**Worksheet: Module 2**

CS 315: Data Structures and Algorithms

Richard S. Stansbury

Embry-Riddle Aeronautical University

Daytona Beach, FL

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

**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];  
 }  
 data = sortedArr;  
}

1. Calculate order of growth of the following:

public void example1(int [] anArray)

{

int n = anArray.length;

for (int i=1; i < n; i\*=2) {

System.out.println(anArray[i]);

}

}

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

1. Tip: Trace through the example on a couple of practice arrays (say one of length 4 and one of length 8).
2. How many steps were required for each? How does this number of steps change as the size increases?
3. So, what is the order of growth?
4. 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];  //calculate results  for (int i=0; i < n; i++) {  result[i] = 1;  for (int j=0; j < 3; j++) {  result[i] = result[i] \* anArray[i];  }    }  //Print results  for (int k=0; k < n; k++) {  System.out.println(result[k] + “ “);  }  } | Example execution code:  int [] arr = {1,2,3,4,5};  cubeArray(arr) ;  Execution Output:  1 8 27 64 125 |

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

public void sums(int [] arr) {

int sum = 0;

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

int j;

for (j=1, sum=arr[0]; j <= i; j++) {

sum += arr[j]

}

System.out.println(i + ", " j + ", " + sum);

}

}