**CSE 214 – Recitation 4: Queues**

**1. [5 minutes] Consider the following statements:**

i. The stack data structure follows the FIFO (First In First Out) principle.

ii. The queue data structure follows the LIFO (Last In First Out) principle.

iii. The enqueue operation inserts an element onto the front of the queue.

iv. The dequeue operation removes an element from the rear of the queue.

v. In a priority queue, using the peek operation will return the highest priority element.

vi. One implementation of the queue data structure is to check whether an

arithmetic expression has balanced parenthesis.

**Which of the following is correct?**

A. (i) and (iv) are true

B. (ii) and (iii) are true

C. (i), (ii), (iii) and (iv) are true

D. (v) is true

E. (i), (ii), (iii), (iv) and (v) are true

F. None of above

**2. [2 minutes] A normal queue, implemented using a circular array, gets full when?**

A. rear == CAPACITY

B. rear == CAPACITY – 1

C. (rear + 1) % CAPACITY == front

D. B and C

E. None of above

**3. [2 minutes] A normal queue, implemented using a circular array, is empty when?**

1. front == −1 && rear == −1
2. rear == front
3. front = 0 && rear = 0
4. A and B
5. None of above

**4. [5 minutes] Consider a priority queue implemented using a sorted array. What is the worst case time complexity for the following operations?**

1. Enqueue O(n)
2. Peek O(1)
3. Dequeue O(1)

**5. [5 minutes] What is the worst-case complexity of the following:**

|  |  |  |
| --- | --- | --- |
|  | **Enqueue** | **Dequeue** |
| **Array with pointer to next available slot** | O(1) | O(n) |
| **Array without pointer to next available slot** | O(n) | O(n) |
| **Circular Array** | O(1) | O(1) |
| **Singly linked list with head reference and head as the front** | O(n) | O(1) |
| **Singly linked list with head and tail references with the head as the front** | O(1) | O(1) |
| **Singly linked list with head and tail references with the tail as the front** | O(1) | O(n) |
| **Doubly linked list with head and tail references with the head or tail as the front** | O(1) | O(1) |

**6. [15 minutes] Write a method that removes and return the nth element in an IntQueue. Assume you have access to the basic queue methods, such as enqueue, dequeue, size, etc.**

public int remove (IntQueue q, int n) {

first\_el = q.pop();

prev\_el = first\_el;

int count = 0;

while(q.peek() != first\_el){

if(count != n){

q.enqueue(prev\_el);

count++;

prev\_el = q.dequeue();

}

}

}

**7. [5 minutes] Given a Queue implemented using a circular array with a capacity of 5**

**and a sequence of operations:**

front = -1

rear = -1

10

2

5

7

8

0 1 2 3 4

1. enqueue(3)
2. enqueue(4)
3. enqueue(6)
4. dequeue()
5. enqueue(8)
6. enqueue(7)
7. enqueue(5)
8. dequeue()
9. dequeue()
10. enqueue(2)
11. enqueue(10)

10-2-5-7-8 OR

5-2-10-8-7 where 10 is R and F is 8

What does the Queue look like after the above 11 operations?

0 1 2 3 4

rear front

**What would happen if we try an extra operation “enqueue(9)” ?**

QUEUE FULL EXCEPTION

8. [10 minutes] Write the following method to reverse a queue using recursion. Assume you have access to the following operations

1. enqueue(x) : Add an item x to rear of queue.
2. dequeue() : Remove an item from front of queue.
3. isEmpty() : Checks if a queue is empty or not.

IntQueue reverseQueue(IntQueue q) {

if(q.isEmpty()){return new intQueue;}

else{

curr\_el = q.dequeue();

updated Q = reverseQueue();

updated.enqueue(curr\_el);

return q;

}

}