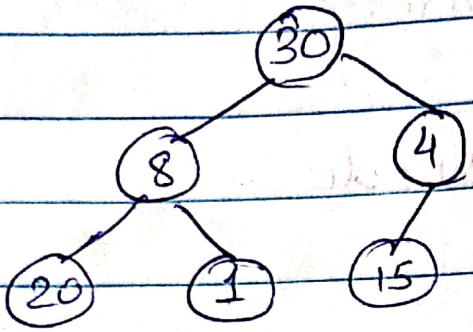
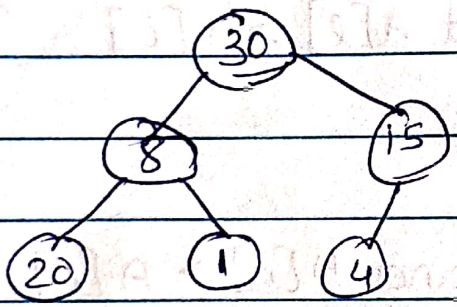
Name : Ritu. Tarun. Kuklani.
NetId : rtk304.
NYU e-mail : rtk304@nyu.edu.

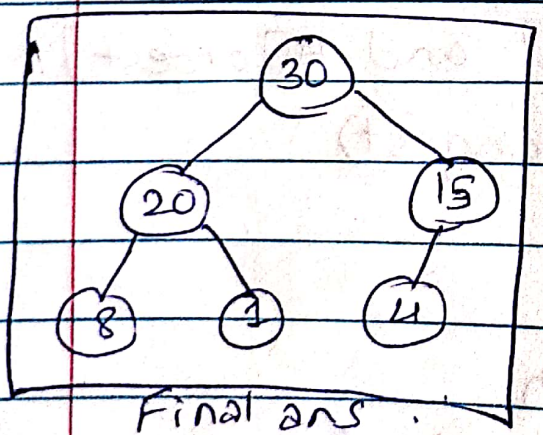
1) a) MAX_HEAPIFY (A, i)

```
l = LEFT(i)
r = RIGHT(i)
if l <= A.heapsize and A[l] > A[i]
        largest = l
else largest = i
if r <= A.heapsize and A[r] > A[i]
        largest = r
if largest ≠ i
        exchange A[i] and A[largest]
MAX_HEAPIFY (A, largest)
```

```
BUILD_MAX_HEAP(A)
A.heapsize = A.length
for i = ⌊A.length/2⌋ downto 1.
    MAX_HEAPIFY(A, i)
```





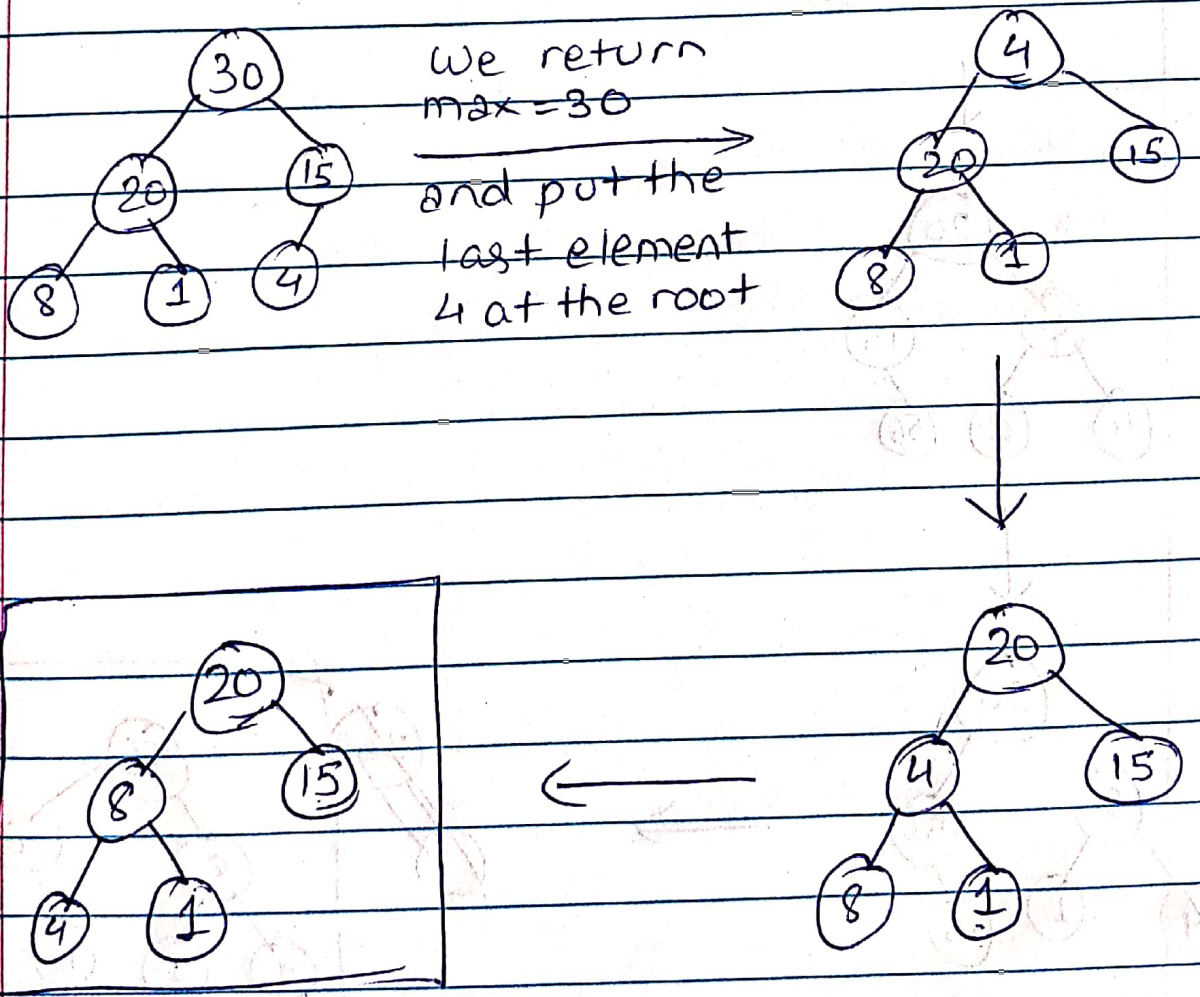


Final ans :

30	20	15	8	1	4

b) HEAP_EXTRACT_MAX(A).

```
if A.heapsize < 1
    error "heap underflow"
max = A[1]
A[1] = A[heapsize]
A.heapsize = A.heapsize - 1
MAX_HEAPIFY(A, 1)
return max
```



Final ans.

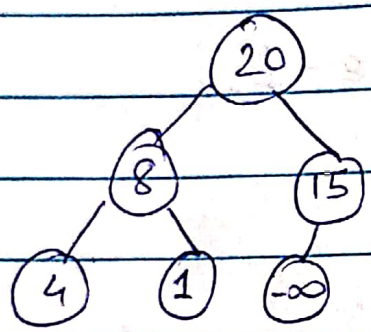
20	8	15	4	1

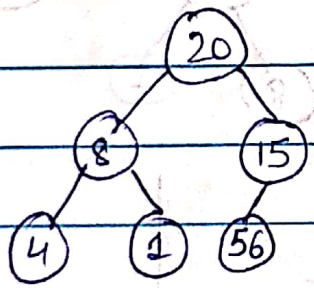
c) MAX_HEAP_INSERT(A, key).

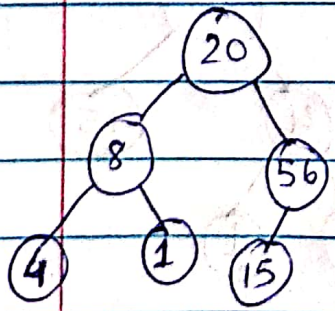
A.heapsize = A.heapsize + 1.

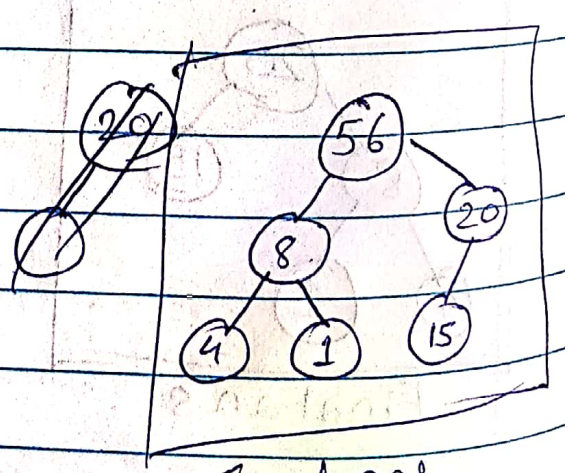
A.[A.heapsize] = -∞.

HEAP_INCREASE_KEY.









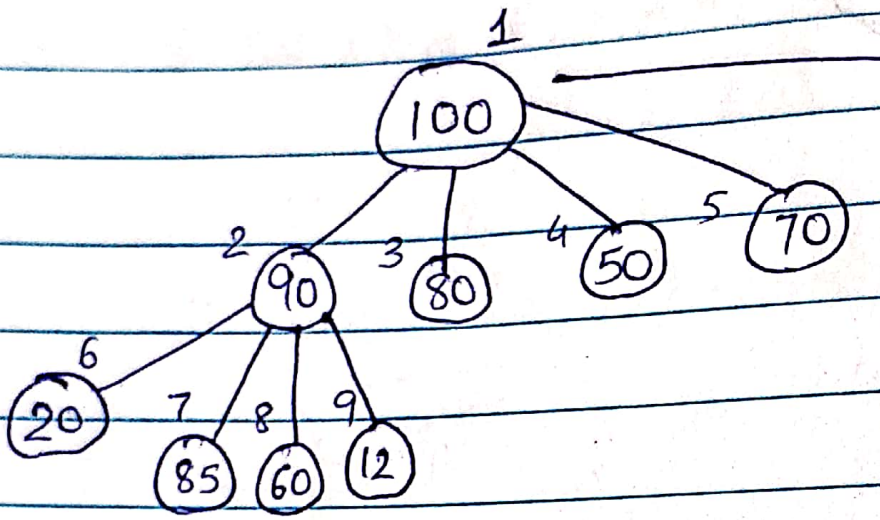
Final ans.

56	8	20	4	1	15

2.

a)

1	2	3	4	5	6	7	8	9
100	90	80	50	70	20	85	60	12



b) Consider a 4-ary max-heap.
Then the max no. of nodes at any level will depend upon height h.

Max no. of nodes = $1+4+16+64+\ldots$

which is a GP.

$$\therefore \frac{a(r^n-1)}{r-1} = \frac{1(4^{h+1}-1)}{4-1}.$$

$$n = \frac{4^{h+1}-1}{3}$$

$$3n+1 = 4^{h+1}-1.$$

$$\log_4(3n+1) = h+1.$$

$\therefore h = \lceil \log_4(3n+1) \rceil - 1.$

we consider the ceiling value because we also have to consides an almost complete tree.

c) HEAP_EXTRACT_MAX(A).

```
if A.heapsize < 1.
    error "heap underflow".
max = A[1].
A[1] = A[heapsize].
A.heapsize = A.heapsize - 1.
MAX_HEAPIFY(A, 1).
return max.
```

Running time : $O(\log_4 n)$

d) MAX_HEAP_INSERT(A, key).

```
A.heapsize = A.heapsize + 1.
A[A.heapsize] = -∞.
HEAP_INCREASE_KEY(A, A.heapsize, key)
```

Running time : $O(\log_4 n)$

e) HEAP_INCREASE_KEY (A, i, k)

```
if key < A[i]
    error "new key is smaller"
A[i] = key.
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] and A[PARENT(i)].
    i = PARENT(i).
```

Complexity : $\log_4 n$.

3 HEAP_DECREASE_KEY(A, i, key)

```
if key >= A[i]
    error "new key is larger."
A[i] = key
while i > 1 and A[PARENT(i)] ≤ A[i]
    exchange A[i] and PARENT(i).
    i = PARENT(i).
```

Initialization:
Prior to decreasing the key, the heap maintains the min-order property

Maintanence:
For every iteration, whenever the parent is greater than the key we swap the parent and the key to move a step closer to forming the min heap.

Termination:
whenever the iterator becomes lesser than one, the loop terminates because the root node has index 1 and we do not have any nodes with indices lesser than 1.

4.

```
BUILD_MIN_HEAP(A).
  A.heapsize = A.length
  for i = LA.length/2] downto 1 .
      MIN_HEAPIFY(A, i).
```

Complexity of this algorithm is O(n)

```
HEAP_EXTRACT_MIN(A).
  if heap-size < 1 .
    error "heap underflow" .
  min = A[1]
  A[1] = A[A.heapsize] .
  A.heapsize = A.heapsize - 1 .
  MIN_HEAPIFY(A, 1)
  return min
```

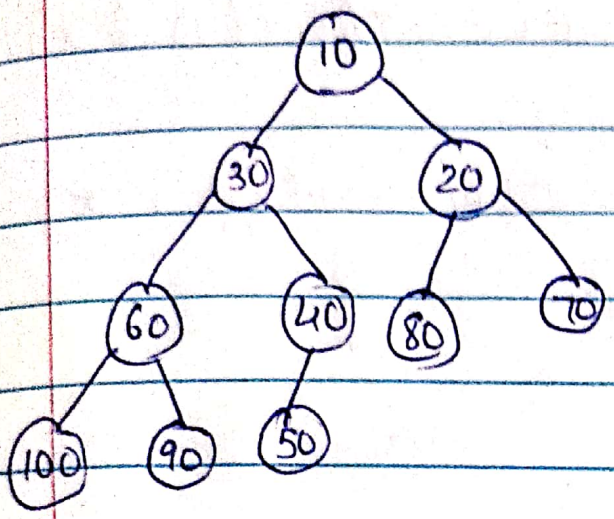
Complexity of this algorithm is O(log n).

```
sum = 0;
sum = sum + min;
```

Complexity of this algorithm is O(1).

∴ the total complexity of this algoritm is O(nlog n).

5) Given heap.



$x = 41$

Here we compare the query to every element in the tree. However, if any of the roots is greater than the query then we skip the rest of the subsequent tree.

* EXTRACT_MIN_ELEMENTS(A, i, x).

```
while (i <= A.heapsize)
    if A[i] <= x
        print A[i]
    l = 2i
    r = 2i + 1
    if A[l] <= x
        EXTRACT-MIN-ELEMENTS(A, l, x)
    if A[r] = x
```

if A[l] <= x.

EXTRACT_MIN_ELEMENT(A, l, x)

if A[r] <= x

EXTRACT_MIN_ELEMENT(A, r, x).

6) We build a min heap of k people where k is the no. of doors.
∴ the setup time will be O(k).

① BUILD_MIN_HEAP(A)
A.heapsize = k // can also be a multiple of k
for i = ⌊A.heapsize/2⌋ downto 1
MIN_HEAPIFY(A, i)

Complexity of above algorithm is O(k).

② EXTRACT_MIN(A)
if heapsize < 1
error "underflow"
min = A[1] // we give a ticket to the person having minimum time.
A[1] = A[heapsize]
A.heapsize = A.heapsize - 1.
MIN-HEAPIFY(A, 1)
return min.

The complexity of above algorithm is O(log k)

7) In a grocery store, we have to form of queue of people depending on the number of items they have. (People with lesser items will be ahead in the queue). ~~What would be the complexity~~ We have to setup the queue in O(n) time. We then have to send the people to the billing queue in O(log n) time.

Sol^n : We create a min heap and then call extract min function every time.