

CSE 301 ASSIGNMENT 2

Fatih Goncagül

1.

```
public static void generateWeights(int[] WN, int[][] WE){
    Random random = new Random();

    //generating node weights
    for(int i=0;i<WN.length;i++){
        WN[i] = random.nextInt(20) + 1;
    }
    //generating edge weights
    for(int row=0;row<WE.length;row++){

        for (int column =0;column<WE[row].length;column++){

            if(2*row+1<WE[row].length){
                WE[row][2*row+1]=random.nextInt(20) + 1;
            }

            if(2*row+2<WE[row].length) {
                WE[row][2*row+2] = random.nextInt(20) + 1;
            }
        }
    }
}
```

2. Greedy Algorithm

```
//Greedy solution, it choses the min total weight that encounters first
public static void greedySol(int[] WN, Tree.Node node) {
    int sumL= 0;
    int sumR= 0;
    if (node!=null) {
        int totalWeight = node.nodeWeight;
        System.out.print("path: 0");

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

            int left = 2 * i + 1;
            int right = 2 * i + 2;
            int rightWeight;
            int leftWeight;

            if (node.left != null) {
                if (left < WN.length) {
                    leftWeight = node.leftEdgeWeight;
                    sumL += node.left.nodeWeight + leftWeight;
                }
                if (right < WN.length && node.right != null) {
                    rightWeight = node.rightEdgeWeight;
                    sumR += node.right.nodeWeight + rightWeight;
                }
                if (sumR>= sumL|| node.right==null) {
                    totalWeight += sumL;
                    node = node.left;
                }
            }
        }
    }
}
```

```

        System.out.print("-" + node.id);
    } else {
        totalWeight += sumR;
        node = node.right;
        System.out.print("-" + node.id);
    }
    sumR = 0;
    sumL = 0;
}
}
System.out.println("total weight: " + totalWeight + ".");
}
}

```

3. Recursive Algorithm

```

public static void recursiveSol(Tree.Node node, ArrayList<Integer>
totalList, ArrayList<String> pathList, int sum, String path) {
    //base case
    if(node == null){
        return;
    }

    sum+= node.nodeWeight;
    path+=node.id;
    if(node.left!=null||node.right!=null) {
        path += "-";
    }
    //base case
    if (isLeaf(node)) {
        totalList.add(sum);
        pathList.add(path);
        return;
    }

    if(node.left!=null){
        sum+=node.leftEdgeWeight;
        recursiveSol(node.left,totalList, pathList,sum,path);
    }

    sum-=node.leftEdgeWeight;

    if(node.right != null){
        sum+=node.rightEdgeWeight;
        recursiveSol(node.right,totalList, pathList,sum,path);
    }

    sum-=node.rightEdgeWeight;
    if(sum==node.nodeWeight){
        int index=minIndex(totalList);
        System.out.println("path: "+ pathList.get(index)
            +" , total weight "+totalList.get(index));
    }
}
}

```

First of all simple assigning statements will take $O(1)$ we won't be dealing with those. So where we use recursion will take $T(X-1)$. height of the tree will get shorter. $X = \text{HEIGHT}$.

So the recursion relation will be $T(X) = T(X-1) + T(X-1) + 1$ since we call the method itself 2 times. $T(X) = 2 * T(X-1) + 1$.

$$T(X) = 2[2T(X-2) + 1] + 1$$

$$T(X) = 2^2 T(X-2) + 2 + 1$$

$$= 2^2 [2T(X-3) + 1] + 2 + 1$$

$$T(X) = 2^3 [T(X-3)] + 4 + 2 + 1$$

$$T(X) = 2^k T(X-k) + 2^{k-1} + 2^{k-2} + \dots + 2^2 + 2 + 1$$

And assume $X - k = 0$, $X = k$

$$= 2^k T(0) + 1 + 2 + 4 + \dots + 2^{k-1}$$

We have

$$= 2^X * 1 + 1 + 2^{k-1}$$

$$= 2^X + 2^X - 1$$

$$= 2^{X+1} - 1$$

So this is $O(2^X)$

I said $X = \text{Height}$ which is $X = \log n$

$O(2^{\log n})$ and this equals to $O(n)$.

4. Dynamic Programming

First we need to think about base cases, then we need to think of subproblems reduced after that we can decompose towards our goal which is finding minimum total weight path from the root to anyone of the leaves.

```
static void dynamicSol( Tree.Node node, int a, int[] array) {

    array[a] = node.nodeWeight;
    int min = Integer.MAX_VALUE;

    if (node.left != null) {
        dynamicSol(node.left, node.left.id, array);
        min = Math.min(min, (array[node.left.id] + node.leftEdgeWeight));
    }

    if (node.right != null) {
        dynamicSol(node.right, node.right.id, array);
        min = Math.min(min, (array[node.right.id] + node.rightEdgeWeight));
    }

    if (node.left == null && node.right == null) {
        min = 0;
    }

    array[a] += min;
}
```

So this will be like the previous one. The height will get shorter we will have the same recurrence relation.

$$T(X) = 2 * T(X-1) + 1.$$

$$T(X) = 2[2T(X-2)+1]+1$$

$$T(X) = 2^2T(X-2) + 2 + 1$$

$$= 2^2[2T(X-3)+1] + 2 + 1$$

$$T(X) = 2^3[T(X-3)] + 4 + 2 + 1$$

$$T(X) = 2^k T(X-k) + 2^{k-1} + 2^{k-2} + \dots + 2^2 + 2 + 1$$

And assume $X - k = 0$, $X = k$

$$= 2^k T(0) + 1 + 2 + 4 + \dots + 2^{k-1}$$

We have

$$= 2^k * 1 + 1 + 2^k - 1$$

$$= 2^k + 2^k - 1$$

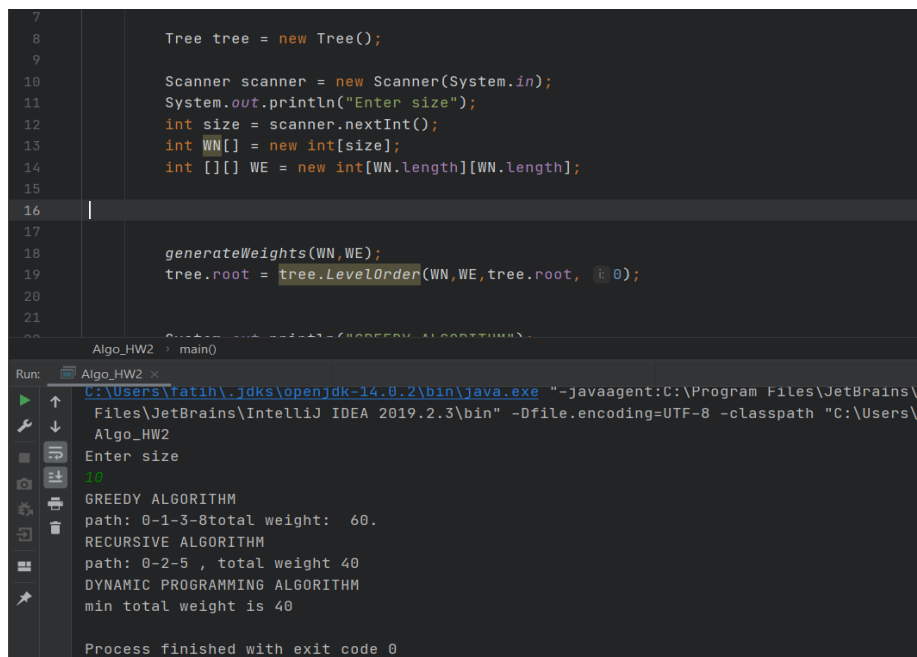
$$= 2^{k+1} - 1$$

So this is $O(2^X)$

I said $X = \text{Height}$ which is $X = \log n$

$O(2^{\log n})$ and this equals to $O(n)$.

5. A.



```

7
8      Tree tree = new Tree();
9
10     Scanner scanner = new Scanner(System.in);
11     System.out.println("Enter size");
12     int size = scanner.nextInt();
13     int WN[] = new int[size];
14     int [][] WE = new int[WN.length][WN.length];
15
16
17
18     generateWeights(WN,WE);
19     tree.root = tree.LevelOrder(WN,WE,tree.root, 0);
20
21
22     System.out.println("GREEDY ALGORITHM");
23
24     Algo_HW2 main0
25
26 Run: Algo_HW2 x
27 C:\Users\Fatih\jdk\openjdk-14.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\I
28 Files\JetBrains\IntelliJ IDEA 2019.2.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Users\F
29 Algo_HW2
30 Enter size
31 20
32 GREEDY ALGORITHM
33 path: 0-1-3-total weight: 60.
34 RECURSIVE ALGORITHM
35 path: 0-2-5 , total weight 40
36 DYNAMIC PROGRAMMING ALGORITHM
37 min total weight is 40
38
39 Process finished with exit code 0

```

Since greedy chooses greedily it does not solve this problem optimally. Greedy algorithm does not compare all paths, it chooses what comes first as minimum weight. Above picture shows that there are another paths which costs less than the path greedy algorithm found!

B.

Input size	Dynamic Programming(ms)	Recursive(ms)
50	0	16
5000	0	20
20000	0	30

Because of the memorization in dynamic programming solution our algorithm speeds up. In recursion we are doing calculations we don't need so this is why it is slower.