5/1)

P[i][j]=

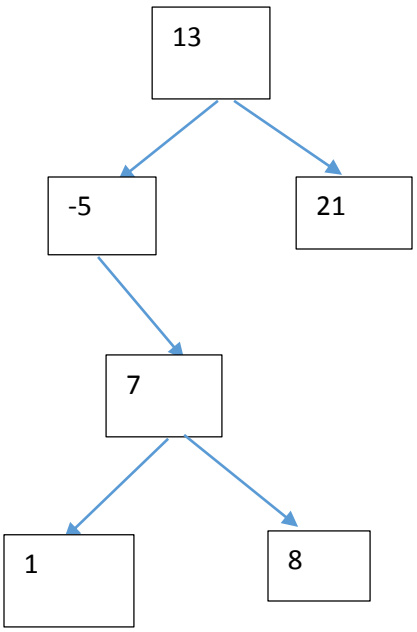
0	4	5	6	8	16	32
0	0	1	2	4	12	28
0	0	0	1	3	11	27
0	0	0	0	2	10	26
0	0	0	0	0	8	24
0	0	0	0	0	0	16
0	0	0	0	0	0	0

W[i][j]=

0	4	6	9	15	31	63
0	0	1	3	7	19	47
0	0	0	1	4	15	42
0	0	0	0	2	12	38
0	0	0	0	0	8	32
0	0	0	0	0	0	16

R[i][j]=

1	1	1	1	5	5
0	2	2	3	5	6
0	0	3	4	5	6
0	0	0	4	5	6
0	0	0	0	5	6



There are total 8 possibilities of the optimal binary search trees. Because there will be three coordinates for which value of r can be selected from any of the two.

3/2) Yes, DP approach can be used to construct the worst O_BST. Everything in the algorithm will remain same. We just have to make some changes in the probability values given. Follow the following steps:-

i)take the lcm of the denominators of all the probabilities.

Ii) multiply every given probability with that LCM and store the values in some output list.

iii) select the maximum value from the output list and subtract every value from the maximum value and hence modifying the output matrix.

iv) instead of probability values given, use the output list.

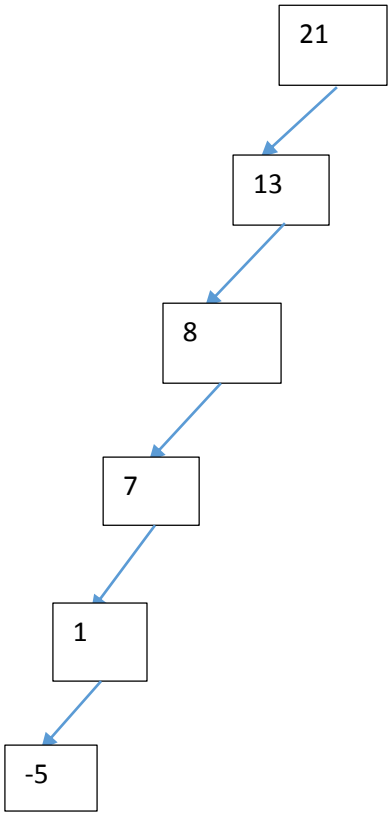
3/3) Because in Huffman coding we build bunary search tree comparing frequencies (can be compared with probabilities), but in optimal O_BST, we use keys as the comparison criteria

3/4)

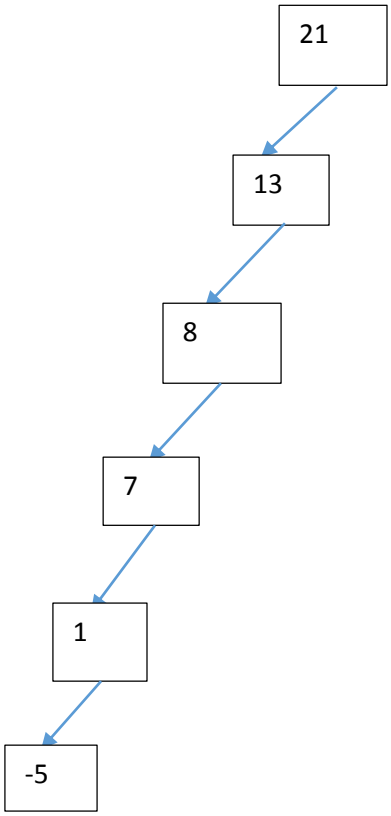
Commonalities:- Both have same search, delete operations and space operation complexities. Both follow binary search tree properties.

Differences:- Insert time in splay is O(n). But in O_BST, insertion complexity is $O(n^2)$.

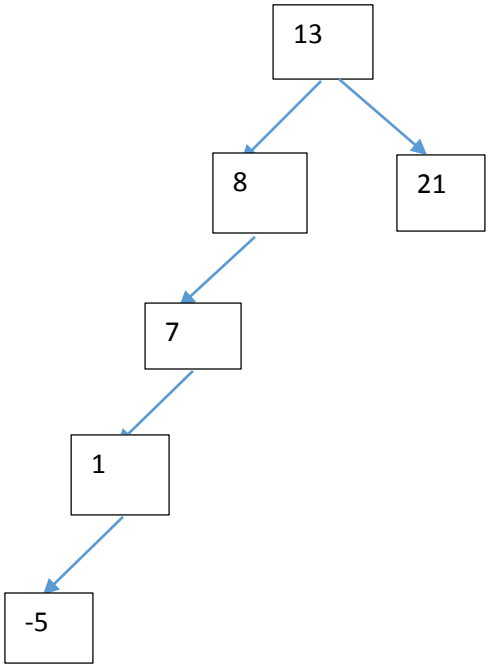
After inserting all the elements in increasing order, splay tree is:-



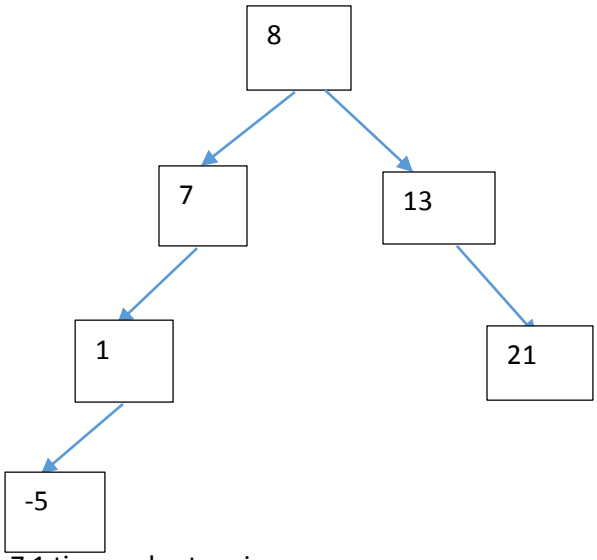
After finding elem. 21 16 times splay tree is



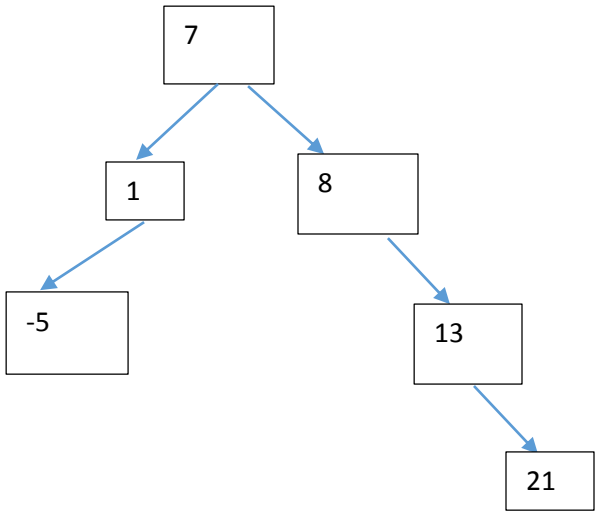
After finding 13 4 times, splay tree is:-



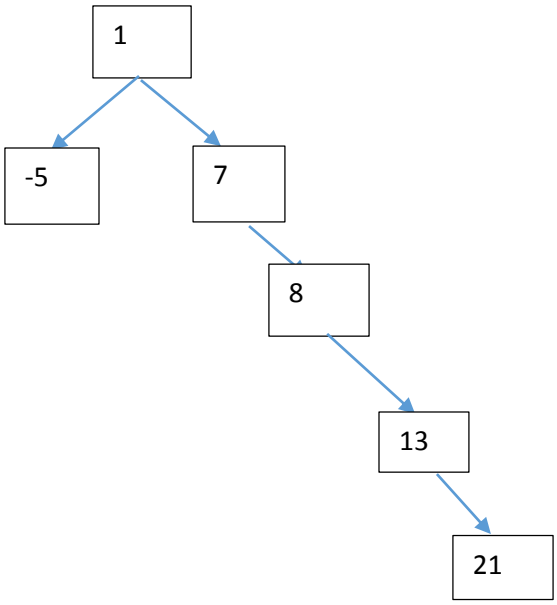
After finding 8 2 times, splay tree is:-



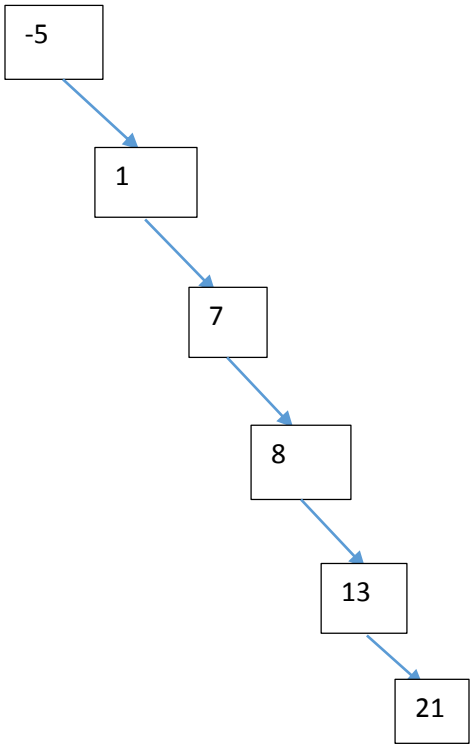
After finding 7 1 time, splay tree is :-



After finding 1 1 time, splay tree is:-



After finding -5 4 times, splay tree is:-



5/5)

(b) def swap(str1,str2):

```
    no_of_swap=0
    str1=list(str2)
    print str1
    for i in range(len(str2)):
        if(str1[i] != str2[i]):
            for j in range(i+1,len(str2)):
                if str1[j]==str2[i]:
                    str1[j],str1[i]=str1[i],str1[j]
                    no_of_swap+=1
                    break

return no_of_swap
```

5/6)

(b) In case of heterogeneous k-d tree, if we have to compare coordinate values then number of leaves to be compared will become twice. Hence total number of leaves to be compared in case of partial match query will be $n^{1/2}$.