CSCI 132 Lab 9: Algorithm Analysis and Big-O Running Time

Instructions: For each of the four algorithms below, you will need to find the running time of the algorithm and state the total running time in Big-O notation. For each operation in the algorithm, you will find the running time/time complexity of that operation (just like we did in class). You can print this out and do it by hand or do it in a word/pdf editor. You will not be submitting any code for this lab. This lab is due Thursday October 31st at 11:59 PM.

Algorithm 1: Given an even-length array, this algorithm splits the input array into two equal-sized sub arrays: `split_array1` and `split_array2`.

Example Execution:

```
Input: [1, 2, 3, 4 ,5 ,6 ,7, 8]
Output array #1: [1, 2, 3, 4]
Output array #2: [5, 6, 7, 8]
```

```java
public static void split_array(int[] input_array) {

     int[] split_array1 = new int[input_array.length / 2];

     int[] split_array2 = new int[input_array.length / 2];

     for(int i = 0; i < input_array.length/2;i++) {

          split_array1[i] = input_array[i];

     }

     int counter = 0;

     for(int j = input_array.length/2; j < input_array.length;j++) {

          split_array2[counter] = input_array[j];

          counter++;

     }

     System.out.println("Output array #1: " + Arrays.toString(split_array1));

     System.out.println("Output array #2: " + Arrays.toString(split_array2));

}
```

Total Running Time:

Algorithm 2: Given two arrays, this algorithm will return the number of matching elements found between the two arrays. Each array will not have any duplicate elements, and **the two input arrays will be the same size**.

If you are in CSCI 246 (Discrete Structures) this semester, this algorithm returns the size of the **intersection** between Array A and Array B, i.e. $\lvert A \cap B \rvert$

Example Execution:

Array 1: [**1**, **7**, 5, 3, **2**]

Array 2: [0, **1**, 6, **7**, **2**]

Output:

Match found: 1 1

Match found: 7 7

Match found: 2 2

Value returned: 3

```
public static int calculate_matches(int[] array1, int[] array2) {

     int num_of_matches = 0;

     if(array1.length != array2.length) {

          System.out.println("Error: Arrays are not same size. Program terminating...");

          return 0;

     }

     else {

          for(int i = 0; i < array1.length; i++) {

               for(int j = 0; j < array2.length; j++) {

                    if(array1[i] == array2[j]) {

                         System.out.println("Match Found:" + array1[i] + " " + array2[j]);

                         num_of_matches++;

                    }

               }

          }

    }

return num_of_matches;

}
```

Total Running Time:

Algorithm 3: Given a doubly linked list, this algorithm returns true if there are no nodes in the linked list, and returns false if there is at least one node in the linked list. It does this by checking if the head and tail nodes are null, and if they are both null, then the algorithm returns true.

You can assume that the code that adds nodes and removes nodes from this linked list is correct.

```
public boolean is_linked_list_empty() {
      if(this.head == null) {
            if(this.tail == null) {
                  return true
            }
            else {
                  return false
            }
      }
      else {
            return false
      }
}
```

Total Running Time:

Algorithm 4: Given a Linked List Queue that holds strings (`LinkedList<String> queue`), this algorithm searches for a String (`search`) and removes it from the Queue.

(You can assume that `.equals()` runs in **O(1)** time)

Hint: `.remove()` runs in **O(n)** time

```
public void search_and_remove(String search) {
    int counter = 0;
    for(String each: this.queue) {
        if(each.equals(search)) {
            this.orders.remove(counter);
        }
        counter++;
    }
}
```

Total Running Time: