Worksheet: Module 2

CS 315: Data Structures and Algorithms

Richard S. Stansbury
Embry-Riddle Aeronautical University
Daytona Beach, FL

Name: Jeremiah Webb Student ID: 2545328

1. Match the order of growth to the common name/descriptor for it. Pick the best answer.

Choices: constant, polynomial, linear, exponential, logarithmic, loglinear, quadratic, factorial,

1	_____Constant_____
$\log(n)$	_____Logarithmic_____
n	_____Linear_____
$n * \log(n)$	_____Loglinear_____
n^2	_____Quadratic_____
n^c	___Polynomial_____
c^n	_____Exponential_____
$n!$	_____Factorial_____

2. What is the tilde approximation **and** order of growth for the following runtime models

   a. $f(n) = 1/6 * n^2 + 2n + 2$

      Tilde approximation: $\frac{n^2}{6}$

      Order of growth: n^2, quadratic

   b. $f(n) = 2 * n * \log(n) + 2n + 2$

      Tilde approximation: $2 * n * \log(n)$

      Order of growth: $n * \log(n)$. loglinear

   c. $f(n) = n^3$

      Tilde approximation: n^3

      Order of growth: n^3. cubic

3. Calculate the order of growth for the algorithm below.
   - First, trace the execution of the algorithm for a sample input (just to observe how it works).
   - Second, approximate the runtime of the inner loop given the number of operations and frequency for which they occur.
   - Third, approximate the runtime of the outer loop (including the nested inner loop).
   - Finally, determine the overall order of growth.

```
public static void Algorithm1(int [] data)
{

    T2->int [] sortedArr = new int[data.length];
    int lastSmallest = -1;

    //Outer Loop
    for (int i=0; i < data.length; i++) {

        int smallestVal = Integer.MAX_VALUE;
        int smallestIndex = -1;


        // Inner Loop

T1->        for (int j = 0; j < data.length; j++) {

            if (data[j] >= lastSmallest) {
                if (data[j] < smallestVal) {
                    smallestIndex = j;
                    smallestVal = data[j];
                }
            }
        }

        lastSmallest = sortedArr[i] = data[smallestIndex];<-T0
    }
    data = sortedArr;
}
```

Sample input: [1,2,3,4,5,6] This is a sorting algorithm.
Inner Loop is n linear runtime.
Due to the outer loop containing inner loop, should be n^2 time.
if statements are assumed constant time.

$$\sum_{i=0}^{n-1}((\sum_{j=i+1}^{n-1}T_0)+T_1)+t_2$$

$$\sum_{i=0}^{n-1}(T_0(N-1)+T_1)+t_2$$

$$(T_0*(N^2-\frac{N(N-1)}{2})+T_1*N+T_2)$$

$(T_0\frac{N^2}{2})+(\frac{N}{2})+T_1*N+T_2$ Overall Growth is n^2, quadratic.

4. Calculate order of growth of the following:

```
public void example1(int [] anArray)
{
    int n = anArray.length;
    T1 → for (int i=1; i < n; i*=2) {
            System.out.println(anArray[i]); ← T0
        }
}
```

Note: notice the increment term of the for loop. How does this impact complexity?

a. Tip: Trace through the example on a couple of practice arrays (say one of length 4 and one of length 8).
   1. AnArray = [1,3,5,7]
   2. AnArray = [1,2,3,4,5,6,7,8]
   1. Output: 3,5
   2. Output: 2,3,5

b. How many steps were required for each? How does this number of steps change as the size increases?
   When inputted array is length 4: 2 Steps
   When inputted array is length 8: 3 Steps

c. So, what is the order of growth?
   The order of growth is log(n). This is due to variable I being multiplied by 2 each iteration.
   I = 2^m = n
   m = log(N)
   $T_0 * \log(n) + T_1$
   So Order of growth is Log(n), logarithmic

5. Perform algorithm analysis of the running time (evaluate summations if necessary) of the below algorithm (show ALL work). Solve for the order of growth in terms of size n.

```
//For a given array, for each value, the result
//of value³ is calculated

public void cubeArray(int [] anArray)
{
      int n = anArray.length;
      int [] result = new int[n]; <- T3


      //calculate results
      T1->for (int i=0; i < n; i++) {
            result[i] = 1;
            for (int j=0; j < 3; j++) {
                  result[i] = result[i] * anArray[i];
            } ← T0

      }

      //Print results
      for (int k=0; k < n; k++) {
            System.out.println(result[k] + " ");
      } ← Constant Time T2
}
```

Example execution code:

```
int [] arr = {1,2,3,4,5};
cubeArray(arr) ;
```

Execution Output:

1 8 27 64 125

Final for loop printing results is linear.

$$f(n)=\sum_{i=0}^{N-1}((\sum_{j=0}^{3}T_0)+T_1)+T_3+\sum_{N-1}^{k=0}T_2$$

$$3*T_0*N+T_1*N+T_3+T_2*N$$

$$(3*T_0+T_1+T_2)*N+T_3$$

End result is linear Order N.

6. Determine the order of growth complexity for the following algorithm:

```
public void sums(int [] arr) {

      int sum = 0;
T2-> for (int i=0; i < arr.length; i++) { <-T1
            int j;
            for (j=1, sum=arr[0]; j <= i; j++) {
                  sum += arr[j]
            }

            System.out.println(i + ", " j + ", " + sum); <-T0
      }
}
```

Order of Growth Complexity: n^2, Quadratic, due to a nested for loop.

$$\left(\frac{T_0 * N(N-1)}{2}\right) + T_1 * N + T_2$$

$$\frac{T_0}{2} N^2 - T_2 * N + T_1 * N + T_2 ¿$$

$$\frac{T_0}{2} N^2 \; order\; is\; N^2$$