Eric Ong (861267345)
Thomas Chin (861290572)
Section 002

1. What is the Big-Oh runtime in terms of n(number of inputs) for the following algorithm:
HINT: It is not O(n^2)

```
for (i = 0; i < n; i = i * 2)
   for (j = 0; j < n; ++j)
      statement - O(1)
        statement - O(1)
        statement - O(1)
```

The outer loop has a Big-Oh runtime of O(log(N)) because i is being multiplied by 2 each iteration.

The inner loop has a Big-Oh runtime of O(N) because i is incremented by 1 during each iteration.

Therefore, because there is a nested loop, you would multiple the two Big-Oh runtimes of both the outer and inner loop together, so the resulting Big-Oh runtime in terms of n would be **O(Nlog(N))**.

2. Convert the following expression from infix to postfix notation. You must show the stack and output at each step (just like the slides). (a+b)*(c-d*e+f)

Stack	**Output**
(	
(	a
(+	a
(+	ab
	ab+
*	ab+
*(	ab+
*(	ab+c
*(-	ab+c
*(-	ab+cd
(-	ab+cd
(-	ab+cde
*(+	ab+cde*-
*(+	ab+cde*-f
*	ab+cde*-f+
	ab+cde*-f+*

3. Write a pseudocode algorithm to solve the following problem:
Given 2 queues, Q1 & Q2, merge them into a single queue, Q3.
Each element in the queues have an arrival time data member, which stores the time that they were inserted into their queue.
Elements with lower time values were inserted first.
When done, the values in Q3 should be stored with their arrival times in ascending order.
You are not changing the arrival times, only accessing them to decide which element should be stored in Q3 next.
You may only use the ADT operations listed in the slides: enqueue, dequeue, getFront, size, isEmpty.

```
//Q1 and Q2 defined

//Q3 intialized

while (Q1 and Q2 not empty)

     if (Q1 arrival time < Q2 arrival time)

          enqueue Q1's front

          dequeue Q1

     else if (Q2 arrival time <= Q1 arrival time)

          enqueue Q2's front

          dequeue Q2
```

4. What is the Big-Oh running time complexity of your algorithm in question 3. Explain.

The Big-Oh runtime complexity of question 3's algorithm is O(N) because the algorithm iterates through the loop N number of times, depending on whether or not Q1 and Q2 are empty.

5. List and explain the Big-Oh runtimes of enqueue, dequeue, and getFront
for a queue that uses a node-based implementation (single-link) that has pointers to both the head and tail nodes:

enqueue at the head and dequeue at the tail
enqueue: O(1) because a node is just inserted at the beginning (head).
dequeue: O(N) because you have to keep track of the node of what's before the tail.
getFront: O(1) because you have direct access to the head node, so you can get the value from there.

enqueue at the tail and dequeue at the head
enqueue: O(1) because you have direct access to the tail node, so you can just insert after the tail node.
dequeue: O(1) since you have direct access to the head, so you can just delete the node.
getFront: O(1) because you have direct access to the head node, so you can get its value.