

Data Structure Midterm Exam:

Your name:

Student number:

Q1. Is it possible to exchange ① and ② in the following insert function for the linked queue? If not, explain why. (20pt)

```
#include <stdio.h>
#include <malloc.h>

typedef int element;
typedef struct QueueNode {
    element item;
    struct QueueNode *link;
} QueueNode;

typedef struct {
    QueueNode *front, *rear;
}QueueType;

void enqueue(QueueType *q, element item) {
    QueueNode *temp = (QueueNode *)malloc(sizeof(QueueNode));
    if (temp == NULL)
        error("error");
    else {
        temp->item = item;
        temp->link = NULL;
        if (is_empty(q)) {
            q->front = temp;
            q->rear = temp;
        }
        else {
            q->rear->link = temp; //①
            q->rear = temp; //②
        }
    }
}
```

Q2. Write a function to reverse the order of a singly linked list. (20pt)

Input) 3 -> 5 -> 7 -> NULL

Output) 7 -> 5 -> 3 -> NULL

```
Node* Reverse(Node *head)
```

```
{
```

```
    Node* tail = NULL;
```

```
    Node* cur;
```

```
    while(head != NULL){
```

```
        1) cur = head->next;
```

```
        2) head->next = tail;
```

```
        3) tail = head;
```

```
        4) head = cur;
```

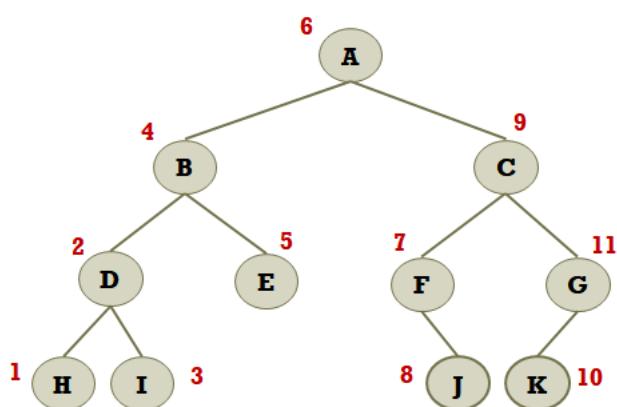
```
}
```

```
    return tail;
```

```
}
```

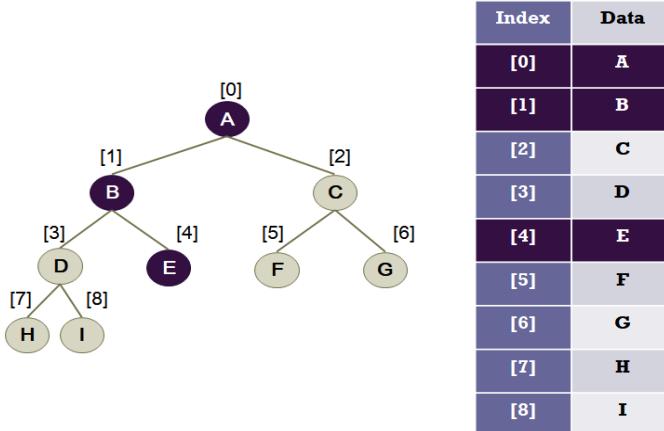
Q3. You want to implement a function for in-order traversal of a binary tree without recursion. For this task, what is the most right data structure among stack, queue, array, and linked list? The tree is NOT threaded. Using the data structure of your choice, sketch the algorithm in pseudo-code. (20pt)

Inorder



Q4. Prove/disprove the following claim for a binary tree. (20pt)

Claim: The index of the parent node of node i is $\lfloor (i - 1)/2 \rfloor$



Q5. Fill the gray box for a recursive function to find the height of a tree in pseudo-code or C / C++. Note that the tree is not necessarily a binary one. You are given two functions: (20pt)

Tree LEFT_MOST_CHILD(Tree T) /* returns a subtree with the left most child as the root */

Tree RIGHT_SIBLING(Tree T) /* returns a subtree with the right sibling as the root */

```

height (Tree T)
{
    Tree subtree = LEFTMOST_CHILD(T);
    if (subtree == NULL) return 0; // leaf node
    int max_height = 0;
    while (subtree != NULL) { // find the maximum among children's heights
        int h = height(subtree);

        if (h > max_height) max_height = h;

        subtree = RIGHT_SIBLING(subtree);
    }
    return (max_height+1);
}

```