|  |  |
| --- | --- |
| **IN CLASS ASSIGNMENT 1** | **Algorithm & Running Time Analysis** |
|  |  |

Your work should be readable as well as correct.

**Team 4**

Janet Eames  
Emmett Rasmussen

Habib Kalia

Jacob Garcia

1. **Algorithm and Running Time Analysis:** Give the tightest possible upper bound for the ***worst case*** running time for each of the following in terms of *N. Assume that the most time-efficient implementation is used.*
   1. Pushing a value onto a stack containing N values, implemented as a linked list.

**Explanation: O(1)**

Given that the top of the stack is at the head of the linked list, then inserting the value at the head of the list, you do not have to traverse the list. Therefore, making the time complexity O(1).

* 1. Pushing a value onto a stack implemented as an array. Assume the array is of size 2N.

**Explanation:**  **O(1)**

As putting an object at the end of an array is constant and the size of the list is determined and does not need to be resized, the worst-case time would simply be O(1). If the array is not large enough and needs to be resized, then the time complexity would be amortized constant.

* 1. Enqueue a value onto a queue containing N values implemented as a circular array (as described in class). (Assume the array is size N+5.)

**Explanation**: **O(1)**

The worst-case scenario would be O(1), as inserting into an empty spot on an array is constant time. If the array is not large enough and needs to be resized, then the time complexity would be amortized constant.

* 1. Pop a value off a stack containing N elements implemented as an array.

**Explanation**: **O(1)**

Like ***b*** popping or pushing something to the top of a stack is constant if the stack grows from the end of the array to the beginning. If the array is not large enough and needs to be resized, then the time complexity would be amortized constant.

* 1. Popping a value in a stack implemented as linked list. Be specific in explaining how you get the runtime you provide.

**Explanation: O(1)**

Popping something off a linked list is constant time as the head of the linked list is popped and then the next point in the linked list is marked as the head. Its worst-case time is constant.

* 1. Given a FIFO queue implemented as a linked list, find and remove all of the values greater than 10, leaving the rest of the queue in its original order.

**Explanation: O(n)**

Given that you enqueue at the tail of the list and dequeue at the head, and that you keep track of the tail of the list; enqueuing and dequeuing will take constant time. To remove all values greater than 10, we go through all the elements in the queue, dequeuing and then enqueuing. This takes O(n) time.

* 1. Given a FIFO queue implemented as a linked list currently containing N values, enqueue N more values so that when you are finished the queue will contain 2N values (Give the total time for enqueueing N more values).

**Explanation: O(n)**

Enqueuing a single value is done in constant time, enqueuing n values would be linear time because the time complexity increases at the same rate n does.

* 1. Finding something in a sorted array?

**Explanation: O(logN)**

Searching an array does increase in time complexity based on the size of the array, but as the array is sorted using the best sorting algorithm it does not take the full O(N) time to complete, and is therefore O(logN) using binary search.

* 1. Finding and removing all values greater than 12 from a stack implemented with a linkedlist (leaving the rest of the stack in its original order)?

**Explanation: O(n)**

Popping a value off the stack, comparing it, and pushing it onto another stack. Once we reach the bottom of the stack, we pop from the second stack and push onto the first stack. This is 2N operations and therefore it takes O(n) time complexity.