

Vietnam National University,  
Ho Chi Minh City

UNIVERSITY OF SCIENCE  
FACULTY OF INFORMATION TECHNOLOGY



[CSC10004] Data Structure And Algorithms

---

## EXERCISE I REPORT

Implementing Stack and Queue from scratch

---

**Student Name**

Bui Minh Duy

**Student ID**

23127040

**Lecturer in charge:**

Bui Duy Dang

Nguyen Thanh Tinh

16/06/2024

# Contents

<b>1</b>	<b>Experiment Details</b>	<b>2</b>
1.1	Stack Array . . . . .	2
1.2	Stack Linked List . . . . .	3
1.3	Queue Array . . . . .	4
1.4	Queue Linked List . . . . .	5
<b>2</b>	<b>Recursive versions</b>	<b>6</b>
2.1	Stack . . . . .	6
2.2	Queue . . . . .	6
2.3	Conclusion . . . . .	7
<b>3</b>	<b>Self-evaluation</b>	<b>8</b>
<b>4</b>	<b>Exercise Feedback</b>	<b>8</b>
4.1	What I learned . . . . .	8
4.2	What was difficult . . . . .	8

# 1 Experiment Details

## 1.1 Stack Array

### Push Operation

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 1  
Enter value: 3  
Stack: 1 2 3  
-----
```

(a) Normal case

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 1  
Enter value: 6  
Stack is full! Can't push!  
Stack: 1 2 3 4 5  
-----
```

(b) Full case

Figure 1: Push elements in Stack Array of size 5

### Pop Operation

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 1  
Enter value: 5  
Stack: 1 2 3 4 5  
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Popped value: 5  
Stack: 1 2 3 4  
-----
```

(a) Normal case

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Popped value: 2  
Stack: 1  
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Popped value: 1  
Stack is empty! Can't print!  
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Stack is empty! Can't pop!  
-----
```

(b) Empty case

Figure 2: Pop elements in Stack Array of size 5

## 1.2 Stack Linked List

### Push Operation

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 1  
Enter value: 4  
Stack: 4 3 2 1  
-----
```

(a) Normal Case

Figure 3: Push elements in Stack Linked List

### Pop Operation

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 1  
Enter value: 5  
Stack: 5 4 3 2 1  
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Popped value: 5  
Stack: 4 3 2 1  
-----
```

(a) Normal case

```
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Popped value: 2  
Stack: 1  
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Popped value: 1  
Stack is empty! Can't print!  
-----  
[1] Push  
[2] Pop  
[3] Exit  
Enter choice: 2  
Stack is empty! Can't pop!  
-----
```

(b) Empty case

Figure 4: Pop elements in Stack Linked List

## 1.3 Queue Array

### Enqueue Operation

```
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 1
Enter value: 3
Queue: 1 2 3
-----
```

(a) Normal case

```
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 1
Enter value: 6
Queue is full! Can't enqueue!
Queue: 1 2 3 4 5
-----
```

(b) Full case

Figure 5: Enqueue elements in Queue Array of size 5

### Dequeue Operation

```
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 1
Enter value: 4
Queue: 1 2 3 4
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Dequeued item: 1
Queue: 2 3 4
-----
```

(a) Normal case

```
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Dequeued item: 3
Queue: 4
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Dequeued item: 4
Queue is empty! Can't print!
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Queue is empty! Can't dequeue.
-----
```

(b) Empty case

Figure 6: Dequeue elements in Queue Array of size 5

## 1.4 Queue Linked List

### Enqueue Operation

```

-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 1
Enter value: 4
Queue: 1 2 3 4
-----

```

(a) Normal case

Figure 7: Enqueue elements in Queue Linked List

### Dequeue Operation

```

-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 1
Enter value: 6
Queue: 1 2 3 4 5 6
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Dequeued item: 1
Queue: 2 3 4 5 6
-----

```

(a) Normal case

```

-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Dequeued item: 5
Queue: 6
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Dequeued item: 6
Queue is empty! Can't print!
-----
[1]. Enqueue
[2]. Dequeue
[3]. Exit
Enter choice: 2
Queue is empty! Can't dequeue!
-----

```

(b) Empty case

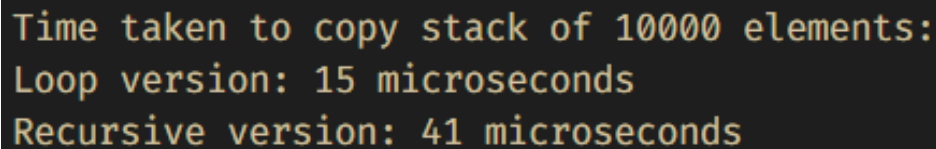
Figure 8: Dequeue elements in Queue Linked List

## 2 Recursive versions

### 2.1 Stack

#### Recursive Stack Array

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of  $O(n)$  for **copy** operations.
- **Actual execution time:**

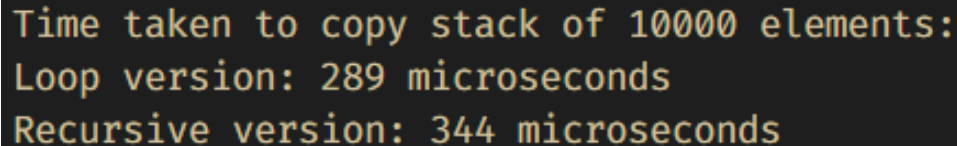


```
Time taken to copy stack of 10000 elements:  
Loop version: 15 microseconds  
Recursive version: 41 microseconds
```

Figure 9: Comparison between Loop version and Recursive version

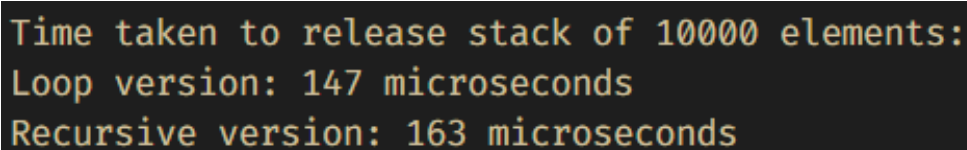
#### Recursive Stack Linked List

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of  $O(n)$  for **copy** and **release** operations.
- **Actual execution time:**



```
Time taken to copy stack of 10000 elements:  
Loop version: 289 microseconds  
Recursive version: 344 microseconds
```

Figure 10: Comparison between Loop version and Recursive version



```
Time taken to release stack of 10000 elements:  
Loop version: 147 microseconds  
Recursive version: 163 microseconds
```

Figure 11: Comparison between Loop version and Recursive version

### 2.2 Queue

#### Recursive Queue Array

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of  $O(n)$  for **copy** operations.
- **Actual execution time:**

```
Time taken to copy a queue of 10000 elements:  
Loop version: 29 microseconds  
Recursive version: 80 microseconds
```

Figure 12: Comparison between Loop version and Recursive version

### Recursive Queue Linked List

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of  $O(n)$  for `copy` and `release` operations.
- **Actual execution time:**

```
Time taken to copy a queue of 10000 elements:  
Loop version: 287 microseconds  
Recursive version: 335 microseconds
```

Figure 13: Comparison between Loop version and Recursive version

```
Time taken to release a queue of 10000 elements:  
Loop version: 145 microseconds  
Recursive version: 185 microseconds
```

Figure 14: Comparison between Loop version and Recursive version

## 2.3 Conclusion

- Though Loop version and Recursive Version have the same theoretical time complexity of  $O(n)$ , the actual execution time shows that Loop version is faster in all cases.
- When working with large data sets, the Loop version still executes as normal, but the Recursive version might encounter Stack Overflow.



### 3 Self-evaluation

No.	Details	Score
1	Stack Array	100%
2	Stack Linked List	100%
3	Queue Array	100%
4	Queue Linked List	100%
5	Recursive versions	100%
6	Report	100%
	<b>Overall</b>	<b>100%</b>

### 4 Exercise Feedback

#### 4.1 What I learned

- Better understanding about the internal processing of stack and queue.
- How to use function templates in C++.

#### 4.2 What was difficult

- The approach of using recursion is quite challenging, it took a significant amount of time and effort to complete the set-up.