Hamza Riaz

18L-1004

Q1:

```
a)  void findTopPlayers (Player** Playerlist, int N,
                         Player** topPlayerlist)
    {
        sort(PlayerList, N);  →  N^2
        for(int i=0; i<3; i++)  →  3
        {
            topPlayer List [i] = Playerlist [i];
        }
    }

    void sort (Player** PlayerList, int N)
    {   int largest;  →  c
        for(int i=0; i<N; i++)  →  (N-1)
        {   largest = i;  →  c
            for (int j=0; j<N; j++)  →  (N-1)
            {
        if (PlayerList[i] → getScore() > PlayerList[i] → getScore())
            {      largest = i;  →  c
            }
        }
      swap(PlayerList[i], Player List[largest])
                                        →  c
    } }
```

time complexity = $O(n^2)$.

b)

using heapsort instead of the sort function will improve the running time of the code.

```
void buildheap( Player** playerList, int N,
                                   int i)
{     int Largest = i;
   int left = 2*i + 1;
   int right = 2*i + 2;

 if (left<N && PlayerList[left]->get
                                  score()
                 >PlayerList[Largest]
                                 ->getscore())
  {    Largest = left;

if (right<N && PlayerList[right]->getscore() >
            PlayerList[left]->getscore())
         Largest = right;

  if (Largest != i)
        swap (&PlayerList[i], PlayerList[
                                 Largest]);

  buildheap( playerList, N, Largest);
}
```

```
void heapsort(Player**playerList,int N)
{

 for (int i= N/2-1; i>=0; i--)
//building heap
     buildheap (PlayerList, N, i);
 for (int i=N-1; i>0; i--)
     ~~swap [illegible]~~.
     swap (playerList[0], playerList[i]);
    buildheap (PlayerList, i,0);
 }
```

now call this heap sort in the findTopPlayers function.

The time complexity by heapsort will be O(nlogn).

Q3:

i- All codes are wrong.

According to the Huffman Algorithm the final ~~[illegible]~~ codes are:

a	01
h	000
m	10
t	001
space	11

Hence non of the agents got the codes right.

Q5:

a) CPU capacity : 2.70 GHz - 2.90 GHz
RAM : 8GB
cache : not available
virtual memory : 1280 MB

b)

Input size	Recursive (Time in ms)		
	in order	Pre order	Post order
10	1000	1400	900
50	2100	2400	2300
500	18400	20200	21400

Iterative Solution

	In order	Preorder	Post order
10	44800	38600	68700
50	183300	143500	327100
500	1380900	1392500	4972000

c)

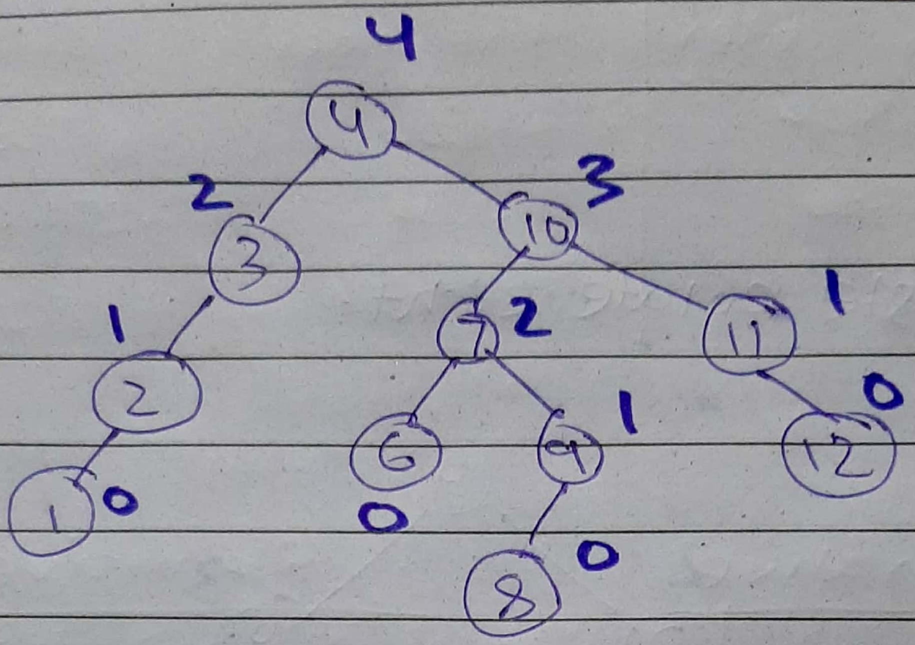
	Iterative	Recursive
Post order	O(n)	O(n)
Pre order	O(n)	O(n)
Inorder	O(n)	O(n)

d)

Recursive version takes less time than iterative one so it grows faster.

The difference in computational time is due to use of stack in iterative method. It works for the left and right child first then processes hence takes more time.

Q7 (a):



→ Node to delete has two children

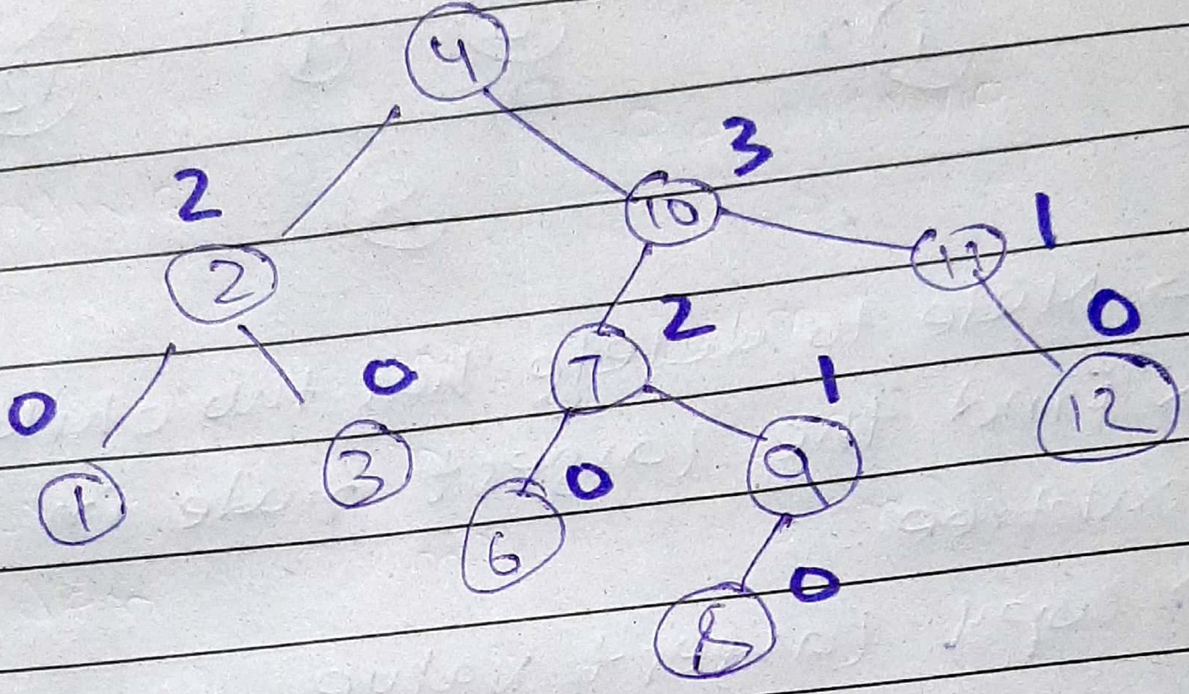
→ Find the largest node in left subtree.

→ copy largest value into node to delete.

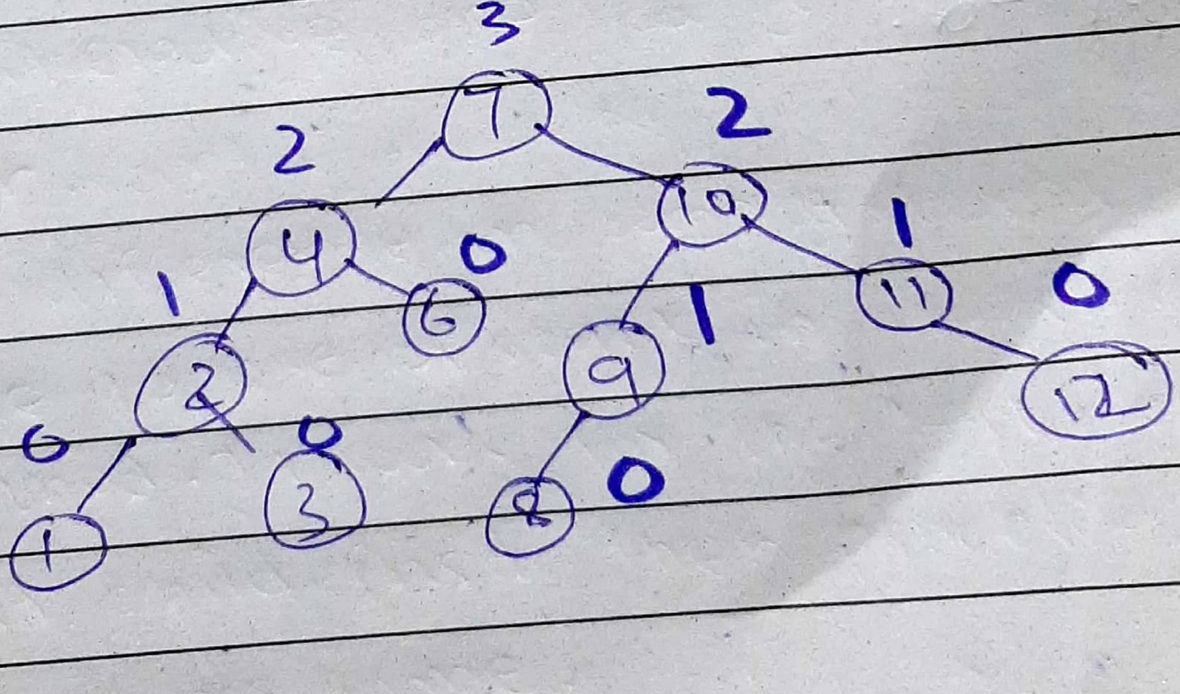
→ Remove node whose value we copied.

b)

single Rotate Right:



Double Rotate left


3
7
2
4
2
10
1
2
6
0
1
3
0
11
1
12
0
9
1
8
0