Homework-3

CS624 -Analysis of algorithms

SAICHARITHA DATHRIKA

02010404

1. Let there be two input arrays X [1…n] and Y[1…n] each has n numbers already in sorted order.
   Algorithm:
   - We first find the median m1 of array X[1..n] and median m2 of array Y[1..n].
   - If median m1 of X[] and median m2 of Y[] are equal then the algorithm returns m1 or m2.
   - If m1>m2 then median is either in subarray X[1…n/2] or in subarray Y[n/2…n]
   - If m1<m2 then median is either in subarray X [n/2….n] or in subarray Y[1….n/2]
   - Repeat the same above steps till the length of both subarrays is 2.
   - Then if the two arrays X [] and Y [] is 2 then use the average(max(X[1],Y[1])+min(X[2],Y[2]))

   Example:

   X [] = [1, 4, 8, 12, 18]

   Y [] = [2, 8, 10, 13, 15]

   m1=8 and m2= 10

   m1<m2 so the median is in [8,12,18] and [2, 8, 10]

   now m1= 12 and m2= 8

   m1>m2 so the median is in [8, 12] and [8, 10]

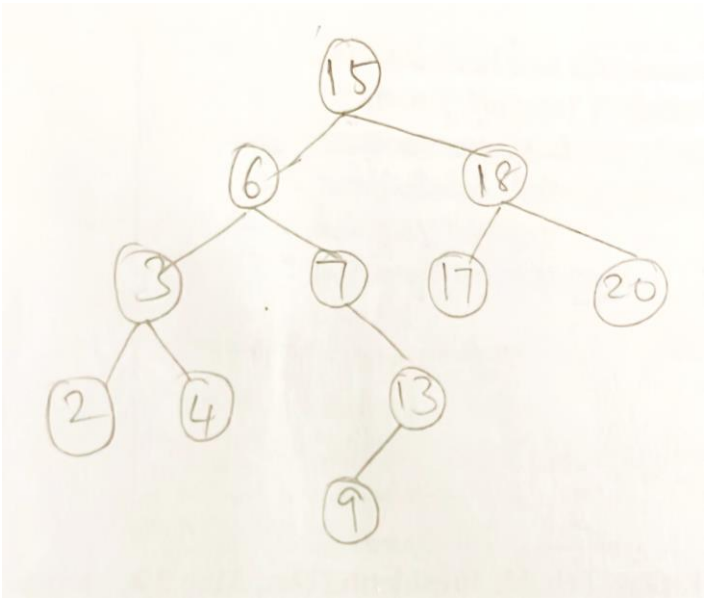
   now the size is 2

   so median = (max(8,12)+ min(8,10))/2 = (12+8)/2 =10

   ➢ Time complexity for this algorithm is O (log n)

2. If the node has two children it means that the node has both a left subtree and a right subtree.

   Suppose consider the below binary tree



   Let that node be 6 which has two children.

   The predecessor of node 6 is 4 and its successor is 7. We can see that node 4 has no right child and node 7 has no left child.

   In general (by contradiction proof)

   if the node n has predecessor p and if this predecessor p had a right child, then this child should be greater than p and less than n if this is true then p's right child will be n's predecessor.

   Similarly, if node n has successor s and if this successor s had a left child, then this child should be less than s and greater than n if this is true then s's left child will be n's successor.

   Hence, by proof by contradiction, we can say that the given question is true.

3. Finding the length of the longest subsequence of a given sequence that has all its components arranged in increasing order is known as the Longest monotonically Increasing Subsequence (LIS) problem.

   By using the recursive algorithm, we get overlapping subproblems

   By using the dynamic programming, we can avoid this overlapping subproblems.

Let there be an input array a[] which ranges from 1 to n-1 and output array called LMIS[] which ranges from 1 to n-1

The output array is used to store the length of LMIS ending at a[i].

Algorithm:

```
LMIS( a, n):
a [] – LMIS[n]
LMIS[0] = 1
for i= 1 to n-1
        LMIS[i] = 1
        for (j= 1 to i-1)
                if ( a[j]<a[i] & LMIS[i]< (LMIS[j]+1) )
                        LMIS[i] =LMIS[j]+1
return max(LMIS[i])
```

example :

let a[]= {4, 11, 3, 12}

and initially let LMIS[] = [1, 1, 1, 1]

1. a[2]is greater than a[1] so LMIS[2]= max(LMIS[2], LMIS[1]+1) = 2
2. a[3] is less than a[1] so don't change
3. a[3] is less than a[2] so no change
4. a[4] is greater than a[1] so LMIS[4] = max(LMIS[4], LMIS[1]+1) = 2
5. a[4] is greater than a[2] so LMIS[4] = max(LMIS[4], LMIS[2]+1) = 3
6. a[4] is greater than a[3] so LMIS[4] = max(LMIS[4], LMIS[3]+1) = 3

Time complexity for this algorithm using dynamic programming is $O(n^2)$ as we used the nested loop.

4. Let n = number of items

   W = maximum weight of items that thief can put in his knapsack.

   Let maximum weight be M

   And array of weight W[i] and corresponding weight value V[i]

   Example:

   Let W[i] = [2, 3, 4, 5], V[i] = [1, 3, 5, 6], M = 9 and n = 4

   We can build a table of options B using recursive formula.

   This table B has n+1 rows and M+1 columns

   - By dynamic programming row 0 includes all zeros.

- Also 0^{th} column includes all zeros.
- Using recursive formula to calculate each cell values
- When calculating table of options, we need to get B[n][M] which is maximum value with weight M=9
- On calculating we will get the below table:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	3	3	4	4	4	4
3	0	0	1	3	5	5	7	8	8
4	0	0	1	3	5	7	7	8	9

1. first we create table B[][]
2. if you do not select package i calculate B[i][j].
3. if you select package i then reset B[i][j]

Now we must write down {x1, x2, x3, x4} values

To find these answers we will take sequence of decision
we have kept the data table ready.
and we must decide which objects to be included and which should not be included in the bag.
M=9 is in 4^{th} column only so include fourth object x4=1
Now what is the profit of fourth object (9-6) we get 3.
Check if 3 is in third row or not == yes
then check if 3 is in second row== yes so don't include third object.x3=0.
Check if 3 is in first row == no, so include second object x2=1.
Now remaining 2-2=0
Check if 0 is in 0^{th} row ==yes, so it means don't include object. x1=0
So, the included objects are {x1, x2, x3, x4} = {0 1 0 1}
Values for the included object are V[2] and V[4] which are 3 and 6 so that the maximum value becomes 9 when we add 3+6=9.
In this way we get maximum profit of 0 and knapsack problem.

a) 5) a) let us take an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities .

ACTIVITY	START TIME	FINISH TIME	DURATION= FINISH TIME - START TIME
1	5	7	4
2	2	7	5
3	7	10	5

We now select the activity that has least duration.

Using this method of least duration time for selecting activity we got the solution as Activity 1 and the optimised activity should be either 2 or 3.

So, we can say that this approach does not work.

b) Let us take an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities.

No. of overlaps is parallel occurrence between start and finish time.

ACTIVITY	START TIME	FINISH TIME	NO. OF OVERLAPS
1	4	7	3
2	0	3	3
3	6	10	4
4	3	7	4
5	2	6	4
6	1	5	4

The output we got from using the least no. of overlaps is {1,2} but the answer should be {3,4,5,6}

So, we can say that this approach does not work.

c) Example for always selecting the compatible activity selection with the earliest start time.

ACTIVITY	START TIME	FINISH TIME
1	2	6
2	3	4
3	5	6

By least starting time activity selection we get the solution as activity 1.

But the required optimised result is activity 2 and activity 3.

6. Consider the interval on the left. We know that it must contain the leftmost point since it will be useless if it extends further to the left than the leftmost point.

As a result, we are confident that its left side is precisely the leftmost point. Therefore, since all points within a unit distance of the leftmost point are included in this single interval, we simply eliminate all of them.

When all the points have been covered, we simply repeat. This last option is the best one because it is always obvious where to place the leftmost interval at each stage.

Given a set {x1, x2, ….,x3} of points on the real line.

Let us take an sorted array A[1, …n]

Let interval array be Ai[1,……q]

The range of intervals be { i [p], i [p]+1}

Algorithm:

```
 q = 1
Ai[1] = A[1]
For p in the range 1 to n
        If(A[p]< Ai[p]+1)
                A[p] = { Ai[q], Ai[q]+1}
        Else
                q =q+1
                Ai[q] =A[p]
Return q
```

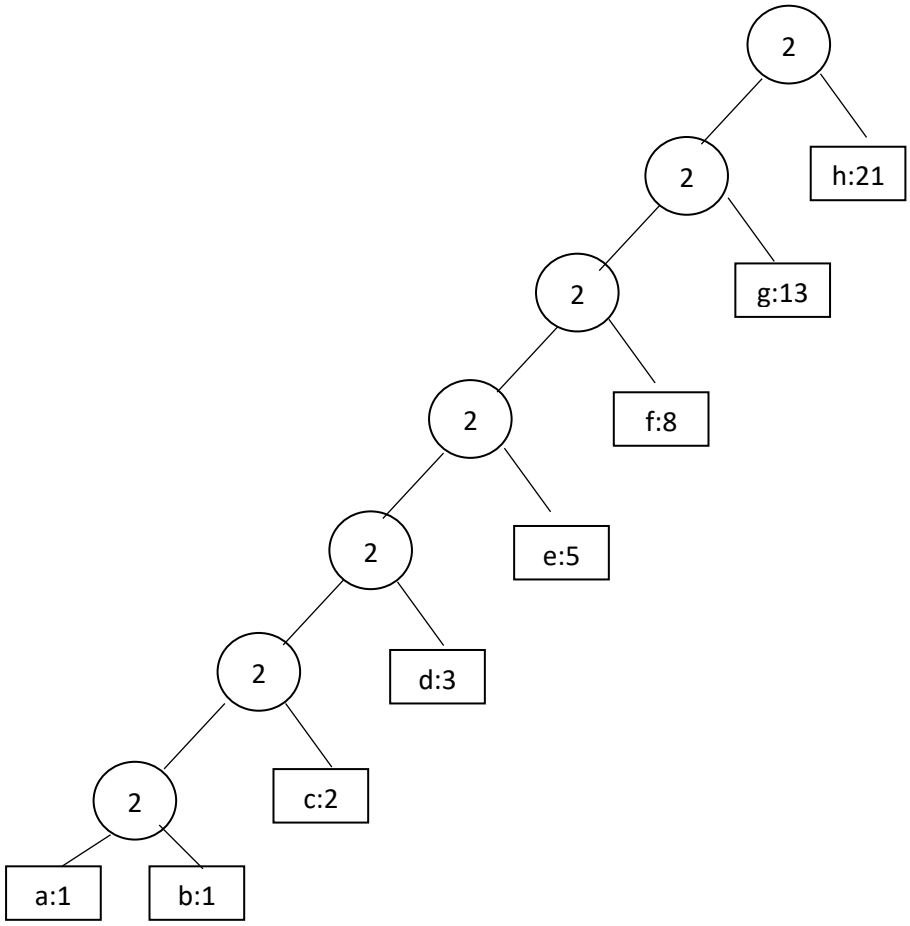
Example :

Let i = { 0.9, 0.12, 0.18, 0.52, 0.78, 1.32, 1.8, 2.15, 2.38, 2.98, 3.32, 3.45}

- i[1] = 0.9
  interval = [0.9, (0.9+1)] =[0.9, 1.9]
  this covers from i[1]=0.9 to i[7]=1.8
- i[8] = 2.15
  interval = [2.15, (2.15+1)] = [2.15, 3.15]
  this interval covers from i[8] = 2.15 to i[10] = 2.98
- i[11] = 3.32
  interval = [3.32, (3.32+1)] = [3.32, 4.32]
  this interval covers the remaining elements in the array till the leftmost point.

In this way, we cover all the points till the left most point in the array.

7. Below is the optimal Huffman code for the given first 8 Fibonacci numbers.



Symbol	Huffman code
a	000 000 0
b	000 000 1
c	000 001
d	000 01
e	000 1
F	001
G	01
h	1

Here we represented left children as 0 and right children as 1 and we assume that at each stage we are adding in the singleton as child of new parent. This is how we get our code.

Now we can generalize this first n Fibonacci numbers.

Let c1 be Nth frequent letter has codeword 0^{n-1}

Let c(i-1) be K<nth frequent letter has codeword $0^{k-1}1$ for 2<=i<=n-1

To check if this holds or not, we can prove it by below recurrence

$$\sum_{i=0}^{n-1} F(i) = F(n+1) - 1$$

We can prove this joining together the letter with smaller frequencies by induction.

Base step: for n=1 , S1 = f1<f3 obivously holds.

Now if we have n-1 >= 1 then,

$$\begin{aligned} & F(n+1) - 1 \\ = \; & F(n) + F(n-1) - 1 \\ = \; & \sum_{i=0}^{n-1} F(i) \end{aligned}$$

So, by this we can claim that at each stage we can maintain one node which contains all the least frequent letters.

For first node we see that the characters with lowest frequencies are c1 and c2.

For 2<=i<n we suppose that our claim is true for all j<I this means it has weight $\sum_{i=0}^{k-1} F(i) = F(k-1) - 1$. All other nodes at this stage have weights {F(k), F(k+1), …. F(n-1)}. Therefore, the greedy algorithm we combine nodes leaving us node containing k+1 least frequent letters for stage K+1.

Hence, we obtain the Huffman tree like the figure above.