HW10: INTRODUCTION TO THE THEORY OF COMPUTATION.

If Yoda was a Pokemone, he would definitely be a psychic type. —Anonymous

Course: CS 5002
Fall 2021
Due: Saturday, December 11, 2021

PROBLEMS

Problem 1: Order of Growth

Rank these functions in terms of order of growth. That is, find an arrangement of functions $g_1, g_2, \ldots g_n$ such that $g_1 = \Omega(g_2), g_2 = \Omega(g_3), \ldots$.

When multiple functions have the same order, please indicate so. (5 points)

1. $\lg(\lg n)$
2. n^3
3. $n!$
4. $(\frac{3}{2})^n$
5. $2^{\lg n}$
6. $(n+1)!$
7. $\ln n$
8. $n \lg n$
9. 2^{2^n}
10. $n^2 + \log n$

Some simplification:
5. $2^{lgn} = n$
6. $(n+1)! = n!$
10. $n^2 + logn = \theta(n^2)$
The ordering of the functions is below:
$9 > 6 = 3 > 4 > 2 > 10 > 8 > 5 > 7 > 1$

Problem 2: O, Ω **and** Θ

For each function $f(n)$ below, indicate whether the function $f(n)$ is $O(g(n))$, $\Omega(g(n))$, or $\Theta(g(n))$, where $g(n) = n \lg n$.

(a) (1 point) $5x + 7$

$f(n) = 5n + 7 = O(g(n))$

(b) (1 point) $(x^2+3)/x$

$$f(n) = \frac{n^2+3}{n} = \theta(n) = O(g(n))$$

(c) (1 point) $(x^3+3x)/(5x+2)$

$$f(n) = \frac{n^3+3n}{5n+2} = \theta(n^2) = \Omega(g(n))$$

(d) (1 point) $2x^3$

$$f(n) = 2n^3 = \theta(n^3) = \Omega(g(n))$$

(e) (1 point) $2^x \lg x$

$$f(n) = 2^n lgn = \Omega(g(n))$$

Problem 3: Algorithm I

Consider the following algorithm, represented in pseudocode.

```
COMPUTE($n$)
1  $x = 0$
2  $y = 1$
3  for $i = 1$ to $n$
4      $x = y$
5      $y = y + i$
6  return $x + y$
```

(a) (2 points) Describe what the algorithm is doing in "natural language".

(a)
The algorithm is looping from 1 to n, so there are total of n iteration. In each iteration, y is added by the index of that iteration, so y is initially 1, and after the while loop, y is $1 + \frac{(1+n)n}{2}$. In each iteration, x is assigned to the previous value of y, so after the last iteration, x is $1 + \frac{n(n-1)}{2}$. Then, the final value returned by the function is the sum of x and y, which is $2 + \frac{n^2}{2}$.

(b) (3 points) What is the runtime of this algorithm? Use Big-Θ notation.

(b)
For this algorithm, there is only a for loop iterating from 1 to n, and during each iteration, only constant operations are performed. Therefore, the big-theta runtime of this algorithm is $\theta(n)$

Problem 4: Algorithm II

This question pertains to the following function.

```
ENIGMA($n$)
$r = 1$
for $i = 0$ to $n - 1$
    for $j = 1$ to $n$
        for $k = 1$ to $j$
            $r = r \cdot 2$
return r
```

(a) (3 points) What value is returned by this function? Express your answer as a function of n.

(a)
From i = 0 to n – 1, there are n iterations.
The number of iterations of k is depending on the value of j, so k is from 1 to n, there is a total of $\frac{(n+1)n}{2}$ iterations.
In the inner loop, r is multiplied by 2 every time. Therefore, r will become:

$$r = 2^{n^2\frac{n+1}{2}}$$

(b) (2 points) What is the worst-case running time, using Big-O?

(b)
In the worst case, the function will iterate every loop, therefore, the big-O run time complexity is $O(n^3)$.

Problem 5: Algorithm III

Write pseudocode to implement the algorithm described below:

Input: an array

Output: a sorted array

For every element in an array, if the previous element is less than the current element, move to the next element. Otherwise, swap the elements and move to the previous element. If there is no previous element, just move to the next element. If there is no next element, you're done.

(a) (6 points) Write pseudocode to express the algorithm.

(a)
Assume the array is zero-based.
The algorithm is insertion sort.

```
sort (A)
        i = 1
        while i < A.length
                key = A[i]
                j = i – 1
                while j >= 0 and A[j] > key
                        A[j + 1] = A[j]
                        j = j – 1
                A[j + 1] = key
                i = i + 1
```

(b) (2 points) What's the worst case running time of this algorithm? Provide an example of a worst-case input.

(b)
The worst-case running time happens when the element is not sorted at all, so every element is less than or equal to its previous value.
For example, an array {9, 8, 7, 6, 5, 4, 3, 2, 1} will cause the algorithm to run n^2 times, so the time complexity is $\theta(n^2)$.

(c) (2 points) What's the best case running time of this algorithm? Provide an example of a best-case input.

(c)
The best-case running time happens when the element is already sorted, so every element.
For example, an array {1, 2, 3, 4, 5, 6, 7, 8, 9} will cause the algorithm to run only n times, so the time complexity is $\theta(n)$.

Problem 6: Algorithm IV

Suppose the following algorithm is used to evaluate the polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$:

Input: Array A such element $A[i] = a_i$

```
EVALUATE($A$)
1  $p = a_0$
2  $x\_pow = 1$
3  for $i = 1$ to $n$
4      $x\_pow = x \cdot x\_pow$
5      $p = p + A[i] \cdot x\_pow$
```

(a) (3 points) Describe the worst-case input for this algorithm.

(a)
In the worst case, the for loop has run for n times, which means that the length of the input array A is n. Therefore, we need to evaluate the polynomial for all x terms from x^0 to x^n.

(b) (2 points) How many multiplications are done in the worst-case? How many additions?

(b)
In the worst case, the multiplication will be performed 2n times, and the addition will be performed n times.

(c) (2 points) How many multiplications are done on average?

(c)
On average, the multiplication is performed 2n times.

(d) (3 points) Can you improve this algorithm? If so, explain how and provide some pseudocode.

(d)
We can reduce the number of multiplication to make the algorithm more efficient.

```
$p = a_0$
for i = 1 to n
        if i is odd number, then $p = p + A[i] \cdot x \cdot (x^2)^{\frac{i-1}{2}}$
        else if i is even number, then $p = p + A[i] \cdot (x^2)^{\frac{i}{2}}$
```

Problem 7: Algorithm V

This question pertains to the following algorithm:

```
DOTHIS($A, i, j$)
if $i > j$
    return
$m = \lfloor (i + j)/2 \rfloor$
DOTHIS($A, i, m$)
DOTHIS($A, m + 1, j$)
if $A[j] < A[m]$
    $a = A[j]$
    $A[j] = A[m]$
    $A[m] = a$
DOTHIS($A, i, j - 1$)
```

(a) (3 points) Describe what this algorithm is doing; and specifically, how it is accomplishing it.

(a)
The algorithm is sorting the input array A in ascending order. The algorithm is accomplishing this through divide and conquer, dividing the array in half, and compare the last element with the middle element. If the middle element is greater than the last element, then switch their position.

(b) (2 points) Provide an example of an input that will produce the worst-case runtime.

(b)
The worst case run time will occur when the input array A is in decreasing order. For example, input array A = {9, 8, 7, 6, 5, 4, 3, 2, 1}. Dividing the array in half, middle element will be 5, so it will run with {9, 8, 7, 6, 5} and {4, 3, 2, 1} again, and it will keep going. Since every element in the last position is less than the middle element in the divided array, the swap operation will always be performed, which creates the greatest number of operations.

Problem 8: Binary Search Algorithm

Consider the sorted list of words, given below. The sorted list is used as an input to a binary search algorithm.

1. PUREE
2. PURGE
3. PURSE
4. PUTTY
5. QUACK
6. QUAKE
7. QUART
8. RELAX
9. RELAY
10. REMEN
11. REMIX
12. REPLY
13. RESTY
14. SALAD
15. SALES
16. SALON
17. SALSA
18. SCUBA
19. SEAME
20. SEARS
21. SEATS
22. SEDAN
23. TANGO
24. TAPIR
25. TARDY
26. TASER
27. TASKS
28. TAUNT
29. TAXED
30. TORTA
31. TORTS
32. TORUS
33. TOTAL
34. TOUGH
35. TRAIL
36. TRAIN
37. TROAT
38. TROUT
39. TRUCK
40. VOUGE
41. VOWEL
42. WACKO
43. WANNA
44. WANTS
45. WAVED
46. WAVES
47. WAVEY
48. WEARY
49. WEEPY
50. WHEAL

(a) (3 points) How many times is binary search called to find the word *SEDAN*? Indicate which elements are looked at in order. (for example, does the algorithm look at element 1, then 2, then 3, ...?

(a)
At the beginning, the lower bound is 1, the upper bound is 50.
In the first iteration, 1 + (50 – 1) / 2 = 25, TARDY, which is greater than SEDAN, so upper bound will become 25 – 1 = 24.
In the second iteration, 1 + (24 – 1) / 2 = 12, REPLY, which is less than SEDAN, so lower bound will become 12 + 1 = 13.
In the third iteration, 13 + (24 – 13) / 2 = 18, SCUBA, which is less than SEDAN, so lower bound will become 18 + 1 = 19.
In the fourth iteration, 19 + (24 – 19) / 2 = 21, SEATS, which is less than SEDAN, so lower bound will become 21 + 1 = 22.
In the fifth iteration, 22 + (24 – 22) / 2 = 23, TANGO, which is greater than SEDAN, so upper bound will become 23 – 1 = 22.
In the final iteration, 22 + (22 – 22) / 2 = 22, SEDAN, which is equal to SEDAN, so the binary search has finished.
Therefore, there are total of 6 times has the binary search run to find the word SEDAN.

(b) (3 points) How many times is binary search called to find the word *SCARF*? Indicate which elements are looked at in order. (for example, does the algorithm look at element 1, then 2, then 3, ...?

(b)
At the beginning, the lower bound is 1, the upper bound is 50.
In the first iteration, 1 + (50 – 1) / 2 = 25, TARDY, which is greater than SCARF, so upper bound will become 25 – 1 = 24.
In the second iteration, 1 + (24 – 1) / 2 = 12, REPLY, which is less than SCARF, so lower bound will become 12 + 1 = 13.
In the third iteration, 13 + (24 – 13) / 2 = 18, SCUBA, which is greater than SCARF, so upper bound will become 18 – 1 = 17.
In the fourth iteration, 13 + (17 – 13) / 2 = 15, SALES, which is less than SCARF, so lower bound will become 15 + 1 = 16.
In the fifth iteration, 16 + (17 – 16) / 2 = 16, SALON, which is less than SCARF, so lower bound will become 16 + 1 = 17.
In the sixth iteration, 17 + (17 – 17) / 2 = 17, SALSA, which is less than SCARF, so lower bound will become 17 + 1 = 18.
Now, the lower bound 18 is greater than the upper bound 17, so we have not found the word SCARF, and the binary search has run 6 times.

Problem 9: Recurrence Tree I

Consider the following recurrence tree, and answer the following questions about it.

$$T(n) = 2T(n/2) + n\lg n$$

(a) (2 points) Draw the recurrence tree (to at least 3 levels including the root)

(a)

Level 1

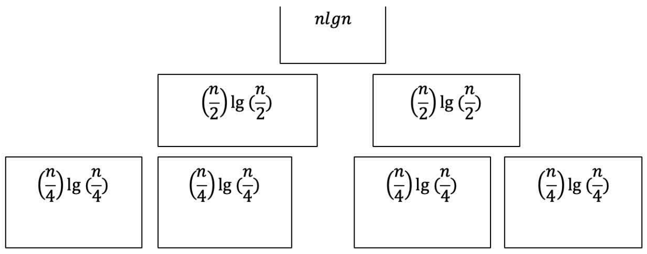
$T(n) = 2T\left(\frac{n}{2}\right) + nlgn$

Level 2

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right)$

Level 3

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right)$



$nlgn$
$\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right)$
$\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right)$
$\left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right)$
$\left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right)$
$\left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right)$
$\left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right)$

(b) (2 points) What is the total amount of work done on the 3^{rd} level of the tree (if the root is the first)?

(b)

The total amount of work done on the third level is

$T\left(\frac{n}{4}\right) = 4\left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right) = nlg\left(\frac{n}{4}\right)$

(c) (2 points) How deep is the tree?

(c)

There are total of lgn levels in the tree.

(d) (2 points) How many leaves are in the tree?

(d)

We can reduce the number of multiplication to make the algorithm more efficient.

```
p = a_0
for i = 1 to n
        if i is odd number, then $p = p + A[i] \cdot x \cdot (x^2)^{\frac{i-1}{2}}$
        else if i is even number, then $p = p + A[i] \cdot (x^2)^{\frac{i}{2}}$
```

Problem 10: Recurrence Tree II

Consider the following recurrence tree, and answer the following questions about it.

$$T(n) = 2T(n/3) + T(n/3) + n/2$$

(a) (2 points) Draw the recurrence tree (to at least 3 levels including the root)

(a)

$$T(n) = 2T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + \frac{n}{2} = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$$

Level 1

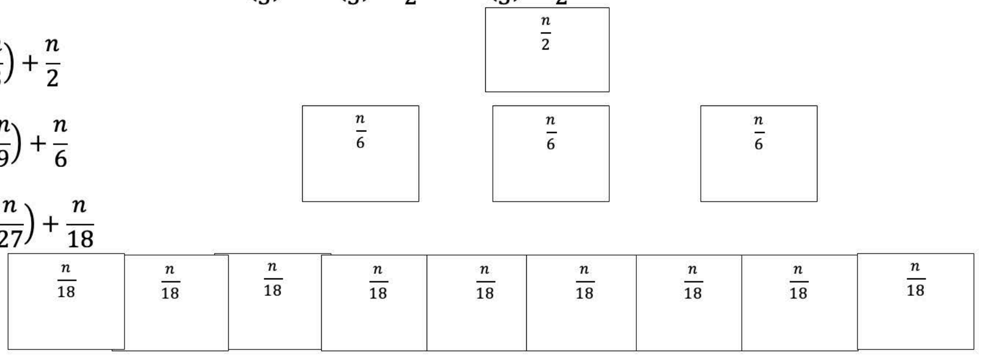
$T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$

Level 2

$T\left(\frac{n}{3}\right) = 3T\left(\frac{n}{9}\right) + \frac{n}{6}$

Level 3

$T\left(\frac{n}{9}\right) = 3T\left(\frac{n}{27}\right) + \frac{n}{18}$



(b) (2 points) What is the total amount of work done on the 3^{rd} level of the tree (if the root is the first)?

(b)
The amount of work done on the third level is:
Level 3: $\frac{9n}{18} = \frac{n}{2}$

(c) (2 points) How deep is the tree?

(c)
There is a total of $\log_3 n$ levels in the tree.

(d) (2 points) How many leaves are in the tree?

(d)
There are total of $3^{\log_3 n}$ leaves in the tree.

(e) (2 points) What is the amount of work done in the tree, EXCEPT for the leaf/bottom level? Show your work.

(e)
The total amount of work done in the tree:
$T(n) = \log_3 n \left(\frac{n}{2}\right)$

Problem 11: Singing Competition

Suppose that each person in a group of n people has to choose (vote) for exactly three people competing in a singing competition. The top three singers all win a prize as long as each receives more than $n/5$ total votes.

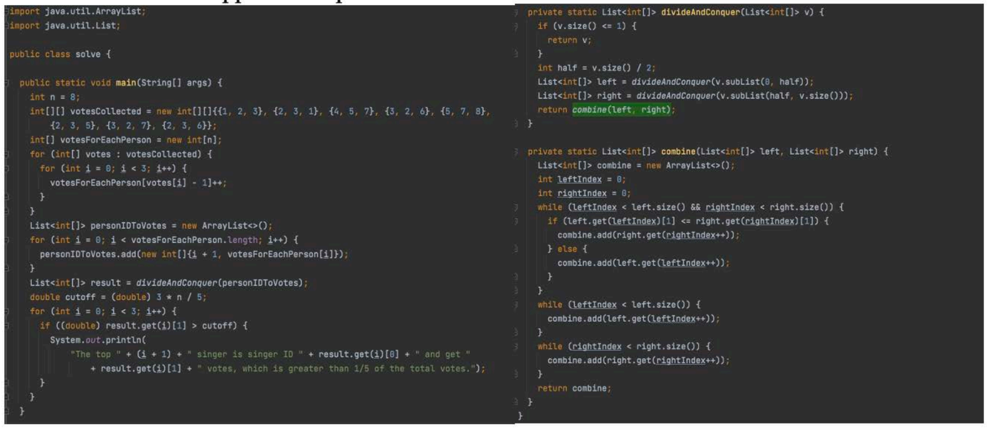
(a) (5 points) Devise a divide-and-conquer algorithm that determines whether the three singers who received the most votes each received at least $n/5$ votes and, if so, determine who these three candidates are.

(a)

Assume that the votes for each person have already been collected. I will design an algorithm like merge sort, where I will pass in the votes for all the person as a list of arrays. The first element of the array is the ID of the singer, and the second element of the array is the votes of this singer.

The first step is to divide, so we will divide the list until the list size is less than or smaller than 1. And then will combine the left and right list based on the descending votes order. So, the singer with the greatest votes will always be the front of the list. After all the list has been combined, we will obtain a list in descending votes order with their ID. Then, we will use the cutoff value 1/5 of the total votes, which is 3n/5, to check if the top three singers have votes number more than this value. If they do, then output the singer ID.

See attached code snippet for implementation in Java.



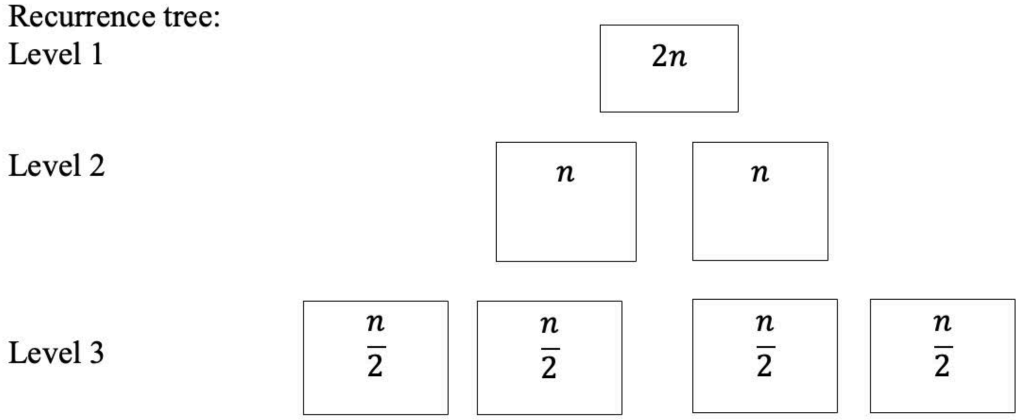
```java
import java.util.ArrayList;
import java.util.List;

public class solve {

  public static void main(String[] args) {
    int n = 8;
    int[][] votesCollected = new int[][]{{1, 2, 3}, {2, 3, 1}, {4, 5, 7}, {3, 2, 6}, {5, 7, 8},
        {2, 3, 5}, {3, 2, 7}, {2, 3, 6}};
    int[] votesForEachPerson = new int[n];
    for (int[] votes : votesCollected) {
      for (int i = 0; i < 3; i++) {
        votesForEachPerson[votes[i] - 1]++;
      }
    }
    List<int[]> personIDToVotes = new ArrayList<>();
    for (int i = 0; i < votesForEachPerson.length; i++) {
      personIDToVotes.add(new int[]{i + 1, votesForEachPerson[i]});
    }
    List<int[]> result = divideAndConquer(personIDToVotes);
    double cutoff = (double) 3 * n / 5;
    for (int i = 0; i < 3; i++) {
      if ((double) result.get(i)[1] > cutoff) {
        System.out.println(
            "The top " + (i + 1) + " singer is singer ID " + result.get(i)[0] + " and get "
                + result.get(i)[1] + " votes, which is greater than 1/5 of the total votes.");
      }
    }
  }

  private static List<int[]> divideAndConquer(List<int[]> v) {
    if (v.size() <= 1) {
      return v;
    }
    int half = v.size() / 2;
    List<int[]> left = divideAndConquer(v.subList(0, half));
    List<int[]> right = divideAndConquer(v.subList(half, v.size()));
    return combine(left, right);
  }

  private static List<int[]> combine(List<int[]> left, List<int[]> right) {
    List<int[]> combine = new ArrayList<>();
    int leftIndex = 0;
    int rightIndex = 0;
    while (leftIndex < left.size() && rightIndex < right.size()) {
      if (left.get(leftIndex)[1] <= right.get(rightIndex)[1]) {
        combine.add(right.get(rightIndex++));
      } else {
        combine.add(left.get(leftIndex++));
      }
    }
    while (leftIndex < left.size()) {
      combine.add(left.get(leftIndex++));
    }
    while (rightIndex < right.size()) {
      combine.add(right.get(rightIndex++));
    }
    return combine;
  }
}
```

(b) (5 points) Indicate the runtime of the algorithm you provided above, using recurrence trees. Please show at least 3 levels of the tree.

(b)

T(n) = 2(T/2) + 2n because we have 2 comparisons for each value when we combine.

Recurrence tree:



Using the master theorem, a = 2, b = 2, d = 1, $\frac{a}{b^d} = \frac{2}{2} = 1$, so $T(n) = \theta(nlogn)$.

Problem 12: Graph

Consider the following graph, depicted on Figure 1.

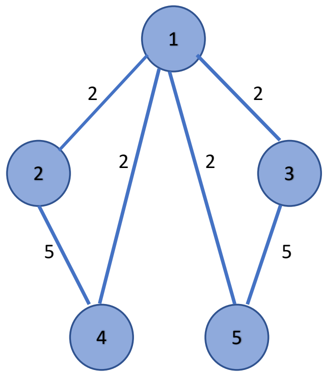


Figure 1: Graph used in Problem 12.

(a) (3 points) Represent the given graph using the **adjacency matrix**.

(a)

$$A = \begin{array}{ccccc} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{array}$$

(b) (2 points) Represent the same graph using the **adjacency list**.

(b)

1	2, 3, 4, 5
2	1, 4
3	1, 5
4	1, 2
5	1, 3

(c) (2 points) Consider the following adjacency matrix below, representing some graph $\mathcal{G}$. Please transform the given adjacency matrix into the adjacency list.

$$M(\mathcal{G}) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

(c)

1	2, 3
2	4
3	
4	5, 6
5	
6	

Problem 13: Tree

Consider the following tree, depicted on Figure 2. Please write down what gets printed when the given tree is traversed using:

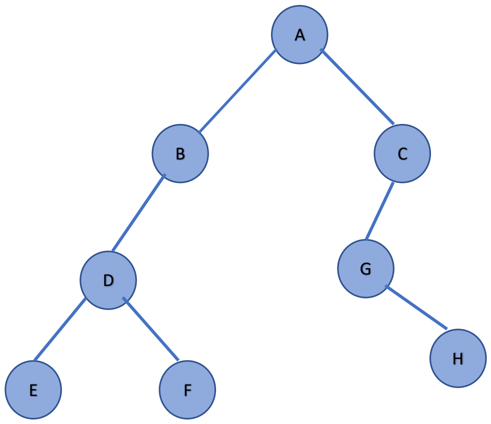


Figure 2: Graph used in Problem 13.

(a) (3 points) Pre-order traversal

(a)
Pre-order:
A, B, D, E, F, C, G, H

(b) (3 points) In-order traversal

(b)
In-order:
E, D, F, B, A, G, H, C

(c) (3 points) Post-order traversal

(c)
Post-order
E, F, D, B, H, G, C, A

Question	Points	Score
Order of Growth	5	
O, Ω and Θ	5	
Algorithm I	5	
Algorithm II	5	
Algorithm III	10	
Algorithm IV	10	
Algorithm V	5	
Binary Search Algorithm	11	
Recurrence Tree I	8	
Recurrence Tree II	10	
Singing Competition	10	
Graph	7	
Tree	9	
Total:	100	

SUBMISSION DETAILS

Things to submit:

- Please submit this assignment as a .pdf named "CS5002_[lastname]_HW10.pdf" through Canvas by 11:59pm PST on Saturday, December 11, 2021.
- Please make sure your name is in the document as well (e.g., written on the top of the first page).