Assignment 2

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

**Section 1: Pen-and-paper Exercises**

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

int i = 1; // 1

while (i <= n)

some O(1) time statements;

i = i\*2; //log(n)

end while

Note: Credit will not be given only for answers - show all your work:

(3 points) steps you took to get your answer. The log(n) is the highest degree n so:

(2 points) your answer. O(log(n))

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



Note: Credit will not be given only for answers - show all your work:

(5 points) steps you took to get your answer. (1/log(n)) \* (log(n)) \* (n/2) = n

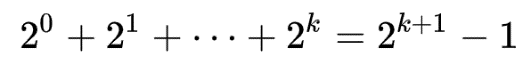
(1/log(n)) \* (log(n)) cancel leaving (n/2) so O(n)

(2 points) your answer. O(n)

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



Note:



Note: Credit will not be given only for answers - show all your work:

(5 points) steps you took to get your answer. (n) \* (log(n)) \* (2^n) = 2^n

(2 points) your answer. O(2^n) because it is the highest order.

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

**j = 1, i = 0;**

**while (i < n)**

**{**

**i = i + j;**

**j++;**

**}**

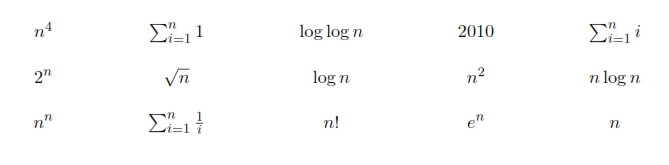
Note: The loop variable ‘i’ is incremented by 1, 2, 3, 4, … until i becomes greater than or equal to n.

Note: Credit will not be given only for answers - show all your work:

(5 points) steps you took to get your answer. *i* is x(x+1)/2 after x iterations and if the loop runs x times then x(x+1)/2 < n which means the time complexity can be written as O.

(2 points) your answer. O.

1. Arrange the following functions in ascending order of growth rate (8 points):



You are NOT required to justify your ordering.

Note:

In this problem, you are asked to identify if f1(n) < f2(n) for a “sufficiently large” input size n. However, for small values of n this is not always true.

, 2010, log(log(*n*)), log(*n*), ,*n*, , *n* log(*n*), , ,,,, *n*!

1. Given a positive integer x, find square root of it. If x is not a perfect square, then return floor (round down).

Examples:

Input: x = 4

Output: 2

Input: x = 11

Output: 3

Outline an algorithm for finding square root of x. Expected in O(log n) time.

**Full credit (10 points) will be awarded for an algorithm that is O(log n). Algorithms that are O(n) or slower will be scored out of 5 points.**

**Note: You should NOT use existing functions like math.sqrt() to obtain the square root of x. Create your own function. Solutions that use existing functions will receive 0 points.**

**(i) describe the idea behind your algorithm in English (2 points);**

Start with 0 and end with x. if x is equal to 0 or 1 then return x as the answer. While the start value is greater than or equal to the end value (x) do this: find middle by adding the start and end then dividing by 2, compare middle squared to x and if they are equal then return the middle since x is a perfect square. If x is greater than middle squared do a binary search between the middle+1 and the end, if x is less than middle squared do a binary search between the start and the middle. At the end, return the answer.

**(ii) provide pseudocode (5 points);**

Start is 1, end is x

While start <= end

Middle is (start = end)/2

If middle squared is x, return middle //perfect square

If middle squared is < x, start is middle + 1 and answer is middle

Else end=middle – 1

Return the answer

**(iii) analyze its running time (3 points).**

it would be O(log n) since it isn’t exponential but rather halving each iteration. (n 🡪 n/2 🡪 n/4 🡪 n/8 🡪 n/16 and so on) We also know that Binary search is O(log n) time complexity.

**Section 2: Java Implementation**

1. Implement problem 6 in Java (30 points).

Note:

Find a file called Problem6.java in assignment 2 folder.

Complete the method of squareroot().

Test your method in the main method provided.

**Programs that are O(n) or slower will be scored out of 10 points.**

**Programs that use existing functions like math.sqrt() will receive 0 points.**

**Important: In all of the assignments of this course, when you are asked to implement an algorithm for a problem, your code will be evaluated based on:**

**5 points - Execution**

**Each file must run without error or warning on valid input described in the main method provided.**

**5 points - Within Code Documentation**

**Is the code documented for obvious understanding of the use, preconditions, and postconditions of each function?**

**20 points - Correctness**

**Is the algorithm implemented correctly? Does your method pass the test?**

**TURN-IN CHECKLIST:**

1. **Answers to Section 1 (.doc/.txt), and to Section 2 (all your source Code (.java files)). Remember to include your name, the date, and the course number in comments near the beginning of your code/report.**
2. **Create a folder and name it 'FirstName\_LastName\_assignment\_2'. In the newly created folder copy and paste your files (.doc/.txt/.java files). Then compress the folder, and push it to iLearn.**