Assignment 1

Date Assigned: 01/19/2017
Due: Midnight 02/02/2017 on iLearn

Please read turn-in checklist at the end of this document before you start doing exercises.

Section 1: Pen-and-paper Exercises

1. Arrange the following functions in ascending order of growth rate (15 points):

2010, loglogn, logn, $\sum_{i=1}^{n} \frac{1}{i}$, $\sqrt{n}$, n, $\sum_{i=1}^{n} 1$, nlogn, $\sum_{i=1}^{n} i$, n^2, n^4, 2^n, e^n, n!, n^n

2. Analyze the following code and provide a "Big-O" estimate of its running time in terms of n. Explain your analysis.

```
1   void compute (int n, double[][] A, double[][] B, double[][] C, double[][] D)
2   {
3      for (int i=0; i < n; i++) {
4        for (int j=0; j < n; j++) {
5          C[i][j] = A[i][j] + B[i][j];
6        }
7      }
8
9      for (int i=0; i < n; i++) {
10       for (int j=0; j < n; j++) {
11         for (int k=0; k < n; k++)
12            D[i][j] = D[i][j] + A[i][k] * B[k][j];
13       }
14     }
15
16  }
```

Line 3: the first for loop will be executed n times
Line 4: inner for loop will be executed n times within the first for loop fo a total of n^2 times.
Line 5: 2 primitive operations; assignment and addition. this is executed n^2 times for a total of 2n^2
Line 9: the first loop is executed n times
Line 10: the second for loop is executed n times, totalling to n^2
Line 11: the third for loop is executed at n times, totaling to n^3
Line 12: this statement has 3 primitive operations that are executed n^3 times. Totalling to 3n^3

The estimate is : 3n^3 + 2n^2
The tight bound is O(n^3)

3. My Program Analyzes each combination of integers in the array and adds them. If the sum is equal to the specified integer, the program will return true, otherwise it will return false.

Input: an array, A, of n integers (positive, negative, or 0), and an integer value t.

```
for i in [0,size(A)-1]                          //n
        for j in [i+1, size(A-1)]               //(n-1)n = n^2 - n
                for k in [j+1, size(A-1)]       //(n-2)(n^2-n) = n^3-3n^2+2n
                        if(A[i]+A[j]+A[k] == t) //3 * n^3-3n^2+2n = 3n^3-9n^2+6n
                                return true;    //doesnt count
                        end if
                end for
        end for
end for
return false;
```

Output: “yes” if there exist three indices i, j and k, with 0 <= i < j < k <= n-1, such that A[i] + A[j] + A[k] = t, and “no” otherwise.

The running time is approximately O(3n^3-9n^2+6n)
Or using a tight bound: O(n^3)

4. Consider the following problem:

I struggled creating this algorithm mostly because no matter what i did I could not get it to print, and therefore moved the print statement inside the algorithm, but since the method is called only once it doesnt make a difference.

The function of the algorithm will work as follows: We instantiate a variable to keep track of the number we are using to partition the array. We create a new array to be used to move the elements into. We check if the given number is less than or equal to the partition, and if it is we put it as the first element in the new array B. if it is greater, we put it in the end of the array.

Input: An array of integers

```
        partition = A[0]                    //1
        B[] = new[size(A)]                  //1
        start = 0                           //1
        end = size(B)-1                     //2
        print(“partition = “ + partition)   //2 (concatenation)
        for i in [1,size(A)]                //n
                if(A[i]<=partition)         //(1)(n) = n
                        B[start] = A[i]     //(1)(n) = n
                        start = start +1    //(2)(n) = 2n
                end if                      //
                else                        //
                        B[end] = A[i]       //(1)(n) = n      <-- We don't count these because
                        end = end - 1       //(2)(n) = 2n           they are not the worst case
                end else                    //
        end for                             //
```

```
if(end==start)                  //1
        B[start] = partition    //1
        A = B                   //1
end if                          //
                                //
print("after partition")        //1
for i in [0,size(A)]            //n
        print(A[i] + "")        //(2)(n) = 2n
 end for                        //
```

Output: You need to rearrange the array to have the following property:

the first 5 lines are all simple operations. the first for loop is executed n times. The if statement is broken into two pieces. The 1st option has a big o of 4n, while the second option has a big o of 3n. so we go with the 1st option over the second. The end for loop is executed n times, for a total of 2n.

The Big-O is O(6n + 11)
The tight bound is O(n)