for delete\_first:**

public void delete\_first() {

if (first == null) {

// List is empty, nothing to delete

System.out.println("List is empty");

return;

}

if (first == last) {

// Only one element in the list

first = last = null;

} else {

// More than one element

first = first.next;

first.previous = null;

}

}

### **Implementation for insert\_last:**

public void insert\_last(int key) {

Node newNode = new Node();

newNode.key = key;

if (last == null) {

// List is empty, so the new node is both the first and last

first = last = newNode;

} else {

// Append the new node to the end

last.next = newNode;

newNode.previous = last;

last = newNode;

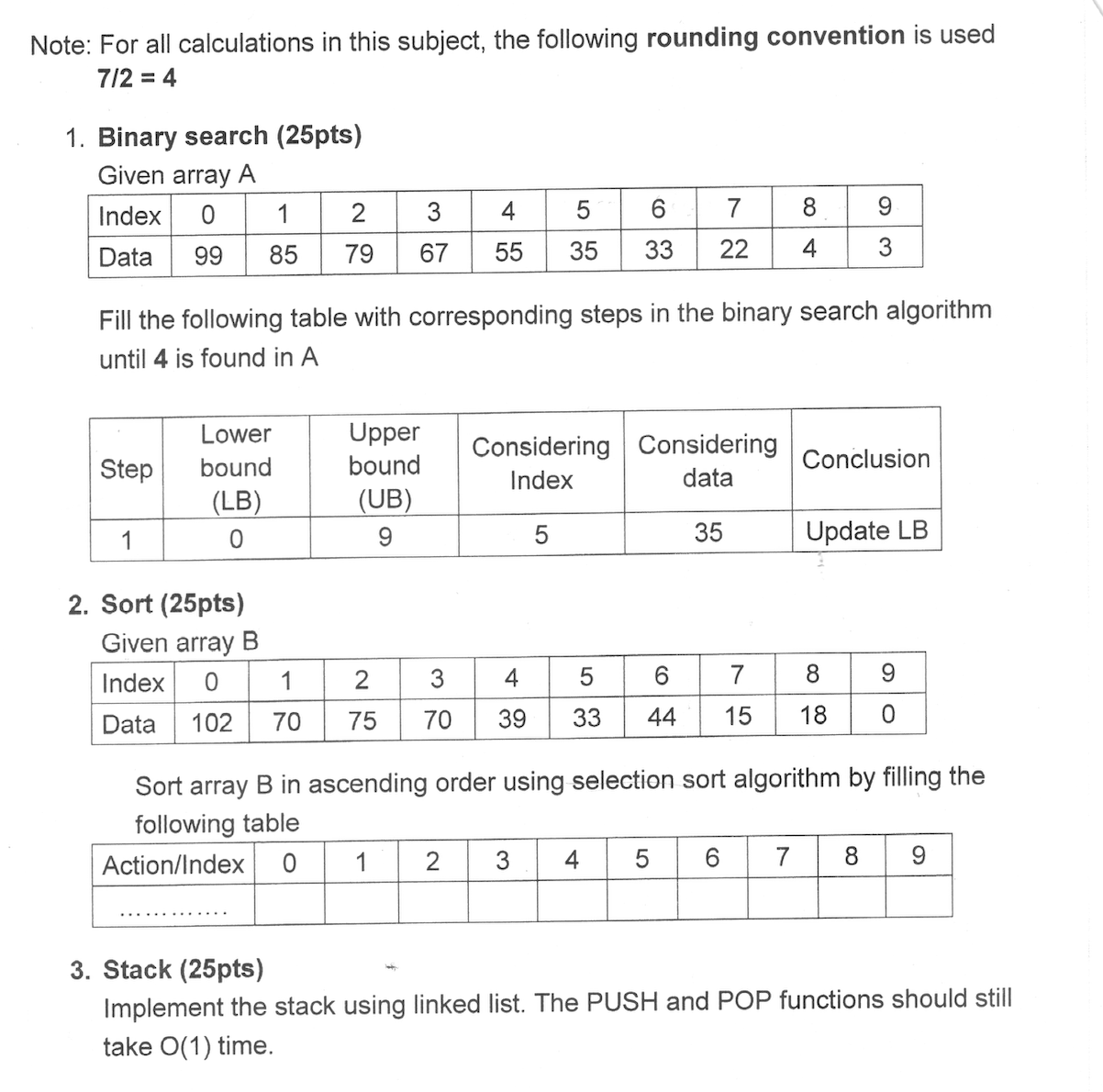
}

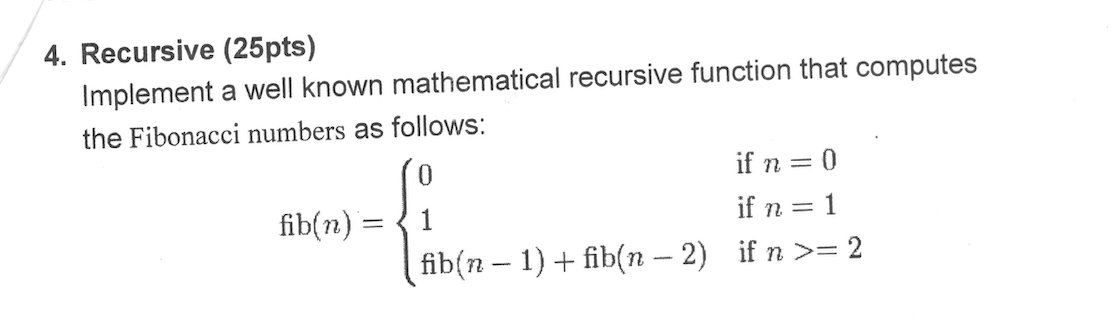
}

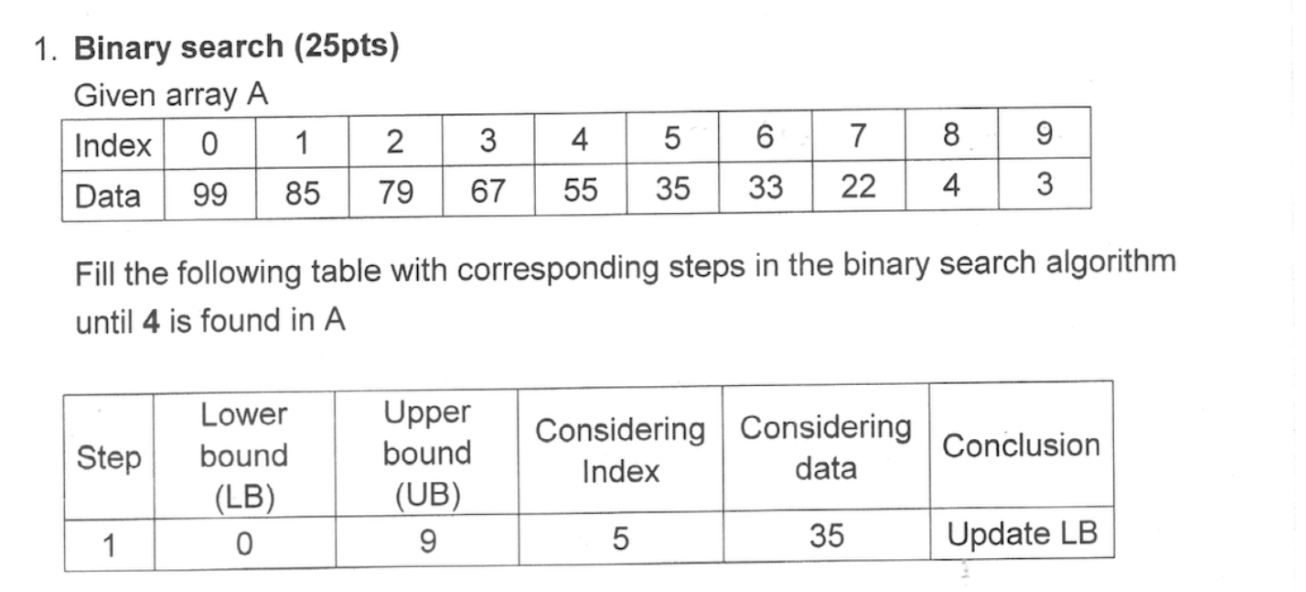
Q5.

| class Deque {  private int[] deque;  private int front;  private int rear;  private int size;  private int capacity;  public Deque(int capacity) {  this.capacity = capacity;  deque = new int[capacity];  front = -1;  rear = 0;  size = 0;  }  // Insert element at the front  public void insertFront(int value) {  if (isFull()) {  System.out.println("Deque is full");  return;  }  // If deque is empty  if (front == -1) {  front = 0;  rear = 0;  } else if (front == 0) {  front = capacity - 1; // Wrap around  } else {  front--;  }  deque[front] = value;  size++;  }  // Insert element at the rear  public void insertRear(int value) {  if (isFull()) {  System.out.println("Deque is full");  return;  }  // If deque is empty  if (front == -1) {  front = 0;  rear = 0;  } else if (rear == capacity - 1) {  rear = 0; // Wrap around  } else {  rear++;  }  deque[rear] = value;  size++;  }  // Delete element from the front  public void deleteFront() {  if (isEmpty()) {  System.out.println("Deque is empty");  return;  }  if (front == rear) {  front = -1;  rear = -1; // Deque is empty  } else if (front == capacity - 1) {  front = 0; // Wrap around  } else {  front++;  }  size--;  }  // Delete element from the rear  public void deleteRear() {  if (isEmpty()) {  System.out.println("Deque is empty");  return;  }  if (front == rear) {  front = -1;  rear = -1; // Deque is empty  } else if (rear == 0) {  rear = capacity - 1; // Wrap around  } else {  rear--;  }  size--;  }  // Check if the deque is full  public boolean isFull() {  return size == capacity;  }  // Check if the deque is empty  public boolean isEmpty() {  return size == 0;  }  // Get the front element  public int getFront() {  if (isEmpty()) {  System.out.println("Deque is empty");  return -1;  }  return deque[front];  }  // Get the rear element  public int getRear() {  if (isEmpty()) {  System.out.println("Deque is empty");  return -1;  }  return deque[rear];  }  } |
| --- |

**Đề 2014**

;



Câu 1: 

| Step | LB | UB | Considering index | Considering data | Conclusion |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 9 | 5 | 35 | Update LB |
| 2 | 6 | 9 | 7 | 22 | Update LB |
| 3 | 8 | 9 | 8 | 4 | Found!! |

Câu 2:

Take B[0] is min value in the outer loop. Creating an inner loop, take B[1] is current value, go through the array and compare it to min value.If current value is less than min value, so min value = current value.Repeat the step until reach the last index. Swap it to B[0] and keep its position, then take B[1] is min value and repeat previous steps

| Action/Index | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B[0] is min value | 102 | 70 | 75 | 70 | 39 | 33 | 44 | 15 | 18 | 0 |
| Swap B[0] with B[9] | 0 | 70 | 75 | 70 | 39 | 33 | 44 | 15 | 18 | 102 |
| B[1] is min value | 0 | 70 | 75 | 70 | 39 | 33 | 44 | 15 | 18 | 102 |
| Swap B[1] with B[7] | 0 | 15 | 75 | 70 | 39 | 33 | 44 | 70 | 18 | 102 |
| B[2] is min value | 0 | 15 | 75 | 70 | 39 | 33 | 44 | 70 | 18 | 102 |
| Swap B[2] with B[8] | 0 | 15 | 18 | 70 | 39 | 33 | 44 | 70 | 75 | 102 |
| B[3] is min value | 0 | 15 | 18 | 70 | 39 | 70 | 44 | 70 | 75 | 102 |
| Swap B[3] with B[5] | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| B[4] is min value | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| Keep B[4] position | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| B[5] is min value | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| Swap B[5] with B[6] | 0 | 15 | 18 | 33 | 39 | 44 | 70 | 70 | 75 | 102 |
| B[6] is min value | 0 | 15 | 18 | 33 | 39 | 44 | 70 | 70 | 75 | 102 |
| Keep B[6] position | 0 | 15 | 18 | 33 | 39 | 44 | 70 | 70 | 75 | 102 |
| B[7] is min value | 0 | 15 | 18 | 33 | 39 | 44 | 70 | 70 | 75 | 102 |
| Keep B[7] position | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| B[8] is min value | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| Keep B[8] position | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| B[9] is min value | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |
| Keep B[9] position | 0 | 15 | 18 | 33 | 39 | 70 | 44 | 70 | 75 | 102 |

Arr [10] int

| #HE34 | #HE38 |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

The code:

import java.util.Arrays;

public class Sort {

public static void main(String[] args) {

int[] B = { 102, 70, 75, 70, 39, 33, 44, 15, 18, 0 };

int n = B.length;

**for (int i = 0; i < n - 1; i++) {**

**int minIndex = i;**

**for (int j = i + 1; j < n; j++) {**

**if (B[j] < B[minIndex]) {**

**minIndex = j;**

**}**

**}**

**swap(i,min);**

**Complexity: (n-1)(n-1)**

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

int minIndex = i;

}

for (int j = i+1 ; j < n; j++) {

if (B[j] < B[minIndex]) {

minIndex = j;

}

**Complexity: (n-1) + (n-1)**

int temp = B[minIndex];

B[minIndex] = B[i];

B[i] = temp;

System.out.println(Arrays.toString(B)); // print the array after each swap

}

}

}

**Câu 3:**

class Node {

int value;

Node next;

public Node(int value) {

this.value = value;

this.next = null;

}

}

class Stack {

private Node head;

private int size;

public Stack() {

this.head = null;

this.size = 0;

}

public void push(int value) {

Node newNode = new Node(value);

newNode.next = head;

head = newNode;

size++;

}

public int pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

int poppedValue = head.value;

head = head.next;

size--;

return poppedValue;

}

public boolean isEmpty() {

return size == 0;

}

public int getSize() {

return size;

}

}

**Câu 4:**

import java.io.\*;

class fibonacci {

static int fib(int n)

{

if (n <= 1)

return n;

return fib(n - 1) + fib(n - 2);

}

public static void main(String args[])

{

int n = 9;

System.out.println(

n + "th Fibonacci Number: " + fib(n));

}

}