香港中文大學（深圳）
The Chinese University of Hong Kong

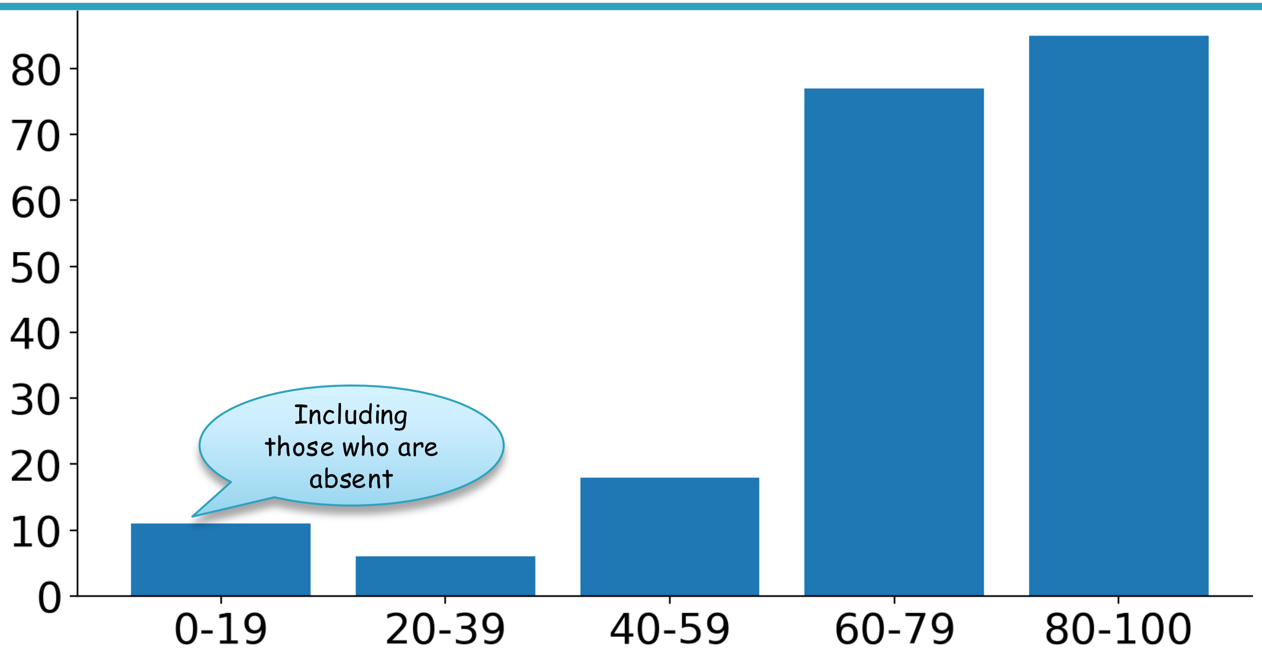
CSC3100 Data Structures
Mid-term Exam

School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen

Overview

#1 Multiple Choice Questions	(30)
#2 True or False	(10)
#3 Short Answer Questions	(30)
#4 Doubly Linked List	(5)
#5 Read Program	(10)
#6 Find Second Largest Value	(5)
#7 Reverse First K Elements in a Queue	(10)

Grade Distribution



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total
Average	20.9	7.8	23.8	4.0	7.4	4.0	8.1	75.3
Median	21	7.8	25	4.3	8	5	10	77
Max	30	10	30	5	10	5	10	98
Min	0	2	6	0	0	0	0	9

#1 Multiple Choice Questions

(1). Which of the following is a linear data structure?

A. Array

B. Linked List

C. Stack

[✓ D. All of the above]

(2). What is the worst-case time complexity of accessing an element in an array with n elements?

[✓ A. $O(1)$]

B. $O(\log n)$

C. $O(n)$

D. $O(n^2)$

(3). Which of the following is NOT a valid operation on an array?

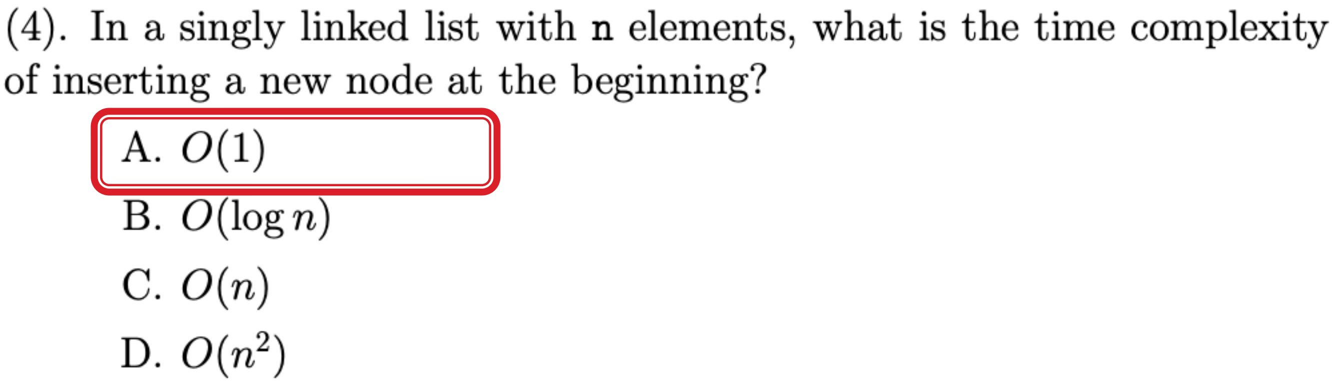
A. Accessing an element at a given index.

B. Appending an element to the end.

[✓ C. Deleting an element at an arbitrary position without shifting.]

[✓ D. Finding the maximum element in O(1) time (unsorted array).]

#1 Multiple Choice Questions



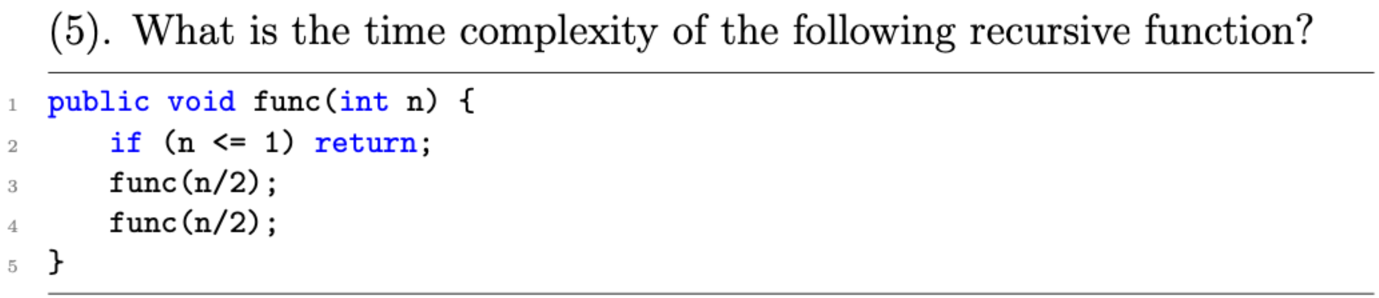
(4). In a singly linked list with n elements, what is the time complexity of inserting a new node at the beginning?

[✓ A. $O(1)$]

B. $O(\log n)$

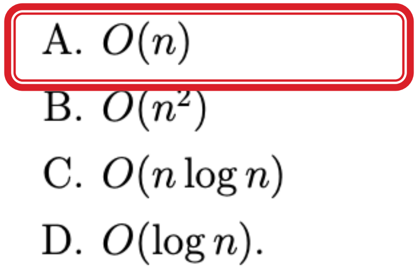
C. $O(n)$

D. $O(n^2)$



(5). What is the time complexity of the following recursive function?

```java
public void func(int n) {
    if (n <= 1) return;
    func(n/2);
    func(n/2);
}
```



[✓ A. $O(n)$]

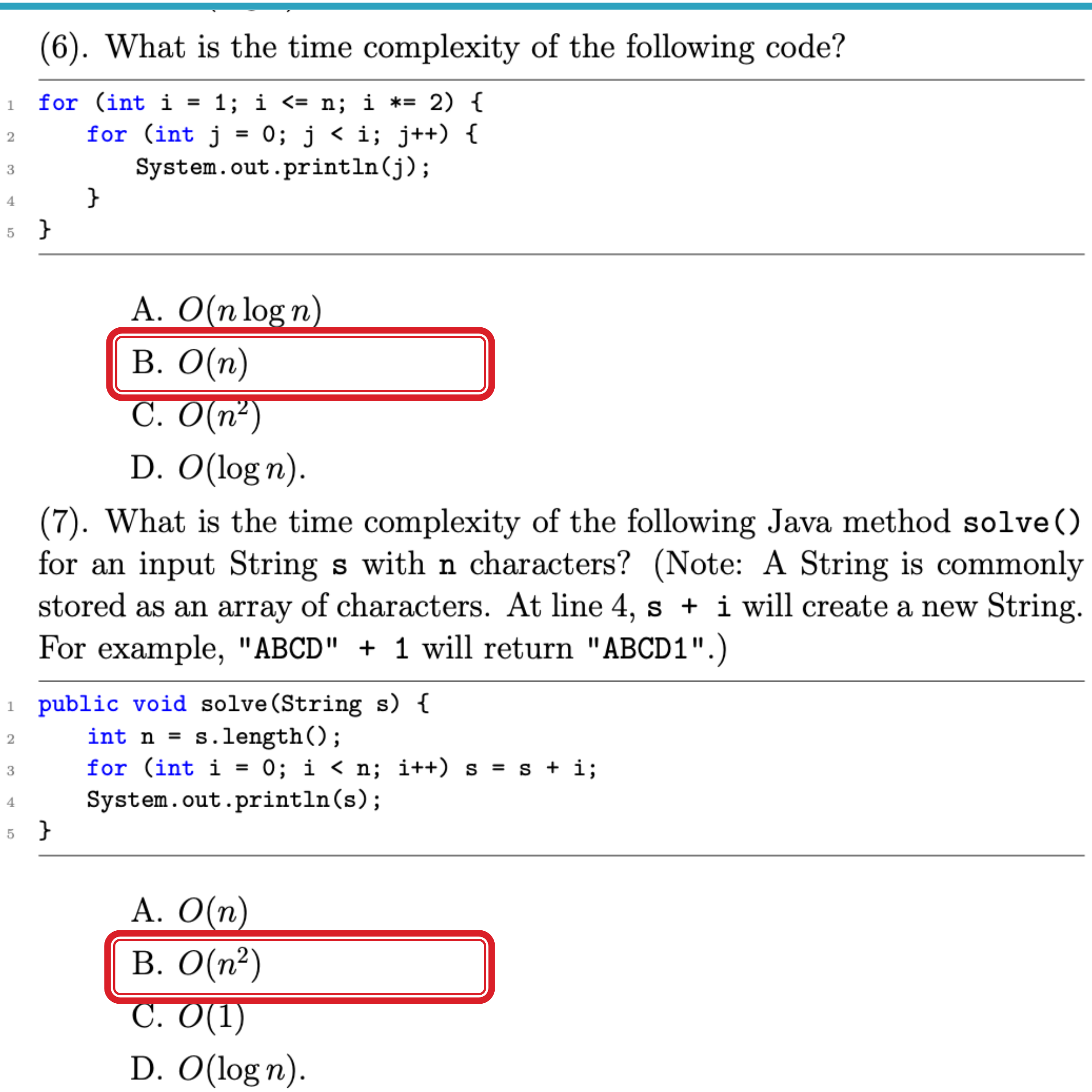
B. $O(n^2)$

C. $O(n \log n)$

D. $O(\log n)$.

- $g(n) = 2g\left(\frac{n}{2}\right) + O(1)$ and let $g(1) = b$
- When $n = 2^x$, we have that: $g(n) \le 2g\left(\frac{n}{2}\right) + c \le 4g\left(\frac{n}{4}\right) + c + 2c \le 8g\left(\frac{n}{8}\right) + c + 2c + 4c \ldots \le 2^x \cdot g(1) + c + 2c + 4c + \cdots + 2^{x-1}c \le bn + cn - c$
- When $n \neq 2^x$, we can extend the array to the nearest power of two n' and show that $g(n) \le g(n') \le bn' + n' - c \le 2bn + 2cn - c$.
- Thus, $g(n) = O(n)$

#1 Multiple Choice Questions



(6). What is the time complexity of the following code?

```java
for (int i = 1; i <= n; i *= 2) {
    for (int j = 0; j < i; j++) {
        System.out.println(j);
    }
}
```

A. $O(n \log n)$

[✓ B. $O(n)$]

C. $O(n^2)$

D. $O(\log n)$.

(7). What is the time complexity of the following Java method `solve()` for an input String `s` with `n` characters? (Note: A String is commonly stored as an array of characters. At line 4, `s + i` will create a new String. For example, `"ABCD" + 1` will return `"ABCD1"`.)

```java
1 public void solve(String s) {
2     int n = s.length();
3     for (int i = 0; i < n; i++) s = s + i;
4     System.out.println(s);
5 }
```

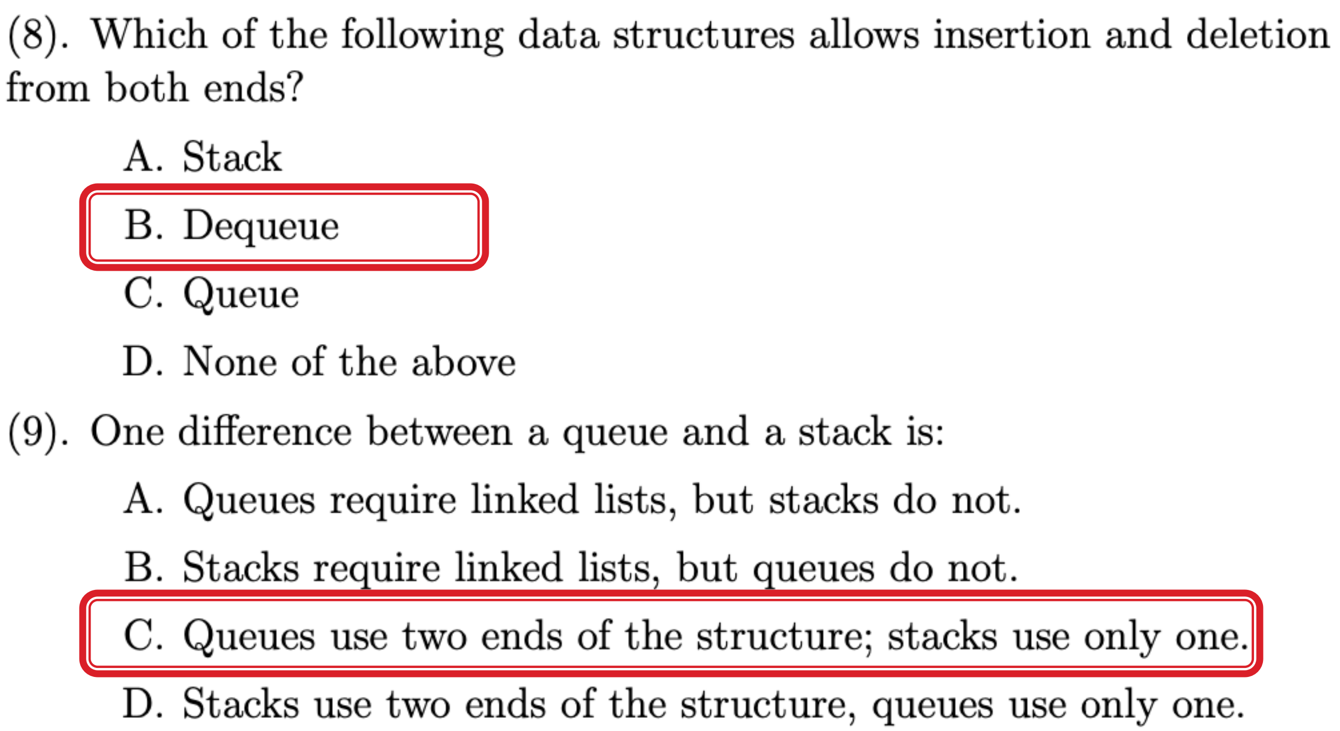
A. $O(n)$

[✓ B. $O(n^2)$]

C. $O(1)$

D. $O(\log n)$.

#1 Multiple Choice Questions



(8). Which of the following data structures allows insertion and deletion from both ends?

A. Stack

[✓ B. Dequeue]

C. Queue

D. None of the above

(9). One difference between a queue and a stack is:

A. Queues require linked lists, but stacks do not.

B. Stacks require linked lists, but queues do not.

[✓ C. Queues use two ends of the structure; stacks use only one.]

D. Stacks use two ends of the structure, queues use only one.

#1 Multiple Choice Questions

(10). Here is an INCORRECT pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
declare a character stack
while ( more input is available) {
    read a character
    if ( the character is a ’(’ )
        push it on the stack
    else if ( the character is a ’)’ and the stack is not empty )
        pop a character off the stack
    else
        print "unbalanced" and exit
}
print "balanced"
```

Which of these unbalanced sequences does the above code think is balanced?

[✓ A. ((())]

B. ())(()

C. (()()))

D. (()))()

#2 True or False

(1) **([✓ True]/False)** One can implement a stack based on a linked list so that each individual `push/pop` operation is in time $O(1)$.

(2) **([✓ True]/False)** One can reverse the order of the elements in a singly linked list with `n` elements in time $O(n)$.

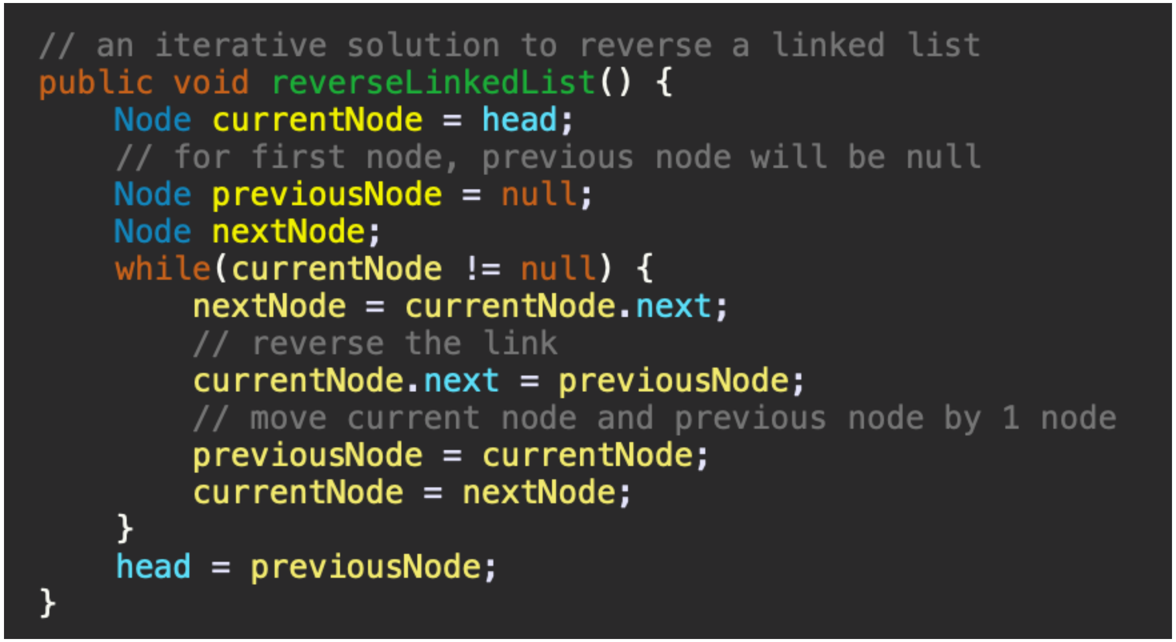
(3) **(True/False)** For any two given singly linked lists of arbitrary sizes, it is possible to concatenate (merge) them in time $O(1)$.

(4) **(True/False)** The worst-case time complexity of searching an element in a sorted array of `n` elements using binary search is $O(\log n)$.

(5) **(True/False)** A queue can be implemented using two stacks.

#2 True or False

- (2) Reverse a linked list



```java
// an iterative solution to reverse a linked list
public void reverseLinkedList() {
    Node currentNode = head;
    // for first node, previous node will be null
    Node previousNode = null;
    Node nextNode;
    while(currentNode != null) {
        nextNode = currentNode.next;
        // reverse the link
        currentNode.next = previousNode;
        // move current node and previous node by 1 node
        previousNode = currentNode;
        currentNode = nextNode;
    }
    head = previousNode;
}
```

#2 True or False

(1) **([✓ True]/False)** One can implement a stack based on a linked list so that each individual **push/pop** operation is in time $O(1)$.

(2) **([✓ True]/False)** One can reverse the order of the elements in a singly linked list with **n** elements in time $O(n)$.

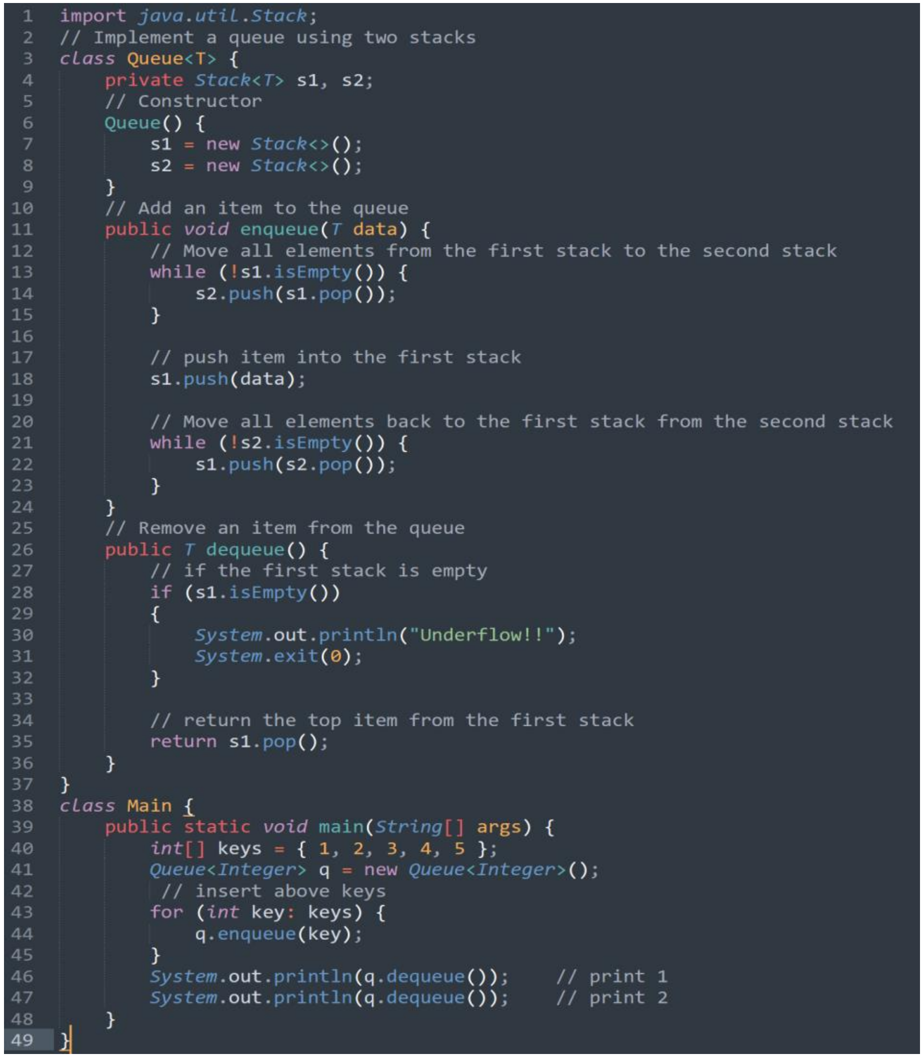
(3) **(True/[✓ False])** For any two given singly linked lists of arbitrary sizes, it is possible to concatenate (merge) them in time $O(1)$.

(4) **([✓ True]/False)** The worst-case time complexity of searching an element in a sorted array of **n** elements using binary search is $O(\log n)$.

(5) **([✓ True]/False)** A queue can be implemented using two stacks.

#2 True or False

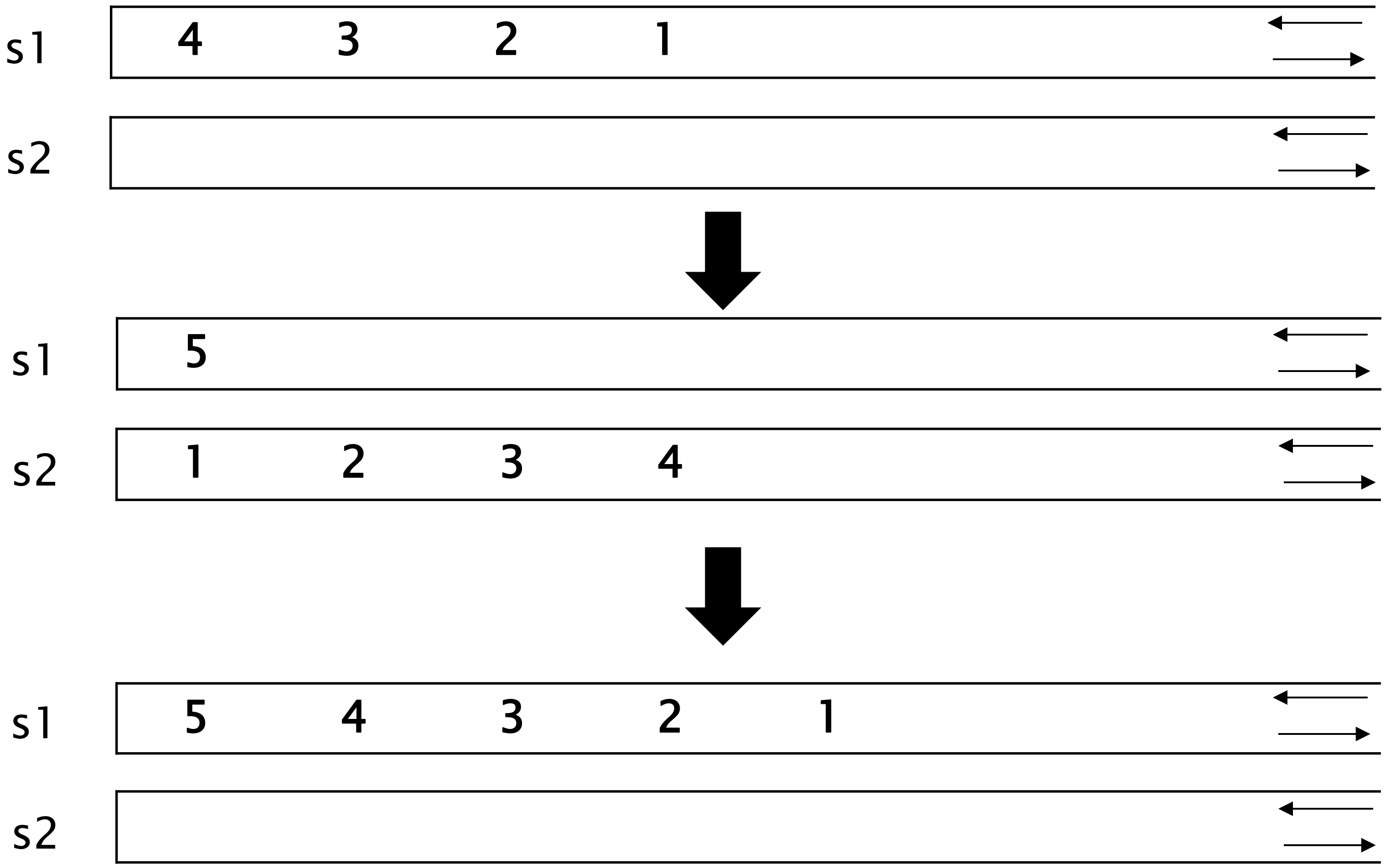
- (5) Implement a queue using two stacks



```java
import java.util.Stack;
// Implement a queue using two stacks
class Queue<T> {
    private Stack<T> s1, s2;
    // Constructor
    Queue() {
        s1 = new Stack<>();
        s2 = new Stack<>();
    }
    // Add an item to the queue
    public void enqueue(T data) {
        // Move all elements from the first stack to the second stack
        while (!s1.isEmpty()) {
            s2.push(s1.pop());
        }

        // push item into the first stack
        s1.push(data);

        // Move all elements back to the first stack from the second stack
        while (!s2.isEmpty()) {
            s1.push(s2.pop());
        }
    }
    // Remove an item from the queue
    public T dequeue() {
        // if the first stack is empty
        if (s1.isEmpty())
        {
            System.out.println("Underflow!!");
            System.exit(0);
        }

        // return the top item from the first stack
        return s1.pop();
    }
}
class Main {
    public static void main(String[] args) {
        int[] keys = { 1, 2, 3, 4, 5 };
        Queue<Integer> q = new Queue<Integer>();
         // insert above keys
        for (int key: keys) {
            q.enqueue(key);
        }
        System.out.println(q.dequeue());    // print 1
        System.out.println(q.dequeue());    // print 2
    }
}
```

#2 True or False

- (5) Implement a queue using two stacks



s1 4 3 2 1

s2

s1 5

s2 1 2 3 4

s1 5 4 3 2 1

s2

#3 Short Answer Questions

3. [6*5] Concisely answer the following short questions.
(a) List five types of common data structures.

Answer:

Array, Linked list, Stack, Queue, Tree, ...

A typical error:

int, double, char, long, bool

#3 Short Answer Questions

3. [6*5] Concisely answer the following short questions.

(b) What is the main advantage of using a linked list over an array?

Answer:

The main advantage of linked lists is dynamic memory allocation and efficient insertion/deletion operations.

A typical error:

Only answer one of the two key advantages.

#3 Short Answer Questions

3. [6*5] Concisely answer the following short questions.

(c) How does the time complexity of inserting an element at the beginning of an array compare to that of a linked list?

Answer:

Array: O(n); Linked List: O(1).

Therefore, the time complexity for array insertion is greater than that of a linked list.

A typical error:

The time complexity for array insertion equals that of a linked list.

#3 Short Answer Questions

3. [6*5] Concisely answer the following short questions.

(d) What is the advantage of using a circular queue over a linear queue?

Answer:

The main advantage of a circular queue is efficient space utilization.

A typical error:

The circular queue has lower time complexity.

#3 Short Answer Questions

3. [6*5] Concisely answer the following short questions.

(e) Why is a stack more suitable for implementing undo operations, which are commonly seen in text editors, than a queue?

Answer:

A stack is ideal for undo operations as its LIFO behavior matches the expected undo sequence, while a queue's FIFO approach would inappropriately undo the oldest action first.

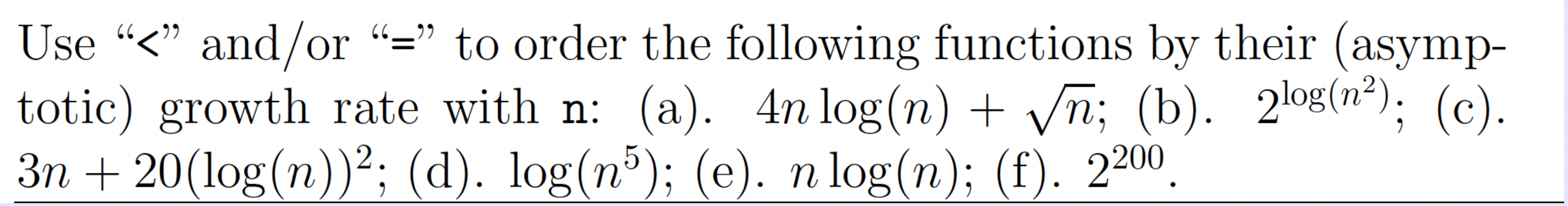
A typical error:

Stacks use LIFO without explaining why this property specifically benefits undo operations.

#3 Short Answer Questions

3. [6*5] Concisely answer the following short questions.

(f) Asymptotic analysis of 6 functions.



Use “<” and/or “=” to order the following functions by their (asymptotic) growth rate with **n**: (a). $4n\log(n)+\sqrt{n}$; (b). $2^{\log(n^2)}$; (c). $3n+20(\log(n))^2$; (d). $\log(n^5)$; (e). $n\log(n)$; (f). 2^{200}.

Answer:

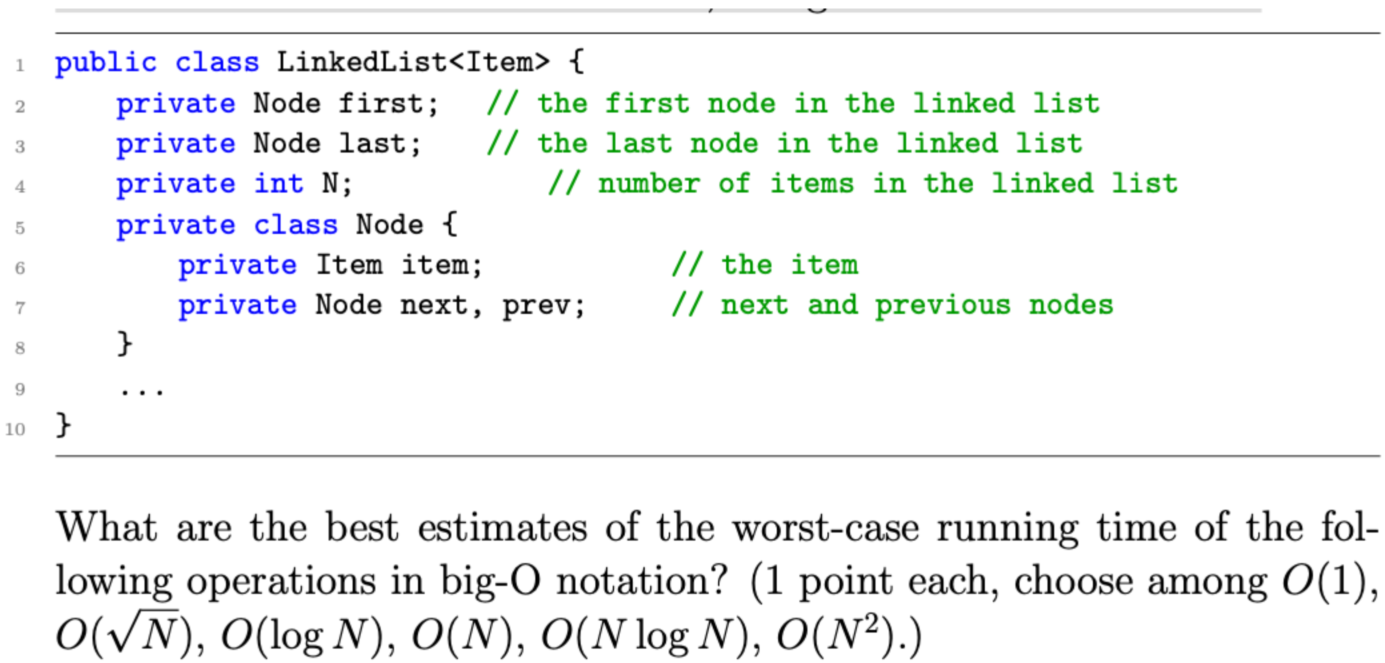
f < d < c < a = e < b

A typical error:

n < (a), n < (b), n = (c), (d) < n, n < (e), (f) < n

#4 Doubly Linked List

4. [5*1] To implement a doubly linked list, maintaining a reference to the first and last node in the list, along with its size as follows.



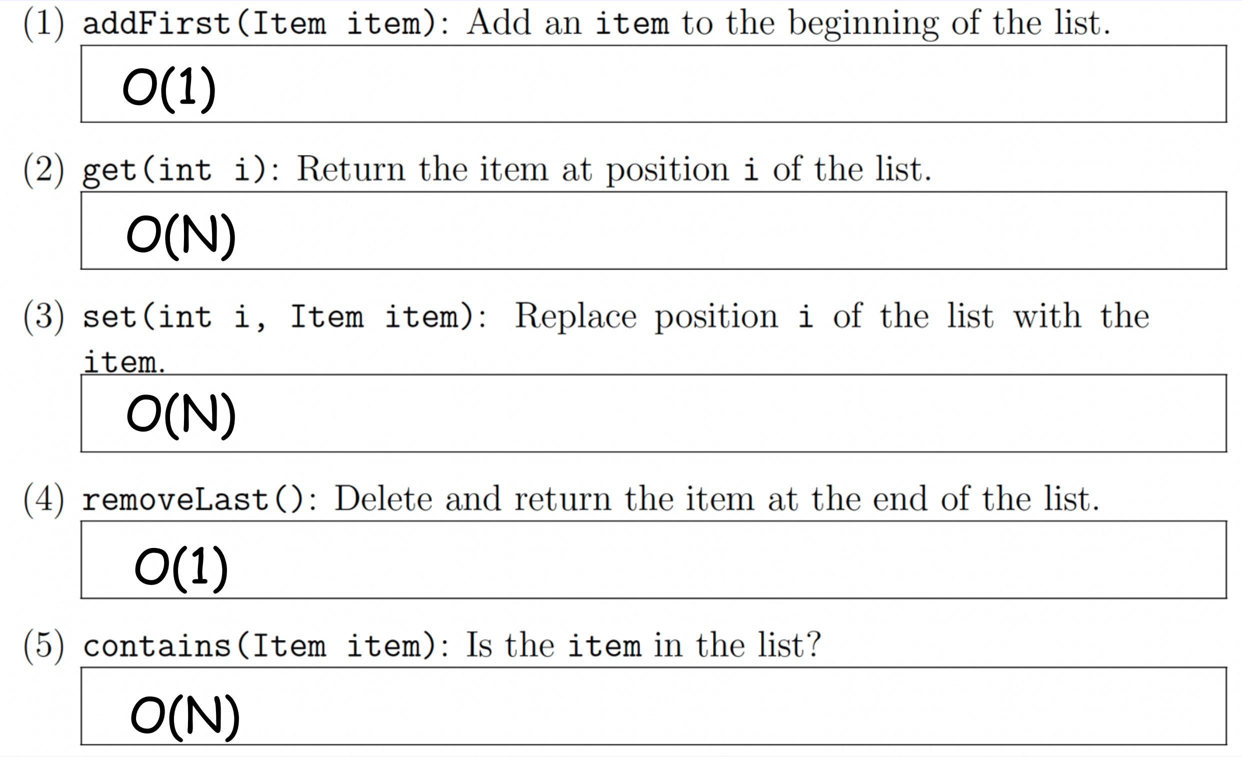
```
public class LinkedList<Item> {
    private Node first;   // the first node in the linked list
    private Node last;    // the last node in the linked list
    private int N;            // number of items in the linked list
    private class Node {
        private Item item;           // the item
        private Node next, prev;     // next and previous nodes
    }
    ...
}
```

What are the best estimates of the worst-case running time of the following operations in big-O notation? (1 point each, choose among $O(1)$, $O(\sqrt{N})$, $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^2)$.)

#4 Doubly Linked List

4. [5*1] To implement a doubly linked list, maintaining a reference to the first and last node in the list, along with its size as follows.

Answer:



(1) `addFirst(Item item)`: Add an `item` to the beginning of the list.

O(1)

(2) `get(int i)`: Return the item at position `i` of the list.

O(N)

(3) `set(int i, Item item)`: Replace position `i` of the list with the `item`.

O(N)

(4) `removeLast()`: Delete and return the item at the end of the list.

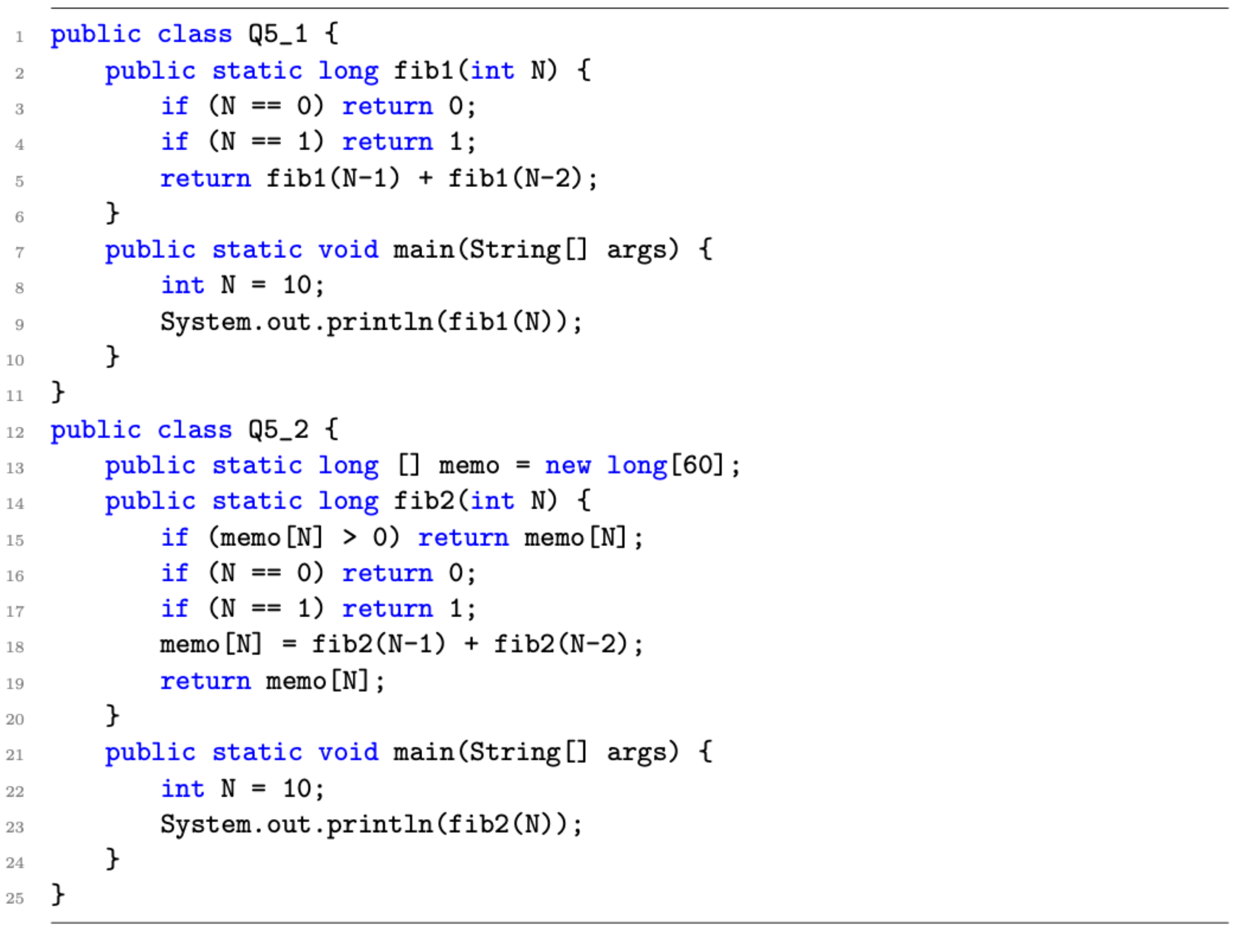
O(1)

(5) `contains(Item item)`: Is the `item` in the list?

O(N)

#5 Read Program

5. [5*2] Read the following Java programs, and answer questions.



```java
public class Q5_1 {
    public static long fib1(int N) {
        if (N == 0) return 0;
        if (N == 1) return 1;
        return fib1(N-1) + fib1(N-2);
    }
    public static void main(String[] args) {
        int N = 10;
        System.out.println(fib1(N));
    }
}
public class Q5_2 {
    public static long [] memo = new long[60];
    public static long fib2(int N) {
        if (memo[N] > 0) return memo[N];
        if (N == 0) return 0;
        if (N == 1) return 1;
        memo[N] = fib2(N-1) + fib2(N-2);
        return memo[N];
    }
    public static void main(String[] args) {
        int N = 10;
        System.out.println(fib2(N));
    }
}
```

#5 Read Program

5. [5*2] Read the following Java programs, and answer questions.

(1) Write down the outputs of the two programs, i.e., the returned values of fib1(10) and fib2(10) respectively.

Answer:

Fib1(10) = 55, Fib2(10) = 55

(2) For methods fib1(int N) and fib2(int N), estimate the orders of growth of operations (time complexity) as a function of the input N, respectively.

Answer:

For Fib1, O(2^N)

For Fib2, O(N)

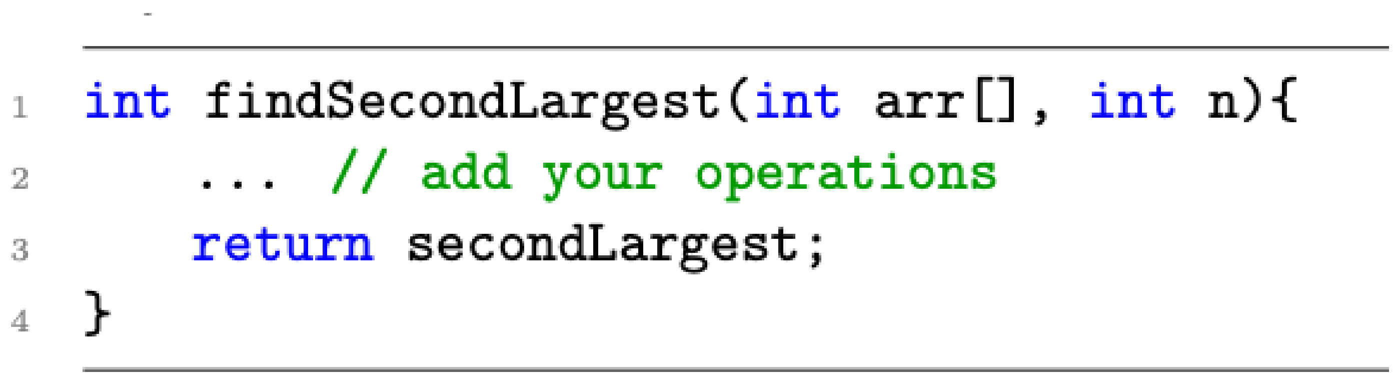
Typical Errors:

O(N^2), O(N)

O(logN)

#6 Find Second Largest Value

6. [5] Given an unsorted array of size n, write an algorithm in O(n) to find the second largest element in the array.



```
int findSecondLargest(int arr[], int n){
    ... // add your operations
    return secondLargest;
}
```

6 Find Second Largest Value

6. [5] Given an unsorted array of size n, write an algorithm in O(n) to find the second largest element in the array.

Answer:

```
if (arr.length < 2) { return Integer.MIN_VALUE; }.
if (arr.length == 2) { return Math.min(arr[0], arr[1]); }

int largest = Math.max(arr[0], arr[1]);
int second_largest = Math.min(arr[0], arr[1]);

for (int idx = 2; idx < arr.length; idx++) {
    if (arr[idx] > largest) {
        second_largest = largest;
        largest = arr[idx]; }
    else if (arr[idx] > second_largest) {
        second_largest = arr[idx]; }
}
return second_largest;
```

6 Find Second Largest Value

6. [5] Given an unsorted array of size n, write an algorithm in O(n) to find the second largest element in the array.

Typical Error:

```
if (arr.length < 2) { return Integer.MIN_VALUE; }.
if (arr.length == 2) { return Math.min(arr[0], arr[1]); }

int largest = Math.max(arr[0], arr[1]);
int second_largest = Math.min(arr[0], arr[1]);

for (int idx = 2; idx < arr.length; idx++) {
    if (arr[idx] > largest) {
        second_largest = largest;
        largest = arr[idx]; }
    else if (arr[idx] > second_largest) {   // Forget to consider this situation
        second_largest = arr[idx]; }
}
return second_largest;
```

#7 Reverse First K elements in a Queue

7. [10] Given an integer k and a queue Q with at least k elements, reverse the first k elements while keeping the rest of the queue unchanged.

Answer:

```
void reverseKELements(queue Q, int k){
  queue Q_aux;
  stack S_aux;
  int n = Q.size();
  for(int i=0;i<k;i++){
    S_aux.push(Q.front());
    Q.pop();
  }
  for(int i=k;i<n;i++){
    Q_aux.push(Q.front());
    Q.pop();
  }
  for(int i=0;i<k;i++){
    Q.push(S_aux.top());
    S_aux.pop();
  }
  for(int i=k;i<n;i++){
    Q.push(Q_aux.front());
    Q_aux.pop();
  }
  return Q;
}
```

```
void reverseKELements(queue Q, int k){
  stack S;
  int n = Q.size();
  for(int i=0;i<k;i++){
    S.push(Q.front());
    Q.pop();
  }
  for(int i=0;i<k;i++){
    Q.push(S.top());
    S.pop();
  }
  for(int i=k;i<n;i++){
    Q.push(Q.front());
    Q.pop();
  }
  return Q;
}
```

#7 Reverse First K elements in a Queue

7. [10] Given an integer k and a queue Q with at least k elements, reverse the first k elements while keeping the rest of the queue unchanged.

A typical error:

```
void reverseKELements(queue Q, int k){
  queue Q_aux;
  stack S_aux;
  int n = Q.size();
  for(int i=0;i<k;i++){
    S_aux.push(Q.front());
    Q.pop();
  }
  for(int i=0;i<k;i++){
    Q_aux.push(Q.front());
    Q.pop();
  }
  for(int i=0;i<k;i++){
    Q.push(S_aux.top());
    S_aux.pop();
  }
  for(int i=0;i<k;i++){
    Q.push(Q_aux.front());
    Q_aux.pop();
  }

  return Q;
}
```