HASHMAP

1. HashMap<Key, Value> is a part of Java Collection.
2. This class is found in Java.util package
3. Hashmap is a datastructure which has an amazing property that most of the operations which we perform on it are done in O(1) time complexity.
   The data is stored in a hashmap in the form of key value pairs.

Country	Population (in Millions)
INDIA	1391
CHINA	1398
USA	329
INDONASIA	268

→ Population Map
∴ Key : Country (STRING)
Value : Population (INTEGER)

∴ Agar Hum Data Hashmap me store krwa rahe hai toh Data Key-value pair ki form me store hoga!

Data Pair
- Key
- value

→ ek Key aur ek value ko mila kr ek Pair Banta hai usse bolty hai Key-Value Pair

4. Hashmap is a MAP : Map is a key-value mapping, which means every key is mapped to exactly 1 value and we can use the key to retreive the corresponding value from a map.
5. Advantage of Hashmap is that time complexity to insert and retreive a value is O(1) on Average.

CREATING HASHMAP

→ It is complusary to use capital I's !

```
HashMap<String, Integer> hm = new HashMap<>();
```

- String → key's Datatype
- Integer → value's Datatype
- hm → name of the hashmap
- new HashMap<>() → like other DS, Hashmap is also created using new keyword.

{ } → This is what an empty hashmap looks like.
So, let's now try to insert some values into it and then displaying it.

PUT INTO HASHMAP → put(Key, value) function can be used to put some values into the hashmap.

Time complexity → O(1)

∴ There are 2 possible cases while we are using the put(key, value) function in a hashmap.

When the key is already present

If the key is already present, we cannot insert the same key again. (Warna humare paas duplicate ke hojayegi). (Aur hume duplicate key chaiye nahi).

→ Only the value of the existing key will get updated in the hashmap.

When the key is not present

if the key is not present in the hashmap, it will get inserted!

put
- add → Agar key phele se present nahi hai toh key add krega!
- update → Agar key phele se present hai toh value ko update krega!

Toh put function humare liye key add ka bhi kaam kr sakta hai aur key update bhi!

```
hm.put("India", 1391);
hm.put("China", 1398);
hm.put("USA", 329);
hm.put("Indonesia", 268);
System.out.println(hm);
```
→ {USA:329, China:1398, India:1391, Indonesia:268}

```
hm.put("Indonesia", 270);
System.out.println(hm);
```
→ {USA:329, China:1398, India:1391, Indonesia:270}

Indonesia:270 value updated.

★ We must observe that the hashmap has changed the order in which the elements occur.

★ So, this is something that we can not control in hashmap. We can not control the order of occurence of elements in a hashmap.

GET A VALUE IN HASHMAP → Time complexity : O(1)

The get(Key) function in a hashmap is used to get the value, corresponding to a particular key in the hashmap.

There are 2 possibilities

- **Key existed** → If the key exists, then we will get the value of that key by using the get() function.
- **Key doesn't existed** → If the key does not exist, then they get(Key) function will returns null.

```
System.out.println(hm.get("India"));
```
→ India : Key already exist in the hm hashmap with value = 1391.

```
System.out.println(hm.get("Utopia"));
```
→ The Key "Utopia" does not exist. So null value will be returned.

* We should get the values of a key only in the Integer.

Capital I Integer datatype can store null as

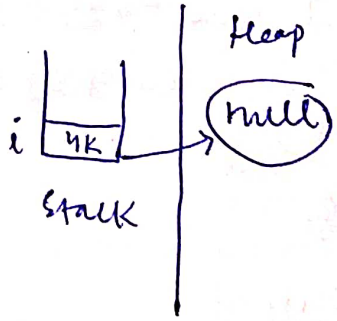
```
Integer x = hm.get("Utopia");
```
- we can store null here
- returns null

```
int x = hm.get("Utopia");
```
- int datatype can not store null so we will get a null pointer exception. → Because int is a primitives datatype.
- returns null

Primitive Type	Wrapper Class
byte	Byte
int	Integer
short	Short
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Primitive datatype ke variable stack me store hotey hai! stack me null store nahi krty!

Wrapper class objects Heap me store hotey hai!



CONTAINS KEY IN HASHMAP → Time Complexity : O(1)

The `ContainsKey()` is a Boolean function.

- we pass a particular key value as a parameter to this function. If the hashmap contains key, it returns true else return false.

```
System.out.println(hm.containskey("China"));
```

→ TRUE

we get true for the key "China" as it exists in the hashmap.

```
System.out.println(hm.containskey("Pakistan"));
```

→ False

we get false for the key "Pakistan" as the hashmap does not consist the key.

KEYSET IN HASHMAP

→ we know that we have key-value pairs in hashmap, we can get only the keys in hashmap by using the KeySet() function.

↓

The Keyset() function returns the set of all the keys in hashmap

↓

we can store it in a Set Container.

```
public static void main(String[] args) {
Hashmap<String, Integer> hm = new Hashmap<>();
hm.put("India", 1391);
hm.put("China", 1398);
hm.put("USA", 329);
hm.put("Indonesia", 268);

Set<String> key = hm.KeySet();

for(String key : keys) {
Integer val = hm.get(key);
System.out.println(key + ":" + val);
}
}
```

```
Set<String> keys = hm.keyset();
System.out.println(keys);
```

Set ek essa collection hota hai, jisme hamesha unique values hogi! Jese hashmap me key hamesa unique hoti hai!

Output:

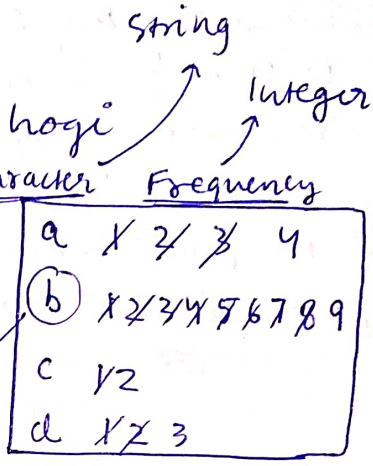
USA : 329
China : 1398
India : 1391
Indonesia : 268

HIGHEST FREQUENCY CHARACTER

Hume ek string given hogi, hume uss string me se maximum occuring character ko print krna hai!

Hume 1 Frequency map banayengy
- → Har character ki Frequency store hogi
- → Key = character
- → value = frequency

String : a b b c b a b c d d a b b d a b b b



∴ we iterate through every character of the string and then increase the count against that character!

APPROACH

(1) We create a new Hashmap of character, Integer, named hm
(character: Key's datatype; Integer: value's datatype; hm: hashmap name)

(2) We iterate through all the character of the string and check if the hashmap contains the given character. If it does, then, we get the frequency of that character & store in the old variable.

- Hum string ke saare characters par iterate krengy aur check krengy, kya hashmap me yeh character phele se exist krty hai?
- Agar hashmap me yeh character phele se exist krty hai toh hum old variable me iss character ki frequency store krengy! → old variable ka naam hai jisme character ki existing frequency hum store krengy!
- Then, we increase it by 1 and store in the new. The 'new' gets inserted in hashmap against character.

new variable me old+1 store krengy,
And then new gets inserted in the hashmap against the character.

(3) Else if, the hashmap does not contain that character then we add it to hashmap with its frequency 1.

④ We initialize a variable "max" with the first character of the given string where "max" is the character with highest frequency!

```
char max = str.charAt(0);
```

⑤ We apply the for loop for every character of the Hashmap. If the frequency of current character is greater than the frequency of "max", then that character gets store in max.

⑥ We print the max.

TIME COMPLEXITY → O(n) + O(n) = O(n)

```
public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
    Hashmap<Character,Integer> hm = new Hashmap<>();
    for(int i=0; i<str.length(); i++){
        char ch = str.charAt(i);
        if(hm.containsKey(ch){
            int old = hm.get(ch);
            int new = old+1;
            hm.put(ch,new);
        }
        else{
            hm.put(ch,1);
        }
    }
    char max = str.charAt(0);
    for(Character Key : hm.KeySet()){
        if(hm.get(Key) > hm.get(max))
            { max = Key;
        }
    }
    System.out.println(max);
}
```

Humne yeh naya hashmap banaya hm naam ka!

String ke sabhi character's pe loop lagaya!

Agar string me vo char. phele se hai toh uski value update hogi!

Agar vo character string me phele se exist nahi krta toh hum ek naya put banayengy uska char. ke liye value 1 ke sath usko hm hashmap me daaldengy!

humne saari keys ke set pr loop laga dia

KeySet() se sabhi Keys receive krengy!

agar Key ki value max ki value se zyda hai toh max me vo Key daldo

last me max. frequency wale char. ko print krdo!

LET COMMON ELEMENTS - 1

Hume 2 array given hongay (a1) and (a2)!

(a1) → 1 1 2 2 2 3 5
(a2) → 1 1 1 2 2 4 5

OUTPUT → 1 2 5

We are required to print all the elements of (a2) which are also present in (a1) (in order of their occurance in (a2))

Hume ek element 1 baar hi print krna hai!
Make sure not to print duplicate!

Toh phele hum array1 ke sabhi elements ke liye ek hashmap Banayengy!

(a1): 1 1 2 2 2 3 5
(a2): 1 1 1 2 2 4 5

keys	values
1	2
2	2 3
3	1
4	
5	1

∴ Aab Hum array2 ko traverse krengy, har element ko check krengy aur Agar vo element (a1) ke frequency map me present hoga toh hum uss element ko print krdengy! Because it is present in freq. map of array1 and array2. so it is a common element.

Now, we will remove that element from the frequency map of array1 if its present in map & array2.

(a2): 1 1 1 2 2 4 5

keys	values
~~1~~	~~2~~
2	2 3
3	1
5	1

OUTPUT: 1

Hum array2 pe Travel krengy hume 1 mila, 1 is also present in the hashmap

(a2): 1 1 1 2 2 4 5

keys	values
~~2~~	~~2 3~~
3	1
5	1

OUTPUT: 1 2

(a2): 1 1 1 2 2 4 5

keys	values
3	1
~~5~~	~~1~~

OUTPUT: 1 2 5

```
public static void main (String [] args) {
  Scanner sc = new Scanner (System.in);
  int n1 = sc.nextInt();
  int [] arr1 = new int [n1];
  for (int i=0; i<arr1.length; i++){
      arr1[i] = sc.nextInt();
  }
  int n2 = sc.nextInt();
  int [] arr2 = new int [n2];
  for (int i=0; i<arr2.length; i++) {
      arr2[i] = sc.nextInt();
  }
```

Hashmap Banaya!

```
HashMap <Integer, Integer> fmap = new HashMap<>();
int [int val : arr1] {
  if (fmap.containsKey(val) == (false)) {
      fmap.put(val, 1);
  }
  else {
      int oldfreq = fmap.get(val);
      int newfreq = oldfreq + 1;
      fmap.put(val, nf);
  }
```

array 1 ke liye frequency map banaya!

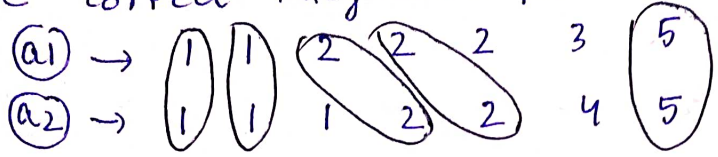
Agar val phele se fmap me hai toh old freq. +1 krdo, value aur new freq. hojayega freq. map me update!

```
for (int val : arr2) {
if (fmap : containsKey (val) == (true)) {
    System.out.println(val);
    fmap.remove(val);
  }
}
}
```

Agar array 2 ki element (val) hume freq. map me milega toh uss value (element) ko print krdo aur val ko a fmap se remove krdo!

GET COMMON ELEMENTS 2

Humne correct integration print krni hai!



output: → 1 1 2 2 5

Character	Freq.
1	✗ 2
2	✗ ✗ 3
3	1
5	1

- Phele a1 ki Freq. map banao
  Humne Hashmap Banalia a1 ke liye →
- Aab Humne a2 pr travel krna hai!
  Aur agar a2 ka koi bhi element freq. map me mila aur uss key ki value > 0 hai toh usse print krdo aur uss key ki value me se 1 minus krdo!

Character	Freq
1	0 ✗ ✗
2	1 ✗ ✗
3	1
5	0 ✗

output

1 1 2 2 5

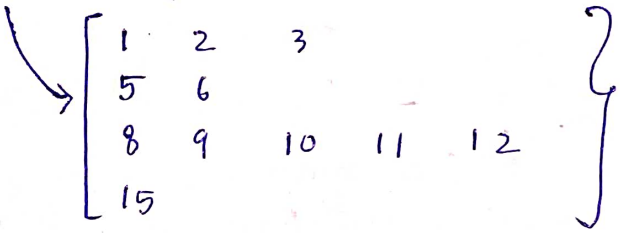
```
public static void main (String args) {
    Scanner sc = new Scanner (System.in);
    int n1 = sc.nextInt();
    int arr1[] = new int[n1];
    for( int i=0; i< arr1.length; i++) {
        arr1[i] = sc.nextInt();
    }
    int n2 = sc.nextInt();
    int arr2[] = new int [n2];
    for( int i=0; i<arr2.length; i++){
        arr2[i]= sc.nextInt();
    }
HashMap <Integer, Integer> fmap = new HashMap<>();
for (int val : arr1){
    if (fmap.containsKey(val) == false) {
        fmap.put(val, 1);
    } else {
        int oldfreq = fmap.get(val);
        int newfreq = oldfreq + 1;
        fmap.put(val, newfreq);
    }
}
for (int val : arr2) {
    if (fmap.containsKey(val)
        == true && fmap.get(val
        > 0) {
        System.out.println(val);
        int oldfreq = fmap.get(val);
        int newfreq = oldfreq - 1;
        fmap.put (val, newfreq);
    }
}
```

LONGEST CONSECUTIVE SUBSEQUENCE

Hume array given hoga!

Example: | 10 | 5 | 9 | 1 | 11 | 8 | 6 | 15 | 3 | 12 | 2 |

The following consecutive sequence can be formed from the given array



The longest sequence here is of 5 elements.

[8 — 9 — 10 — 11 — 12]

APPROACH

- Hum ek hashmap Banayengy (Integer V/s Boolean) for all the elements of the array and store "TRUE" against them all initially!

10 5 9 1 11 8 6 15 3 12 2

Keys (Integer)	Values (Boolean)
~~10~~	~~TRUE~~ F
5	TRUE
~~9~~	~~TRUE~~ F
1	TRUE
~~11~~	~~TRUE~~ F
8	TRUE
~~6~~	~~TRUE~~ F
15	TRUE
~~3~~	~~TRUE~~ F
~~12~~	~~TRUE~~ F
~~2~~	~~TRUE~~ F

- The true here represents that its corresponding element is the starting element of the desired sequence.
- Ek loop lagaya array ke upper aur sabki elements ko hashmap me store krdia as keys and unn sabki values TRUE daal di!
- Aab Hum pure ko iterate krengy loop lagake and we will check whether a number 1 less than that element is present in the hashmap or not. If the element is not present, then we do nothing. And if that element is present then that will be the starting element of sequence.
- After iterating on all the elements of array, Now, the values of keys are updated in Hashmap.

- For every TRUE value, we find the consecutive elements in the hashmap. Humare pass 2 variables hongay:
  - * msp = max starting point
    - For the starting point of sequence of maximum length
  - * mlen = max length
    - for the length of sequence of max. length.

= 5 → is have true as it's value in the hashmap.
∴ 5 – 6] → The consecutive sequence of 5 becomes (5–6).
Size of this sequence is 2.

∴ mlen = 2
msp = 5

= Now, the next sequence with '1' as the starting element is 1–2–3
length of this sequence = 3 → it is greater than mlen

∴ mlen = 3
msp = 1 → updated

= This will go untill we find the max. length out of all sequences.

∴ 8 – 9 – 10 – 11 – 12] → this is our max. sequence

∴ mlen = 5
msp = 8 → Ab Hum print krdengy!

```
public static void main(String[] args){
Scanner sc= new Scanner(System.in);
int n = sc.nextInt();  → size of array as input
int arr[] = new int[n];
for(int i=0; i<arr.length; i++){  → array fill kiya elements se
arr[i] = sc.nextInt();
}
HashMap<Integer,Boolean> map = new HashMap<>();  → ek naya hashmap banaya jisme keys aur uski values daali
for(int val: arr){  → sabhi keys ki values ko initially TRUE kra
map.put(val, true);
}
for(int val: arr){  → agar vo element se previous element
if(map.containsKey(val-1) == true){  present hai toh uss
map.put(val, false);  element/key ki value FALSE krdo
}
}  → maximum starting point
int msp=0;
int mlen=0;  → maximum length
for(int val: arr){  → Array ke sabhi elements pe loop lagega
if(map.get(val) == true){  aur vo element select hoga jo
int tsp=val;  TRUE hai hashmap me!
int tlen=1;
```

→ temporary starting point
→ temporary length

```
while (map.containsKey(tsp + tlen) == true) {
    tlen++;
}
if (tlen > mlen) {
    msp = tsp;
    mlen = tlen;
}
}
}
```

→ Hum har element ka Krengy aur temporary length ko increase Krty chalengy.

→ tlen is greater than mlen then update the values of msp and mlen.

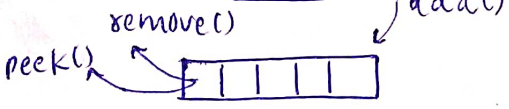
```
for (int i=0; i<mlen; i++) {
    System.out.println(msp + i);
}
}
```

→ Now, msp aur uske next elements ko print Krdo!

PRIORITY QUEUE / HEAP

- It is an Abstract DataType.
- It is very similar to Queue.
- In Priority Queue, har element ki kuch property hoti hai.
- The Priority of the elements in a Priority Queue determines the order in which elements are removed from the Priority Queue.
- ∴ All the elements are arranged in Ascending or Decesding Order
- PRIORITY QUEUE is an EXTENSION of the QUEUE with Following properties:
  - ✯ Every Queue has a PRIORITY associated with it.
  - ✯ An element with higher priority is dequeued before an element with lower priority
  - ✯ If two elements have the same priority, they are served according to their order of occurance in the queue.
  - ✯ The elements From Priority Queue are retreived in sorted Order.
  - ✯ The Priority Queue class has the same structure and function as that of Queue.

QUEUE



PRIORITY QUEUE

peek() and remove() are based on priority.

- ✯ Time Complexity of functions of Priority Queue.
  - add() ⟶ O(log n)
  - remove() ⟶ O(log n)
  - peek() ⟶ O(1)
- ✯ Similarities B/w PQ and Queue ← Structure Same; Same Functions
- ✯ Differences B/w PQ and Queue
  - We can set priority according our needs!
  - Highest Rand Order Priority: (smaller no. ko jyada Priority) → Ascending Order
  - Highest score Order Priority: (greater no. ko jyada Priority) → Decesending order

CREATING A PRIORITY QUEUE

★ Priority Set To Smaller Values

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
```

- `Integer` → Datatype of Element
- `pq` → name of Priority Queue

★ By Default, the priority will be set of smaller values (Highest Rank Order)
Ascending Order (SORTED)

★ Priority Set To Greater Values

```
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
```

★ Greater value elements will be given more priority.

reverse order me elements remove hongy!

HEAP SORT

Hume ek array given hoga. This array will be unsorted.

★ Approach for getting all elements in Ascending Order

10	57	33	44	56	11	98

- Make a priority Queue, priority will be set as default.
- Loop laga kar array ke sabhi elements ko PQ me add kardo!
- Now, PQ se ab hum elements ko retrieve krengy toh elements will be in sorted order (ASENDING ORDER)

10	11	33	44	56	57	98

```
import java.util.*;
public class FirstPriorityQueue
{
    public static void main(String [] args){
        int [] arr = {40, 90, 20, 14, 33}
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for(int val : arr) {          → One For loop has O(n)
            pq.add(val);              → add() function of PQ has O(logn)
        }                             O(nlogn)
        while( pq.size() > 0 )        → while loop has O(n)
        {
            System.out.println( pq.remove() + " ");   → remove() function of PQ has O(logn)
        }                                              O(nlogn)
}
```

∴ OUTPUT

14	20	30	40	90

∴ Time complexity : $O(n\log n) + O(n\log n)$

$= 2O(n\log n)$

$= O(n\log n)$

Heap Sort are of 2 Types:

- The one with space complexity → $O(n)$
- Another one where constant space k is used.

✱ Approach for getting alle elements in Descending Order

10	57	33	44	56	11	98

Make a (pq), priority will be given more to greater elements

```
import java.util.*;
public class FirstPriorityQueue{
public static void main (String[]args){
  int[] arr = {40, 90, 20, 14, 33};
  PriorityQueue<Integer> pq = new PriorityQueue<>
  (Collections.reverseOrder());
```

↓ Highest score priority Order

O(nlogn)
```
  for (int val : arr){
    pq.add(val);
  }
```

O(nlogn)
```
  while (pq.size > 0){
  System.out.println(pq.remove() + " ");
  }
}
```

OUTPUT :

90	55	40	33	30	11

K - Largest Element

Hume ek array given hoga aur ek (K) given hogi! Hum array me se Kth no. wala largest element print krdengy!

INPUT

10	19	3	74	86	57	24	5	11

∴ K = 3 OUTPUT → ∴ 74, 86, 57

APPROACH! ← Time complexity = O(n log n)
Space complexity = O(n)

- Phele ek (PQ) Banayengy where priority will be given more to greater elements. Uss (PQ) me array ke saare elements daal dengy! ← Time complexity = O(n log n), Space complexity = O(n)
- Fir K-times uss (PQ) me se elements ko remove karayengy toh humare paas K-largest element retreive hongy!
  ← Time complexity = O(K(log n)) → no. of elements removed.
  Space complexity = O(n) → remove() func

∴ Overall Time complexity ⇒ O(n log n) + O(K log n) ⇒ O((n+K) log n) ⇒ O(n log n)

Overall Space complexity ⇒ O(n)

```
public static void main (String [] args) {
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt();
    int arr[] = new int[n];
    for(int i=0; i<n; i++) {
        arr[i] = sc.nextInt();
    }
    int K = sc.nextInt();
PriorityQueue <Integer> pq = new PriarityQueue <>(Collections.reverseOrder());
    for( int val: arr) {
        pq.add(val);
    }
    for( int i=0; i<K; i++) {
        int val = pq.peek();
        pq.remove();
        System.out.println(val);
    }
}
```

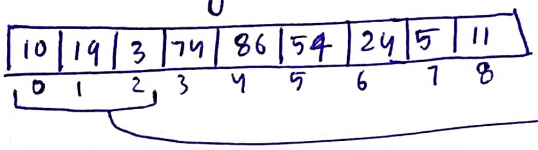
APPROACH 2

Time Complexity ⟶ $O(n \log n)$

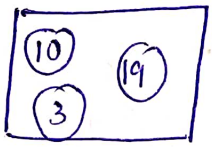
Space Complexity ⟶ $O(K)$

Highest Rank ⟵ Priority Order

- Hum (PQ) banayengy where the priority is set as default.
- (PQ) me array ke K^{th} elements hi add krey jayengy.

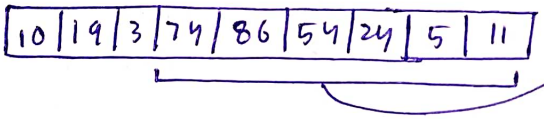


let's say K = 3



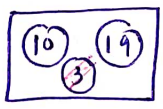
Peek() pr 3 hai

- Aab array pas ek loop lagega K^{th} index wale element se array ke last element tak aur hum saare elements ko peek se compare krengy!

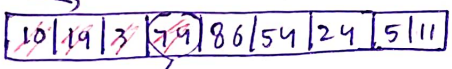


COMPARE WITH THE PEEK() = 3

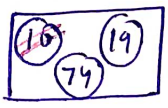
- If the element in the array is less than peek(), then do nothing
- But if the element in the array is greater than peek of (PQ) then,
  - ⟶ We, will remove that element from (PQ).
  - ⟶ Add the element from the array. in the (PQ)



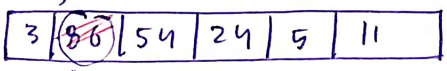
peek() = 3 ∴ 74 > 3

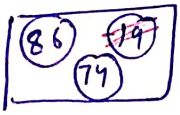


∴ Remove element from array and add to PQ

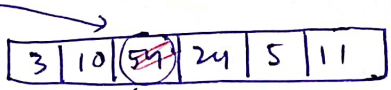


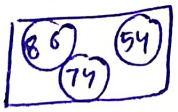
peek() = 10 ∴ 10 < 86





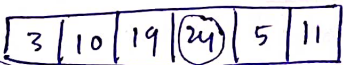
peek() = 19 ∴ 19 < 54



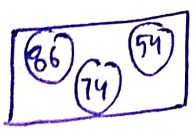


peek() = 54 ∴ 54 > 24

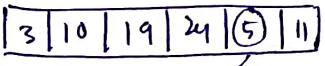
peek > element



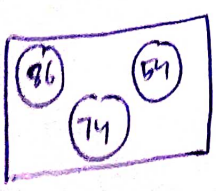
Do nothing and move to next element



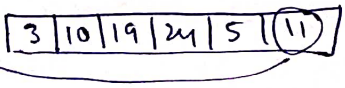
peek() = 54 54 > 5



Do nothing and move to next element



peek() = 54 54 > 11



Do nothing and move to next element

Now we will print and remove K elements from the (pq). We are left with 3 largest elements in (pq).

```
public static void main (String [] args) {
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt();
    int arr [] = new int [n];
    for (int i=0; i<n; i++) {
        arr [i] = sc.nextInt();
    }
    int K = sc.nextInt();
PeriorityQueue <Integer> pq = new PeriorityQueue <>();
    for (int i=0; i<K; i++) {
        pq.add (arr [i]);
    }
    for (int i=K; i< arr.length; i++) {
        int val = arr [i];
        if (val > pq.peek()) {
            pq.remove();
            pq.add(val);
        }
    }
    while (pq.size()>0) {
        System.out.println (pq.peek());
        pq.remove();
    }
}
```

MERGE K-SORTED LISTS

[LIST = Arraylist]

Humne K-sorted list given hogi, humne unn K-sorted lists ko merge krke 1 sorted list banani hai.

TC → O(nlogn) SC → O(K)

∴ K = 4

L1 = 10, 20, 30, 40
L2 = 5, 7, 9, 11, 19, 55, 57
L3 = 1, 2, 3
L4 = 32, 39

Merged List = 1, 2, 3, 5, 7, 9, 10, 11, 19, 20, 30, 32, 39, 40, 50, 55, 57

Approach 1

Phele saari list ke sabhi elements (PQ) me daaldo aur unhe (PQ) se remove krke (AL) me daaldo.

Par yaha space complexity bahut high hojayegi!

Approach 2

Humne K-sorted list diiven hai!

Hum (PQ) Banayegi aur usme har list ka phela element daaldengy Aur is (PQ) me humne priority di hai smaller elements ko!

Toh humare pass (PQ) me K elements hai because of K no. of lists.

Now, we will remove the element with highest priority and add 1 element in the (PQ) (from the same Arraylist to which the remove element belong).

This will be done in loop unless the (PQ) becomes empty.

In this way we will get our sorted Arraylist and we will print the resultant Arraylist.

NOTE

Hume har element ke sath yeh track rakhna hai ki vo konsi arraylist ka data hai aur kya vo Arraylist ka last index hai? (For resolving null pointer exception)

Inn sab baato ka track rakhna jaruri hai!

Issliye Hum ek Pair class Banayengy

Pair class Consist of →
- data → data is the value at (di) index of (li).
- dataindex (di) → to store the index of data in the (li) list of arraylist.
- list index (li) → to store the index of arraylist.

APPROACH

① Hum ek Pair Class Banayengy!
- list index (li)
- data index (di)
- data

li → 0 1 2 3
di → 0 1 2 3 4 5 6

li							
0	10	20	30	40	50	60	
1	5	7	9	11	19	55	57
2	1	2	3				
3	32	39					

→ data index represent the index of data in the list index of arraylist

→ K list are provided in Arraylist. So listindex will store the index of li (th) list of arraylist. And ∴ it helps us to keep track of the position in the arraylist.

② Hum ek (PQ) banayengy of Pair class (type), aur (PQ) me Pair class ke objects dalengy, But (PQ) me hume essa data dalana hota hai jiski periority (PQ) ko pata chal sakey!
∴ By default the smallest element is given more periority in (PQ).

Now, (PQ) ko kaise pata chalega ki Smallest Pair konsa hai?
Toh hume compare krna hai Pair class ke objects ko,
Toh yeh krne ke liye hume pair class me comparable implement krna hoga,
Taki Pair class ka har object ka data Dusre object ke data se hume compare kr paye!

```
static class Pair implements Comparable <Pair>
{
    int li;
    int di;
    int data;

    public int compareTo (Pair o)
    {
        return this.data - o.data;
    }
}
```

Agar hum kisi class me comparable implement krty hai toh waha compareTo function ana hi chaiye!

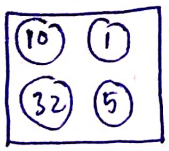
o means other objects of class Pair

- Agar yeh difference (+ve) mila toh this is Bigger!
- Agar (-ve) mila toh this is smaller!
- Agar 0 toh this and other are equal.

Matlab smaller elements / Pairs are give more Periority

this.data − o.data	→ Highest Rank Periority
o.data − this.data	→ Highest Score Periority

(3) So, First of all we will create a (PQ).
ussme har list ke First element ke liye Pair Banake (PQ) me dalogy!



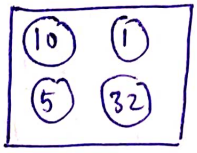
PQ

Peek() = 1

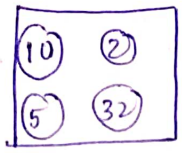
Toh humne har element ke liye Pair banaya where the list index will be the index of list to which the element belong.

(4) Now, we will create a Resultant Arraylist, Aab hum (PQ) me ek element remove krengy aur ek new element (PQ) me dalengy (From the same list ~~to~~ to which element is removed)

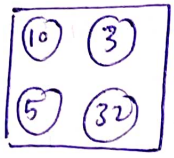
DRY RUN



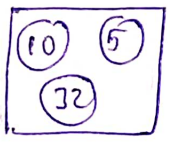
PQ
Resultant Arraylist (RA) = _
Peek() = 1



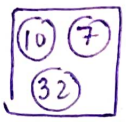
RA = 1
Peek() = 2



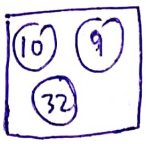
RA = 1 2
Peek() = 3



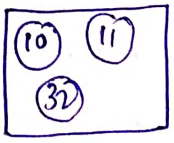
RA = 1 2 3
Peek() = 5



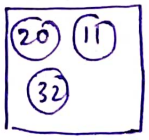
RA = 1 2 3 5
Peek = 7



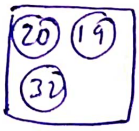
RA = 1 2 3 5 7
Peek = 9



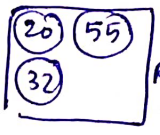
RA = 1 2 3 5 7 9
Peek() = 10



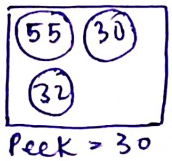
RA = 1 2 3 5 7 9 10
Peek = 11



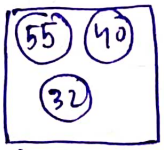
RA = 1 2 3 5 7 9 10 11
Peek = 19



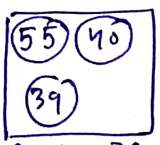
RA = 1 2 3 5 7 9 10 11 19
Peek = 20



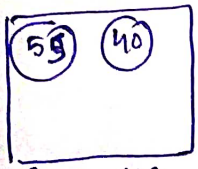
RA = 1 2 3 5 7 9 10 11 19 20
Peek = 30



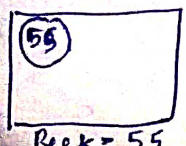
RA = 1 2 3 5 7 9 10 11 19 20 30
Peek = 32



RA = 1 2 3 5 7 9 10 11 19 20 30 32
Peek = 39



RA = 1 2 3 5 7 9 10 11 19 20 30 32 39
Peek = 40



Peek = 55
RA = 1 2 3 5 7 9 10 11 19 20 30 32 39 40

Peek = 57
RA = 1 2 3 5 7 9 10 11 19 20 30 32 39 40 55

PQ

∴ RA = 1 2 3 5 7 9 10 11 19 20 30 32 39 40 55 57

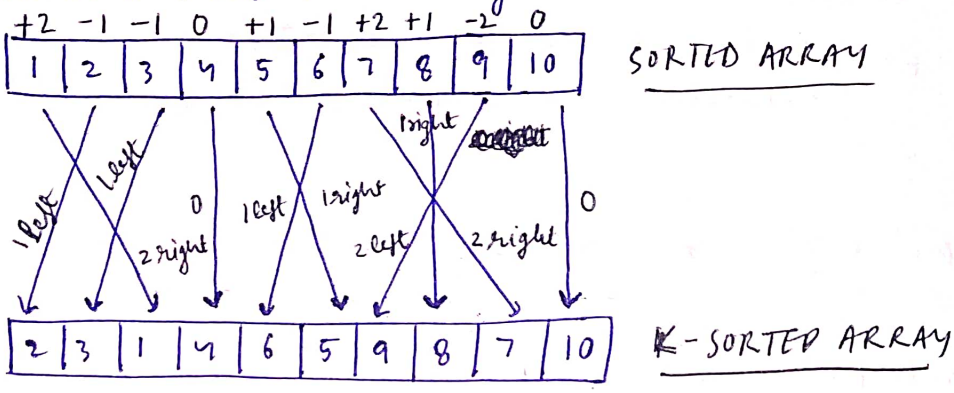
```
public class Pair implements Comparable {
    int data;
    int di;
    int li;
    public int compareTo(Pair o) {
        return this.data - o.data;
    }
}

public static ArrayList<Integer> mergeKSortedLists(ArrayList
                                <ArrayList<Integer>>
                                lists) {

ArrayList<Integer> resultAL = new ArrayList<>();
PriorityQueue<Pair> pq = new PriorityQueue<>();
for(int li = 0; li < lists.size(); li++) {
    Pair p = new Pair();
    p.li = li;
    p.di = 0;
    p.data = list.get(li).get(0);
    pq.add(p);
}
while (pq.size()>0) {
  Pair removedPair = pq.peek();
  pq.remove();
  resultAL.add(removedPair.data);
  removedPair.di++;
  if(removedPair.di < list.get(removedpair.li).size()) {
   removedPair.data = list.get(removedPair.li).get(removedPair.di);
   pq.add(removedPair);
  }
}
return resultAL;
}
```

SORTED K-SORTED ARRAY

Hume ek array given hoga, aur ek integer K given hoga!
The integer K denotes that the array is a K-sorted array.



∴ Hume ek K-sorted array given hoga! K-sorted array me har element apni position se (0 to K) K-position left ya right par move ho sakta hai!

if K = 2 ⟶ Har element ko 1 ya 2 position left ya right move kr sakty hai!

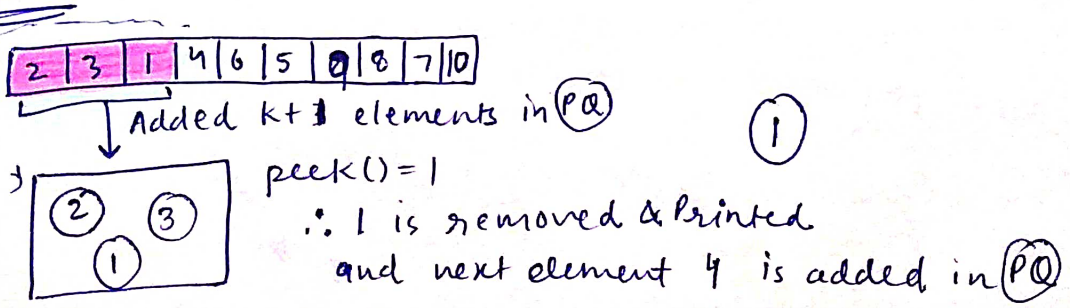
It is also possible that element does not move!

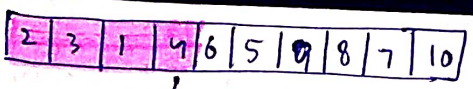
∴ We can only move 0/1/2 positions if K=2

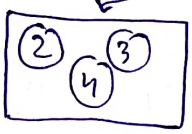
APPROACH

- Hum ek (PQ) Banayengy of size (K+1) — Highest Rank → Priority Order → smaller ↑
- Now, Hum ek element remove krengy (PQ) se uss removed element ko print kradiya!
  Aab Humare pass (K) elements hai (PQ) me!
- So, we insert next element in the (PQ) and size again becomes (K+1), we keep on doing this unless (PQ) becomes empty, we keep on removing 1 of the element from (PQ) and adding the next element of the array (unless we have no next element in the array to be added in the (PQ)).
- If the array is completely visited & some elements are left in the (PQ), we empty the (PQ) completely and Print the removed element, and then this procedure stops here.

DRY RUN

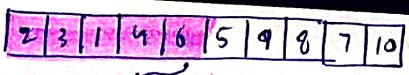


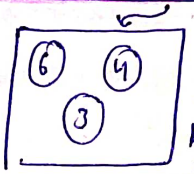




peek() = 2

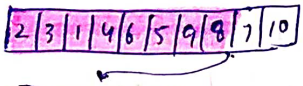
2 is removed and print, next element 6 is added

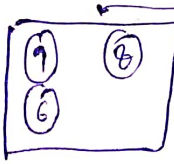




peek() = 3

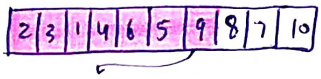
3 is removed and printed, next element 5 is added

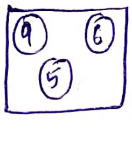




peek() = 6

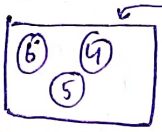
6 is removed, printed, 7 is added





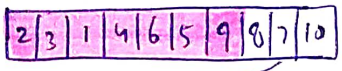
peek() = 5

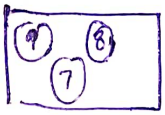
5 is removed, printed, 8 is added



peek() = 4

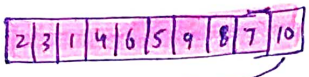
4 is removed & printed, next element 9 is added

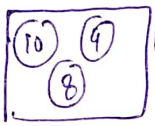




peek() = 7

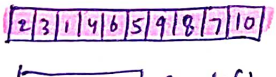
7 is removed and printed, 10 is added

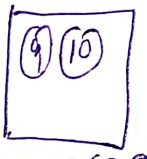




peek() = 8

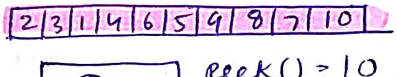
8 is removed, printed,

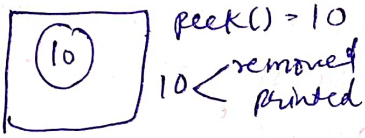




peek() = 9

9 removed printed

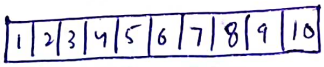




peek() = 10

10 removed printed

∴ OUTPUT



```
public static void main (String [] args){
Scanner sc = new Scanner (System.in);
int n = sc.nextInt();
int arr[] = new int [n];
for(int i=0; i<n; i++){
    arr[i] = sc.nextInt();
}
```

Humne K-sorted array input liye

```
int K = sc.nextInt();
```

→ K ki value input li

```
PriorityQueue <Integer> pq = new PriorityQueue <>();
```

→ Ek Periority Queue Banai

```
for (int i=0; i<=K; i++){
    pq.add(arr[i]);
}
```

→ array ke K+1 elements pq me dadle!

```
for(int i=k+1; i<arr.length; i++){
    int val = pq.peek();
    System.out.println(val);
    pq.remove();
    pq.add(arr[i]);
}
```

→ Jab tak array ke bache huay elements (PQ) me na daal jaye tab tak loop chalega

→ peek kro

→ print kro

→ remove kro

→ next element add kro!

```
while(pq.size()>0){
    int val = pq.peek();
    System.out.println(val);
    pq.remove();
}
```

→ Agar (PQ) me ab bhi kuch elements bache hai toh print krdo aur unhe (PQ) me se remove krdo!

Time Complexity

$O(n) + O(\log K)$

↓ This is because we are traversing the entire array.

↓ removing the elements from (PQ) since the size of (PQ) is O(K)

$O(n \log K)$

Space Complexity = $O(K)$

MEDIAN PRIORITY QUEUE

Humne apni ek (PQ) banani hai yeh normal (PQ) nahi hai yeh hai MEDIAN PRIORITY QUEUE.

Iss (PQ) me MEDIAN ko sabse jyda priority milegi!
Har baar remaining elements ka median hi return krengy!

10	20	30	40	50

Normal (PQ) (smaller priority)
peek() → 10
remove() → 10

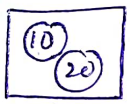
Normal (PQ) (larger priority)
peek() → 50
Remove() → 50

Median (PQ) (median priority)
peek() → 30
remove() → 30
→ 30 is median of all elements in (PQ).

APPROACH

① Median Priority Queue

→ Max Priority Queue ← left (PQ) stores the first (smaller) half of queue



peek = 20 max

→ Min Priority Queue ← right (PQ) stores the greater half of the element.

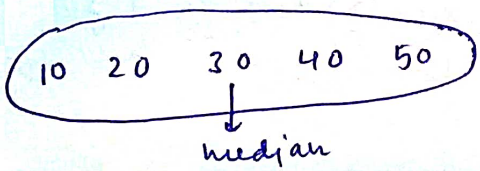
peek = 30 min

∴

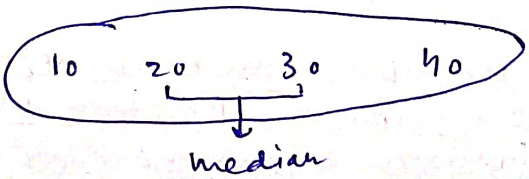
	left (PQ)	right (PQ)	
peek	n	n	left
	n+1	n	left
	n	n+1	right

✓ If the median queue has even no. of element i.e. If both left (PQ) & right (PQ) has equal size. left (PQ) me se element remove hoga and that element will be printed as a median.

✓ If there are odd number of elements in the (PQ) if the left priority /right priority queue has n+1 elements & the other priority queue right/left (PQ) has n elements, then in this case, the (PQ) with n+1 element will remove an element.

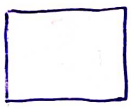


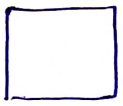
Odd no. of elements in a median Priority Queue



Even no. of elements in a median priority Queue

DRY RUN

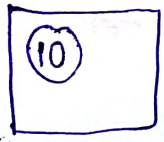

peekQ() = –


peekQ() = –

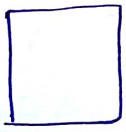
∴ Our (MPQ) is empty (initially)

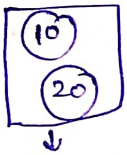
∴ Add(10)

Add(20)



peekQ() = 10


peekQ() = –



peekQ() = 20
size = 2


peekQ() = –
size = 0

By default, element is inserted in left (PQ).

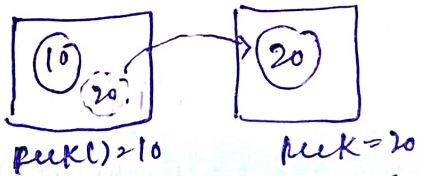
By Default, an element is inserted in the left (PQ).

Rule 1

Rule 2

But, if the difference of size of left and right (PQ) should be (1).

∴ But here difference becomes (2). So, remove 1 element from (PQ) having more element & put it in another.



Now size of left and right PQ is equal.

RULE 3

When the element being added is greater than peekQ() of right (PQ), then the element will get inserted in right (PQ).

element > right.peekQ() → inserted in right (PQ)

When the element being added is smaller than the peekQ() of left (PQ) then the element will be inserted in left (PQ).

element < left.peekQ() → Inserted in left (PQ).

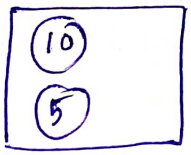
After inserting we need to check wheather the left and right (PQ) is balanced.

If size difference is greater than 1 then use RULE 2

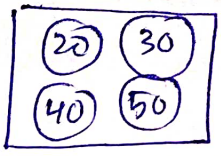
If the element < right.peekQ() & element > left.peekQ() toh by default, uss element ko left (PQ) me add krengy! Then we will check if the (PQ) has size difference < –1.

Add(30) → 30 > right.peek() → inserted in right (PQ)
Add (5) → 5 < left.peek → inserted in left (PQ)
Add (40) → 40 > right.peek → inserted in right (PQ)

∴ Add(50) → 50 > right.peek → inserted in right (PQ).

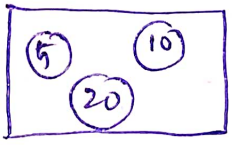


peek() = 10
(Max. PQ)

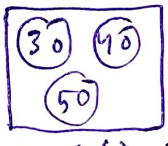


peek() = 20
(Mini. PQ)

size diff. = 2 ← There isn't balance in size ← USE RULE 2 ← Remove one element from (RPQ) and add to (LPQ).



peek() = 20



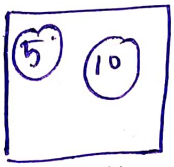
peek() = 30

Now, MedianPQ.peek() → 20 → WHY ?

The median is the peek of (PQ) with more elements in total But if, in total there are even no. of elements, then median is the peek of left PQ

MedianPQ.remove() ← 20

equal no. of element in left (PQ) and right (PQ) and even no. of elements in total, we will remove the element from left PQ.



peek() = 10
size = 2

peek() = 30
size = 3

5 elements in Total
∴ when the size is Odd (when size of left PQ and right PQ is unequal) then remove the element from (PQ) having more elements.

rightPQ size > leftPQ size → Remove element from right PQ

∴ Order of Removal of element from median (PQ's).
→ 20 → 30 → 10 → 40 → 5 → 50

In this Question, we have to write following Functions

Time Complexity
- Add() → O(log n)
- Peek() → O(log n)
- Remove() → O(1)
- Size() → O(1)

Similar to normal PQ

Space complexity → O(1)

```
public static class MedianPriorityQueue
{
  PriorityQueue<Integer> left;  → Yeh left PQ banaya (Max Priority)
  PriorityQueue<Integer> right; → Yeh right PQ banaya (Min Priority)
  public MedianPriorityQueue() → constructor of median PQ class
  {
    left = new PriorityQueue<>(Collections.reverseOrder()); → left PQ (max)
    right = new PriorityQueue<>(); → right PQ (min)
  }
  add func of Median PQ
  public void add(int val)
  {
    if(right.size() > 0 && val > right.peek())
    {
      right.add(val);
    }
    else {
      left.add(val);
    }
    handleImap(); → Agar left aur right PQ ka size diff. 1 se jyada hua toh hum balance karna padega!
  }
  private void handleImap()
  {
    if( left.size() - right.size() == 2)
    {
      int val = left.remove();
      right.add(val)
    }
    else if( right.size() - left.size() == 2)
    {
      int val = right.remove();
      left.add(val);
    }
  }
  public int remove()
  {
    if( this.size() == 0)
    {
      System.out.println("Underflow");
      return -1;
    }
    if( left.size() >= right.size())
    {
      return left.remove();
    }
    else
    {
      return right.remove();
    }
  }
```

```
public int peek()
{
    if (this.size() == 0)
    {
        System.out.println("Underflow");
        return -1;
    }
    if(left.size() >= right.size())
    {
        return left.peek();
    }
    else
    {
        return right.peek();
    }
}
public int size()
{
    return left.size() + right.size();
}
}
public static void main (String [] args)
{
    BufferReader br = new BufferReader (new InputStreamReader
                                        (System.in));
```

↑ Humne medianPQ class ka ek object banaya

```
MedianPriorityQueue queue = new MedianPriorityQueue();
String str = br.readline();
while ( str.equals ("quit") == false)
{
    if( str.startsWith("add"))
    { int val = Integer.parseInt (str.split(" ")[1]);
      queue.add(val);
    }
    else if ( str.startsWith("remove"))
    { int val = queue.remove();
      if ( val != -1) {
          System.out.println(val);
      }
      else if (str.startsWith ("peek"))
      { int val= queue.remove();
        if (val != -1)
        { System.out.println(val);
        }
      }
}
```

object pe ~~add~~ add func call kiya

```
else if (str.startsWith("size"))
{
    System.out.println(queue.size());
}
str = br.readLine();
}
}
```

PRIORITY QUEUE USING HEAP

(Priority Queue ek class hai, jo Heap Data Structure use krke Banayi gayi hai)

Hume apni kudki (PQ) class banani hai which will have following functions:
- add()
- peek()
- remove()
- size()

(MAIN)

Heap is a Data Structure which is based on BINARY TREE

Priority Queue Banane ke liye Humare pass 3 Tarike hai!

WAY 1 Priority Queue using a Sorted LinkedList

Works but the Time complexity is not good.

add (10) → 10
add (20) → 10 → 20
add (5) → 5 → 10 → 20
add (40) → 5 → 10 → 20 → 40

∴ n^2 time chalega loop array ke sabhi elements ke liye! (add ke liye)

add function: me hume puri linked list ko travel krna pad raha hai tab jake correct position pe add hoga → O(n)

peek() → O(1) → gives us the smallest element by default. LinkedList is sorted ∴ smallest element is head of LinkedList

remove() → O(1) → returns and remove the smallest element of LinkedList.

USING SORTED LINKEDLIST

	add()	peek()	remove()
For 1 element	O(n)	O(1)	O(1)
for array	$n(n) = n^2$	$n(1) = n$	$n(1) = n$

∴ NOTE

Hume,
add() → O(logn)
remove() → O(logn)
peek() → O(1)

(Chaiye hume yahi)

WAY 2 USING UNSORTED LINKEDLIST

Elements ko randomly tail pe add krty chalo, toh humare paas unsorted LinkedList bani jayegi!

add(10) → 10
add(20) → 10 → 20
add(5) → 10 → 20 → 5
add(22) → 10 → 20 → 5 → 22

add(): Hume bs TAIL pr next element ko add krna hoga and that element will be the next Tail. ∴ O(1)

remove()
In (PQ), we remove the smallest element, because the peiority is given to the smallest by default,
Lekin humare LinkedList unsorted hai, toh smallest remove krne ke liye hume puri LinkedList ko travel krna hoga! → O(n)

peek()
Similar to remove()-function ∴
O(n)

USING UNSORTED LINKEDLIST

add → O(1)
remove → O(n)
peek → O(n)

n time me add → n(1) → n
n time me remove → n(n) → n²
[for array ke sabhi elements ke liye]

Here also time complexity are not the desired one.

WAY 3 USING ARRAYLIST LIKE HEAP DATA STRUCTURE

Isse use krne se,

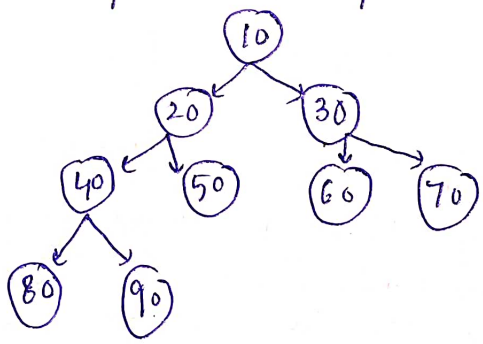
PQ me
- add() → O(log n)
- remove() → O(log n)
- peek() → O(1)

[For single Element]

- pure array ko PQ me add krne ke liye T.C
  = O (n element x log n) ← T.C of 1 element
- pure array ke sabhi element ko PQ me se remove krne ke liye T.C → O(n log n)
- pure array ke sabhi elements ke liye peek() ki TC
  = n X 1 = O(n)

WHAT IS HEAP ? Hum Heap ka USE Kr rahey hai !

- Heap is a Binary Tree having 2 Special Properties !
  * Complete Binary Tree (CBT)
  * Heap Order Property (HOP)



- Hum Arraylist ko Heap ki trah use kr rahey hai !

Taki
- add → log (n)
- remove → log (n)
- peek → O(1)

] → hojaye !

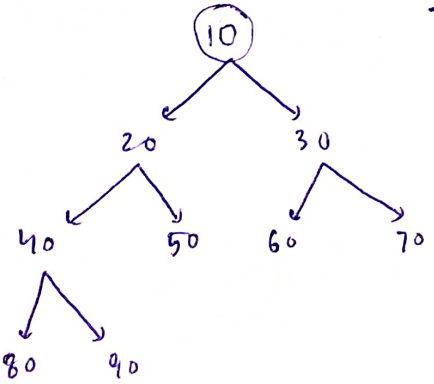
⭐ Heap Order Property

parent ki periority Dono children se kadi hoti hai ! (hamesha)
[parent ki value hamesha children ki value se kadi hogi] ← matlab

left aur right ki periority me koi relation nahi hai koi bhi bada hosakta hai !
And to be specific the periority of any child is not pre-decided. This property helps to acheive the T.C of peek() = O(1)

Humne Heap Order Property use kr kri?

peek() ⟶ T.C = O(1) → kuki Hume bas ROOT return krna hoga kuki Parent uske dono childrens se jyda periority hoga!



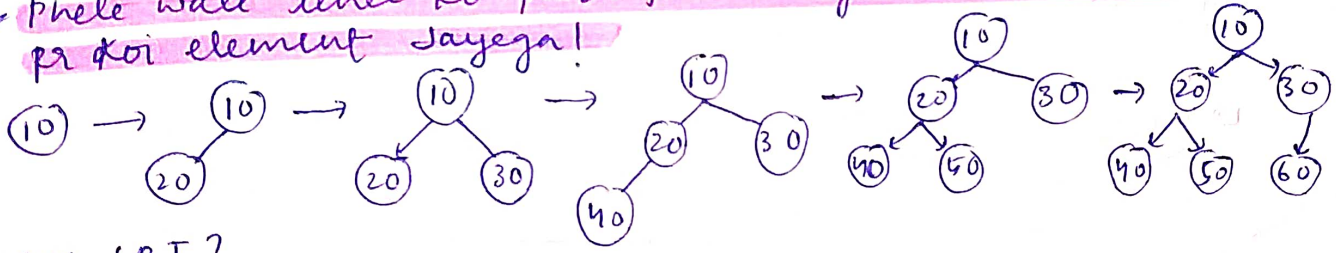
This tree is a Binary Tree (every node has 2 child)
This tree follows Heap Order Property kuki har parent node ki periority uske childrens se jyda hai

HOW?
- Toh sabse Jyada Periority hogi ROOT node ki!
- So peek() func will simply return root.
- ∴ T.C of peek() ⇒ O(1) due to HOP.

(A) COMPLETE BINARY TREE

Let suppose the height of the tree is (h), then according to this property atleast (h-1) levels should be completely filled, and the last (h^{th}) level should be filled from left to right.

- Har Parent ka right child attach krne se phele left child attach krna hoga!
- Phele wale level ko pura fill krnay ke bad hi next level pr koi element jayega!



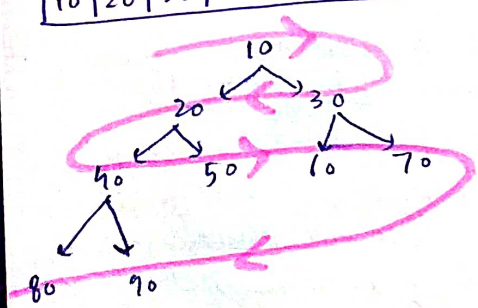
WHY CBT?

add() / remove() ⟶ T.C = O(log n) → KESE?

→ CBT ke wajah se hum ek tree ko ek array / arraylist se represent kr sakty hai

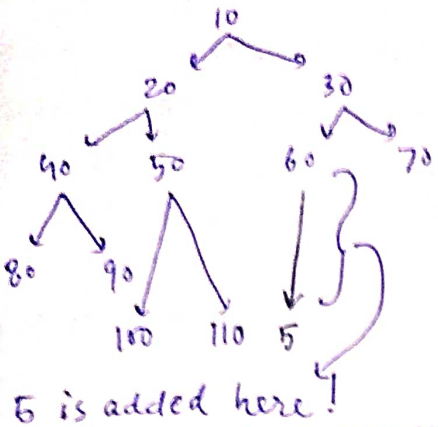
0	1	2	3	4	5	6	7	8
10	20	30	40	50	60	70	80	90

Iss arraylist ko tree ki form me bana dia!



Agar arraylist me element badety jayengy toh tree me left to right vo element lagaty jayengy aur jab yeh level full ho jayega toh next level me left to right yeh element lagaty jayengy!

So, this CBT is our Priority Queue. This CBT is actually a arraylist behind the scene.
This arraylist also follows the HOP!



peek() : peek() gives the root ∴ O(1)
Root → list.get(0); → (10)

add() : add() func will add the data in the end of arraylist.

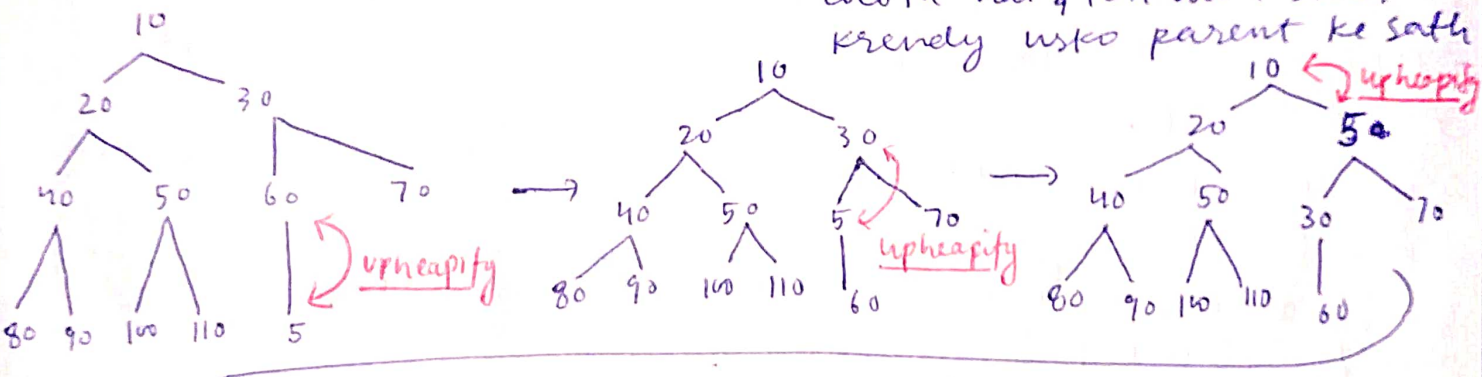
10	20	30	40	50	60	70	80	90	100	110	5

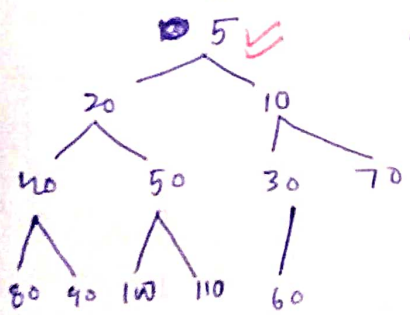
5 is added here

5 is added here!
Ab Hum **UPHEAPIFY()** func ka use krengy aur 5 ko uske parent se swap krdengy yeh tab tak krengy jab tak 5 chota rahega apne parents.

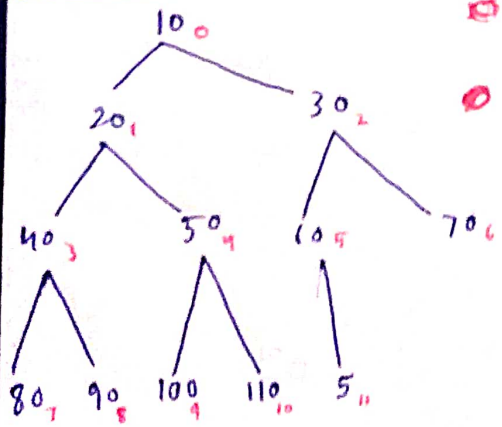
UPHEAPIFY()

node agar uske parent se chota hai, toh hum swap krenely usko parent ke sath





∴ we have upheapified (3) times (loop 3 baar chalega!
height = no. of edges = 3
Height ki T.C = O(log n) hoti hai
inta time lagega upheapify ke loop me!

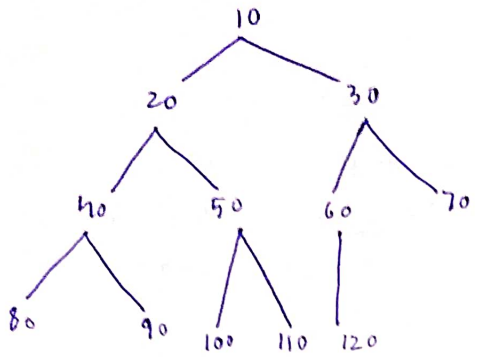


2° Arraylist se tree banane ke Benefits?
Hum har node se uska Parent index find kr sakty hai!

= Hume konse spot pe 5 add krna hai yeh arraylist se pata chalega ki (11) index pe krna hai!

= Lekin, Agar hum Tree use krty CBT banane ke liye, toh hum pura Euler Tree Traverse krengy aur pata chalega ki (60) ko child hai (5).

remove() → Hume highest priority ko remove krna hai!



0	1	2	3	4	5	6	7	8	9	10	11
10	20	30	40	50	60	70	80	90	100	110	120

Arraylist me smallest element ko remove krna hai!
0th index wala element remove krne ke liye arraylist ki T.C → O(n) hoti hai kuki sabhi elements ko 1 index aage shift krna padega!

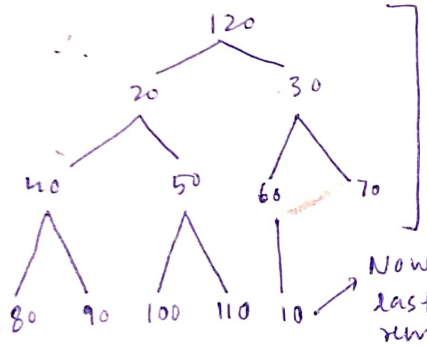
Toh sabse phle Arraylist ke 1st aur last element ko swap krengy

10	20	30	40	50	60	70	80	90	100	110	120

SWAP

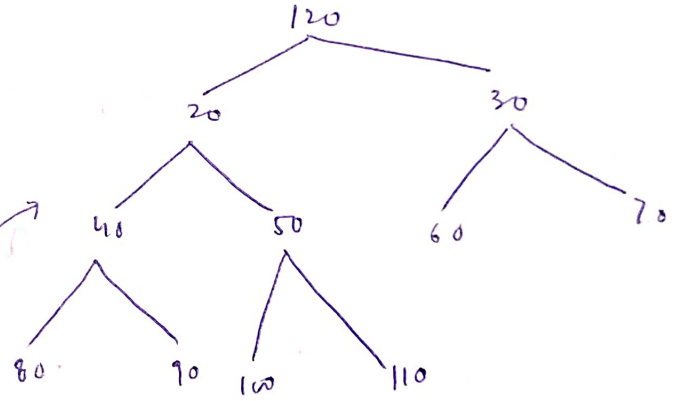
→

120	20	30	40	50	60	70	80	90	100	110	10

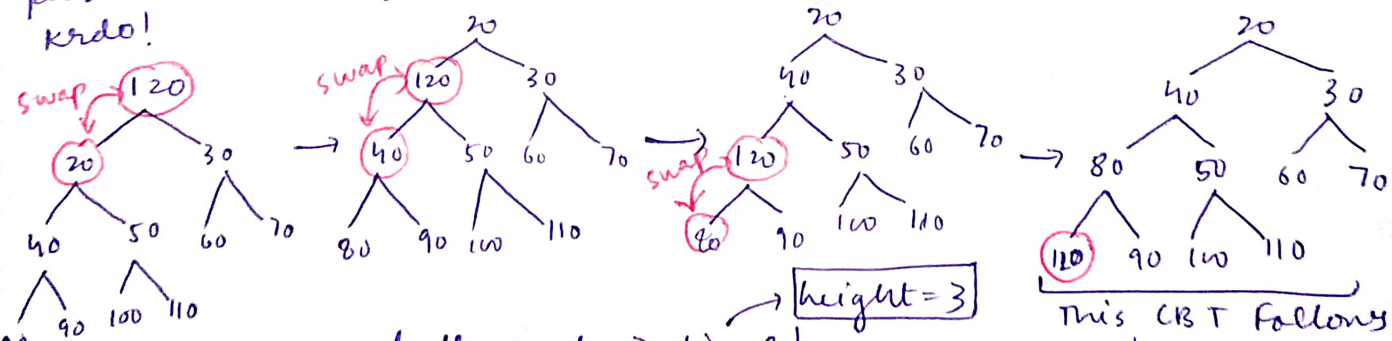


This CBT doesn't follow the HOP

Now, Hum last element remove krengy! Arraylist se!



Ab Hum DOWNHEAPIFY() function use krengy! Root ko compare krengy left, right child ke sath aur agar kisi node (child) ki value parent node se kam hai toh uss child se parent ko swap krdo!



height = 3

This CBT follows HOP

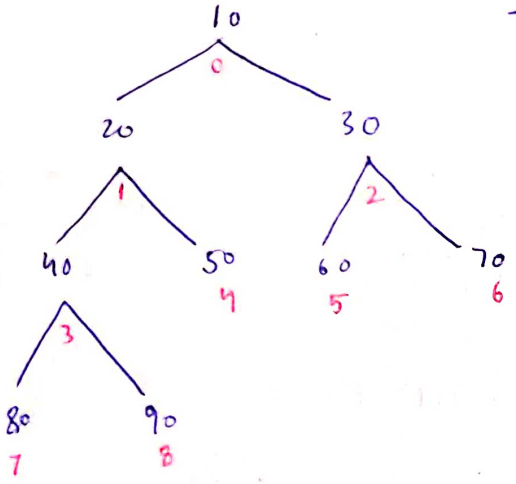
∴ we have swapped the node 3 times!

PRIORITY QUEUE USING ARRAYLIST AS A HEAP

- add → O(log n) → First add the element in arraylist, then upheapify
- remove → O(log n) → First swap first, last element of arraylist, remove the last, and Now downheapify.
- peek → O(1) → Just return the First element of Arraylist

IMPLEMENTATION

Data Structure used for storage by Heap is Arraylist.



FORMULAS

left child index = 2 [Parent Index] + 1

right child index = 2 [Parent Index] + 2

Parent	left child	right child
0	1	2
	2×0 + 1 = 1	2×0 + 2 = 2
1	2	3
	2×1 + 1 = 2	2×1 + 1 = 3

∴ parent index = $\frac{\text{child index} - 1}{2}$

For left child → $\frac{2-1}{2} = \frac{1}{2} \simeq 1$

For right child → $\frac{3-1}{2} = \frac{2}{2} = 1$

CODE

```
public static class PriorityQueue {
    ArrayList<Integer> data;                    // Humne ek arraylist ko declare kra!
    public PriorityQueue() {                    // Constructor of Priority Queue class
        data = new ArrayList<>();               // data arraylist ko initialized kiya
    }                                           // arraylist works as a heap
    public void add(int val) {                  // add function
        data.add(val);                          // adding data at the last of arraylist
        upheapify(data.size() - 1);             // upheapify the data at the last of the arraylist
    }
    void upheapify(int i) {                     // upheapify function
        if (i == 0) {
            return;
        }
        int pi = (i - 1) / 2;                   // parent index find kiya
        if (data.get(pi) > data.get(i)) {       // Agar parent index ki value
            swap(pi, i);                        // jyada hai toh swap krdo
        }                                       // aur fir recursive call
        upheapify(pi);                          // lagado!
    }
```

```
public int remove() {  ⟶ remove function
    if (data.size() == 0) {
        System.out.println("Underflow");
        return -1;
    }
    swap(0, data.size() - 1);   ⟶ Pehle AL ke 1st aur last element swap karo!
    int val = data.remove(data.size() - 1);   ⟶ fir last element ko remove karo!
    downheapify(0);   ⟶ fir 1st element pe downheapify call karo!
    return val;
}
```

parent index

```
void downheapify(int i) {  ⟶ downheapify function
    int mini = i;   ⟶ mini ko parent index dia!
    int lci = 2*i + 1;
    int rci = 2*i + 2;
    if (lci < data.size() && data.get(lci) < data.get(mini))
    {   mini = lci;
    }
    if (rci < data.size() && data.get(rci) < data.get(mini))
    {   mini = rci;
    }
    if (mini != i) {
        swap(mini, i);
        downheapify(mini);
    }
}
```

Agar kisi bhi child ki value parent index se kam hai toh parent index ko uska child banado

Agar mini aur parent index same nahi hai toh matlab (pi) ki value change hui hai, aur hum recursive call lagani hai!

```
public void swap(int i, int j) {
    int ith = data.get(i);
    int jth = data.get(j);
    data.set(i, ith);
    data.set(j, jth);
}
public int peek() {
    if (data.size() == 0) {
        System.out.println("Underflow");
        return -1;
    }
    return data.get(0);
}
public int size() { return data.size(); }
```

```
public static void main (String [] args) {
BufferReader br = new BufferReader (new InputStreamReader
                                    (System.in));

PriorityQueue qu = new PriorityQueue ();
String str = br.readLine();
while ( str.equals ("quit" == false)) {
      if ( str.startsWith ("add")){
       int val = Integer.parseInt (str.split (" ") [1]);
       qu.add (val);
      }
      else if (str.startswith ("remove")) {
       int val = queue.remove ();
       if (val != -1) {
            System.out.println (val);
       }
      }
     else if ( str.startswith ("peek")){
         int val = queue.peek();
          if (val != -1){
             System.out.println(val);
          }
      }

else if ( str.startswith ("size")) {
      System.out.println (qu.size());
}
str = br.readline();
}
}
```

Efficient Heap CONSTRUCTOR

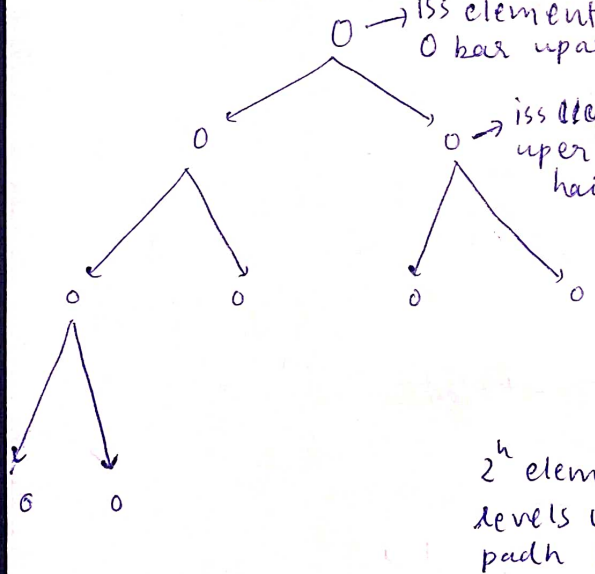
```
public MyPriorityQueue(int[] arr){
    data = new ArrayList<>();
    for(int val : arr){
            add(val);
    }
}
```

Jo Heap Constructor humare pass tha! Usme agar humari array me (n) elements given hai, toh (PQ) me T.C = O(logn) hai. Toh array ke sabhi elements par for loop lagake agar hum (PQ) me add krty hai toh,

for loop ki T.C = O(n)

1 element ki (PQ) me add ki T.C = O(logn)

∴ T.C of add n elements in PQ = O(nlogn)



element ← 2^0 . 0 → level

$2^1 \cdot 1$

$2^2 \cdot 2$

$\leftarrow 2^h \cdot h = 2^{\log n} \cdot \log n$

$= n \log n$

- $n = 2^0 + 2^1 + 2^2 + \ldots + 2^h$ → HP of Total terms = h+1

$a = 1 = 2^0$

$r = 2^1$

$n = h+1$

$\therefore HP_{sum} = \frac{a(r^h - 1)}{r-1} = \frac{2^0 (2^{h+1} - 1)}{2-1}$

$n = \frac{1(2^{h+1} - 1)}{1} = \frac{2^{h+1} - 1}{1} \Rightarrow n = 2^{h+1} - 1 \Rightarrow n+1 = 2^{h+1}$

Taking log both side

$h+1 = \log_2(n+1) \Rightarrow h = \log_2(n+1) + 1 \Rightarrow \boxed{h \simeq \log n}$ Hence Proved

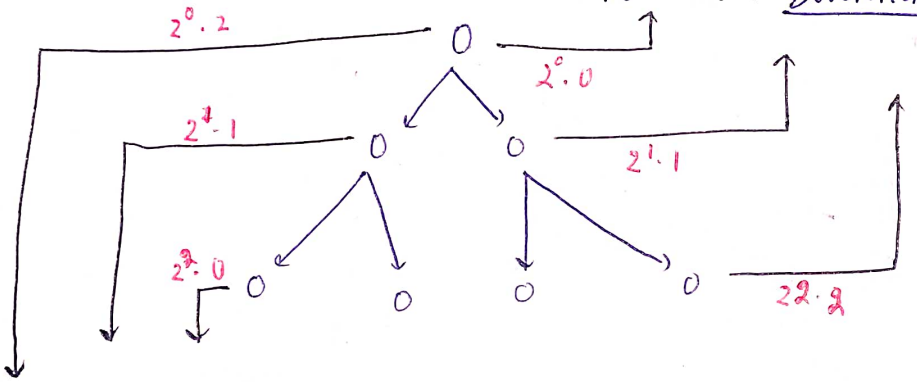
$T.C = \underbrace{2^0 \cdot 0 + 2^1 \cdot 1 + 2^2 \cdot 2 + \cdots + 2^h \cdot h}$ → T.C to add n element in PQ and then upheapify.

$= \underbrace{2^0 \cdot 0 + 2^1 \cdot 1 + 2^2 \cdot 2 + \cdots}_{\text{IGNORED}} \underbrace{2^h \cdot h}_{\star}$ → $2^h \cdot h = 2^{\log n} \cdot \log n$

$\therefore T.C = \underbrace{2^{\log n}}_{n} \cdot \underbrace{\log n}_{*\ \log n} = n \log n$

∴ Sabse jyada elements last level pr hongy (2^h) toh sabse jyda kaam bhi (upheapify) hume last level pe krna hoga!

BETTER APPROACH → Hum UPHEAPIFY se heap nahi Banayengy
↳ Ab Hum DOWNHEAPIFY se heap Banayengy!



$2^0 \cdot h + 2^1 \cdot h-1 + \cdots\cdots 2^h \cdot 0 = T(n)$ —(1)

multiplying both sides by 2

$2^1(h) + 2^2(h-1) + \cdots\cdots + 2^{h+1}(0) = 2T(n)$ —(2)

Substract (2) From (1)

$-2^h - 2^1(h-1) \cdots\cdots 2^1 = -T(n)$

$T(n) = 2^h + 2^{h-1} + 2^{h-2} + \cdots 2^1 - 2^0(h) = \frac{2(2^h - 1)}{1} = h$

$\therefore T(n) = 2n - 2 - \log n$

$\boxed{T(n) \propto n}$

Phle sabhi elements ko PQ me add krdo aur fir,

$\left(\frac{n}{2} - 1\right)$ index se (0) index tak downheapify function call krdo!

```
public MyPriorityQueue(int[] arr)
{
    data = new ArrayLists<>();

    for (int val : arr)
    {
        data.add(val);
    }
    for (int i = data.size()/2 - 1; i >= 0; i--)
    {
        downheapify (i);
    }
}
```

Now the T.C of add all elements of array in PQ will be O(n).