11-JAN-22

Date :

Page No.

1. Multisolver -

Heap me bnenge

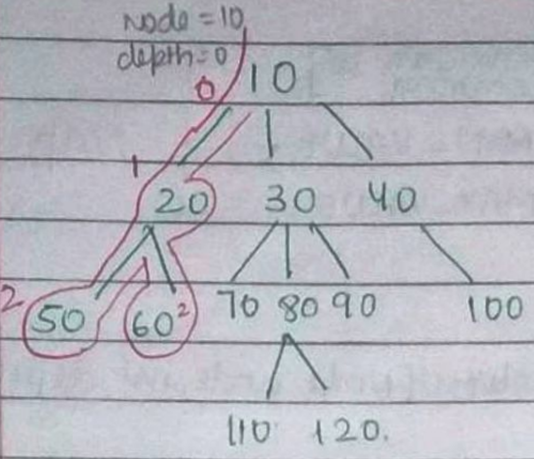
```
static int sum =0; [sum variable is common]
static int max = Integer.MIN_VALUE;
static int min = Integer.MAX_VALUE;
static int height = -1;
```

heap

```
public static void multisolver(Node node, int depth) {

    sum += node.data;
    if(node.data > max) {
        max = node.data;
    }
    if(node.data < min) {
        min = node.data;
    }
    if(depth > height) {
        height = depth;
    }
    for(Node child : node.children) {
        multisolver (child, depth + 1);
    }
}
```

[There are many depth variables]

Preorder



sum max min ht
0 -∞ ∞ -1

S : ~~0~~ ~~10~~ ~~30~~ ~~80~~ ~~140~~ ~~170~~ ~~240~~
~~320~~ ~~480~~ ~~550~~ ~~640~~ ~~680~~ 780.

max : ~~-∞~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~110~~
120.

min : ~~+∞~~ 10.

ht : ~~-1~~ ~~0~~ ~~1~~ ~~2~~ 3.

→ Euler represents stack.

| | |
|---|---|
| ~~50~~ | ~~2~~ |
| 20 | 1 |
| 10 | 0. |
| node | depth |

| | |
|---|---|
| ~~60~~ | ~~2~~ |
| ~~20~~ | ~~1~~ |
| 10 | 0 |

| | |
|---|---|
| ~~70~~ | ~~2~~ |
| 30 | 1 |
| 10 | 0 |

| | |
|---|---|
| ~~110~~ | ~~3~~ |
| 80 | 2 |
| 30 | 1 |
| 10 | 0 |

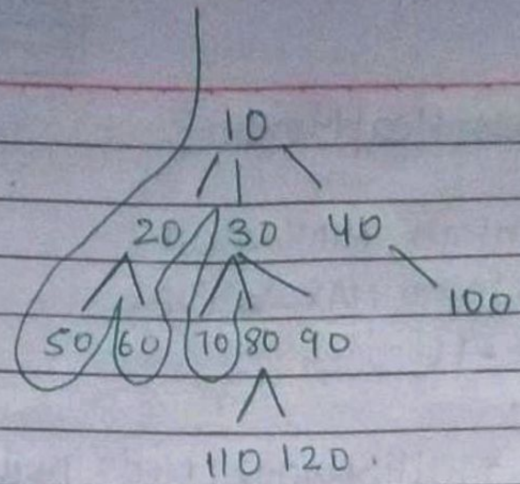
| | |
|---|---|
| ~~120~~ | ~~3~~ |
| ~~80~~ | ~~2~~ |
| 30 | 1 |
| 10 | 0 |

| | |
|---|---|
| ~~90~~ | ~~2.~~ |
| ~~30~~ | ~~1~~ |
| 10 | 0. |

| | |
|---|---|
| 100 | 2 |
| 40 | 1 |
| 10 | 0 |

With
HeapMover

```
public static class HeapMover {
      int sum = 0;
      int max = Integer MIN-VALUE;
      int min = Integer MAX_VALUE;
      int height = -1;
}
public static void MultiSolver2(Node node, int depth,
                                   HeapMover mover) {.

  mover.sum += node.data;
  if(node.data > mover. max) {.
         mover. max = node.data;
  }.
  if(node.data < mover. min) {
         mover. min = node. data;
  }.
  if(depth > mover. length){
         mover. height = depth;
  }

  for(Node child : node. children) {
        multisolver2(child, depth + 1, mover);
}

HeapMover mover = new HeapMover();
multisolver2(root, 0, mover);
southn(" sum = " + mover. sum);
southn ("Max =" + mover. max);
      ("Min = " + mover. min);
      ("Height =" + mover . height);
```



| | | |
|---|---|---|
| | | |
| | | |
| (30) | 1 | 4K |
| (10) | 0 | 4K |

S : ~~0 10 50 80 140 170~~ 240

max : ~~-∞ 10 20 80 60~~ 70

min : ~~+∞~~ 10

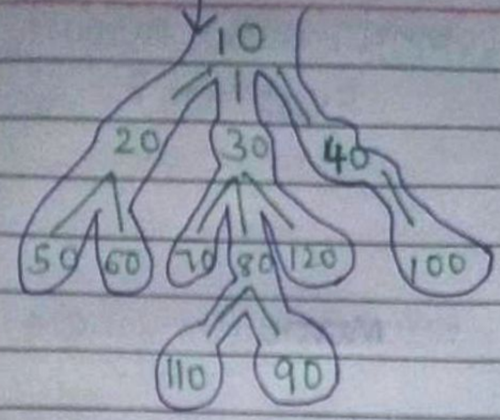
ht ~~4 0 1~~ 2

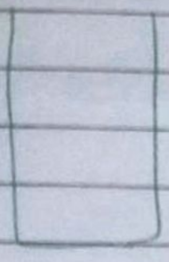
4K

(movero obj)

2. <u>Second largest</u>

```
static int largest = Integer.MIN_VALUE;
static int slargest = Integer.MIN_VALUE;
public static void secondLargest(Node node) {
        if(node.data >= largest){
                slargest = largest;
                largest  = node.data;
        } else if(node.data > slargest) {
                slargest = node.data;
        }.

        for(Node child : node-children) {.
                secondLargest(child);
        }.
}.
```





~~-∞~~
l = ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~110~~ 120

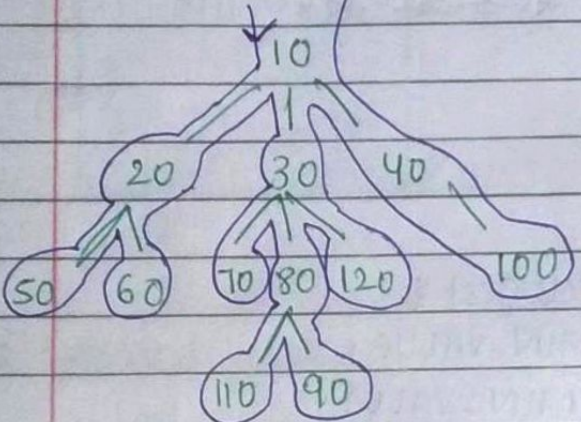
~~-∞~~ ~~90~~
sl = ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ 110

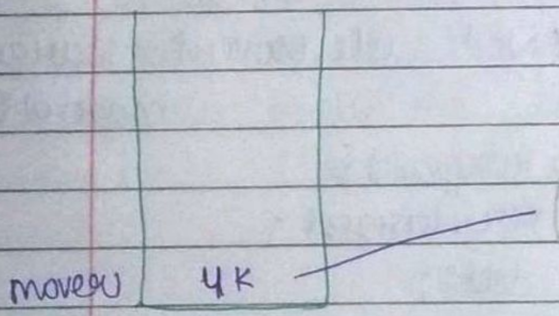
HeapMover -

```
static class MoverForSlargest {.
int largest = Integer.MIN_VALUE;
int slargest = Integer.MIN_VALUE;
}.
public static void slargest2(Node node, MoverForSlargest
                                              movero) {.
   if(node.data >= movero.largest) {.
       mover.slargest = mover.largest;
       mover.largest = node.data;
   }else if(node.data > mover.slargest) {
       mover.slargest = node.data;
   }.

   for(Node child : node.children) {
             slargest2(child, movero);
   }
}.
```

```
public static void main(string[] args) throws
                                  Exception {.

   .
   .
   .
```

(obj banaliya) 
```
MoverForSlargest mover = new MoverForSlargest();
slargest2 (root, mover);
souttn (mover.largest  +  "   " + mover. slargest);
}.
```



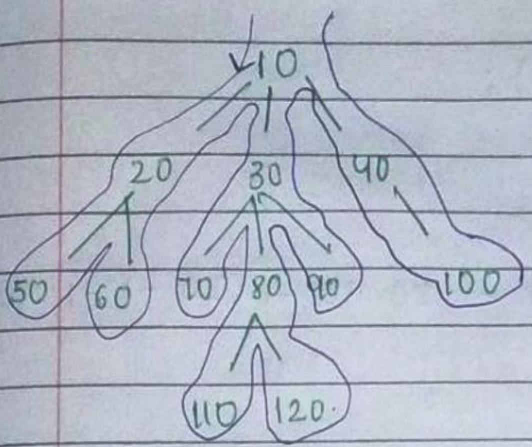


l - ~~0~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~110~~ 120
sl - ~~0~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~90~~ 110

[multisolver → sum, max, min, ht, size] 12 .
[second largest ] 12 .
[ceil & floor] 12 .
[K$^{th}$ Largest ]

## 3. Ceil & Floor

ceil - isse bdo me sbse chota ( smallest among greaters).
floor- isse chote valo me sbse bda. (largest among smallers).



⟵o $\boxed{35}$ o⟶

c ~~+∞~~ ~~50~~ 40.
f ~~-∞~~ ~~10~~ ~~20~~ 30

```
                                                   (of the largest elements)
static int ceil = Integer. MAX_VALUE; // because it is a min
static int floor = Integer. MIN-VALUE; // because it is a max
                                                   (of the smaller elements)
                void.
public static ^ceilAndFloor1(Node node, int data) {.
        if (node. data > data) {.
            // valid for ceil
            if (node. data < ceil) {.
                ceil = node. data ;
            }
        }.
        if (node. data  < data){
            // relevant for floor
            if( node. data > floor) {.
                floor = node. data;
            }.
        }.
        floor (Node child : node. children) {.
                ceilAndFloor1(child, data);
        }.
}.
```

```
int d = Integer.parseInt(br.readLine());
ceilAndFloor1(root,d);
soutln(" ceil = " + ceil);
soutln(" Floor = " + floor);
```

I/P
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90
-1 -1 40 100 -1 -1 -1

| | | |
|---|---|---|
| 35 | 5 | 125 |
| o/p | | |
| ceil = 40 | 10 | ... |
| floor = 30. | -... | 120 |

With HeapMover -

```
static class CFMover {
      int ceil = Integer.MAX_VALUE;
      int floor = Integer.MIN_VALUE;
}.

public static void ceilAndFloor2(Node node, int data,
                                 CFMover cfmover) {
    if(node.data > data) {

        if(node.data < cfmover.ceil) {
            cfmover.ceil = node.data;
        }.
    }.
```

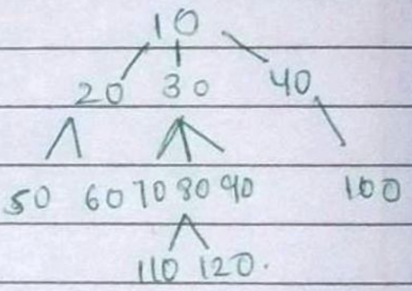
```
        if (node.data < data){.

            if (node.data > cfmover.floor){
                cfmover.floor = node.data;
            }.
        }.
        for(Node child : node.children){.
            ceilAndFloor2(child, data, cfmover);
        }.
}.
 .
 .
 .

CFMover  cfmover = new CFMover();
ceilAndFloor2(root, d, cfmover);
soutln (" ceil = " + cfmover.ceil);
soutln ("Floor = " + cfmover.floor);
```

4. Kth Largest -

```
public static void ceilAndFloor1 (Node node, int data){

}.
  .

  .

int K = Integer.parseInt(br.readLine());
int kthLargest = Integer.MAX_VALUE;
for(int i=0; i<K; i++){.
        ceilAndFloor1(root, kthLargest);
        kthLargest = floor;
     ★ floor = Integer.MIN_VALUE;
}.
soutln(K + "th Largest =" + kthLargest);
}
```

K = 3

| KL | f | |
|---|---|---|
| +∞ | ~~-∞~~ 120 ~~10 20 50 60 70 80~~ ~~40~~ | 1 |
| 120 | ~~-∞~~ 110 ~~10 20 50 60 70 80~~ | 2 |
| 110 | -∞ 100. ~~10 20 50 60 70 80 90~~ | 3. |
| 100 | -∞ | |

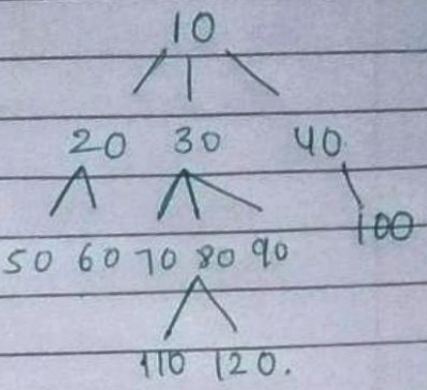


1st largest 120
2nd largest 110
3rd largest 100.
12th largest 10.

Date :

Page No.

× w/o Initializing floor.



| kl | f | |
|---|---|---|
| +∞ | ~~-∞~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~110~~ 120. | 1 |
| 120 | 120 ~~-∞~~ ~~10~~ ~~30~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~110~~ ~~90~~ ~~40~~ ~~100~~. | 2 |
| | (floor reset / update nhi kra). | |
| 120 | 120 | 3 |
| 120 | | |

heapMover

```
int K = Integer.parseInt (br.readLine ());

int KthLargest = Integer.MAX_VALUE ;
for (int  i=0 ; i<K ; i++) {.
        CFMover cfmover = new CFMover();
        cellAndFloor2 (root , KthLargest, mover);
        KthLargest = mover.floor ;
}.
soutln( K + "th largest = " + Kthlargest) ;
}
}.
```