DAY 125 DSA

HashMap and Heap Continued

Date - 29 Jan
Day → Saturday

Module → HashMap and Heap

Questions →
1. Merge K Sorted Lists
2. Sort K-Sorted Array.
3. Median priority Queue.

Ques. 1 MERGE K-SORTED LISTS

Approach 1 → पहले सारी list के सारी element pq में डालो और उसे pq से remove करके AL → result AL में डालो. But here Space complexity is very high.

list here is Arraylist. basically हमें 2D Arraylists दी Arraylist given होगी;

हमें K sorted list given होगी।
हमें इन k sorted lists को merge करके 1 sorted list बनानी है। TC → nlogn SC → K.

K=4
List l1 = 10, 20, 30, 40.
List l2 = 5, 7, 9, 11, 19, 55, 57.
List l3 = 1, 2, 3.
List l4 = 32, 39

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

Approach 2.

तो हमे K lists given है जो कि sorted होगी।

* तो हम एक Priority Queue बनाएँगे जिसमे हर list का first element डालेंगे, इस Priority Queue में priority दी गई है smaller elements को।
तो हमारे पास priority queue में K element है।
↳ Because of K number of lists

Now, we will remove the element with highest priority and add 1 element in the priority queue (from the same Array list to which the removed element belongs)

This will be done in a loop unless the priority queue becomes empty

In this way, we will have our final sorted list

Result

→ We will return an Arraylist
↳ This Arraylist will be the final arraylist to have the sorted list of all elements of all arraylist

हमें हर element के साथ ये track रखना पड़ेगा कि वो कौनसी Arraylist का data है और क्या वो Arraylist का last index है?

↳ इन सब बातों का track रखने के लिए हम 1 Pair class बनायेंगे

Pair class में हम रखेंगे data → data is the value at di
data index (di) → index of li
list index (li)
↳ to store the index of arraylist.

to store the index of data in the li list of arraylist

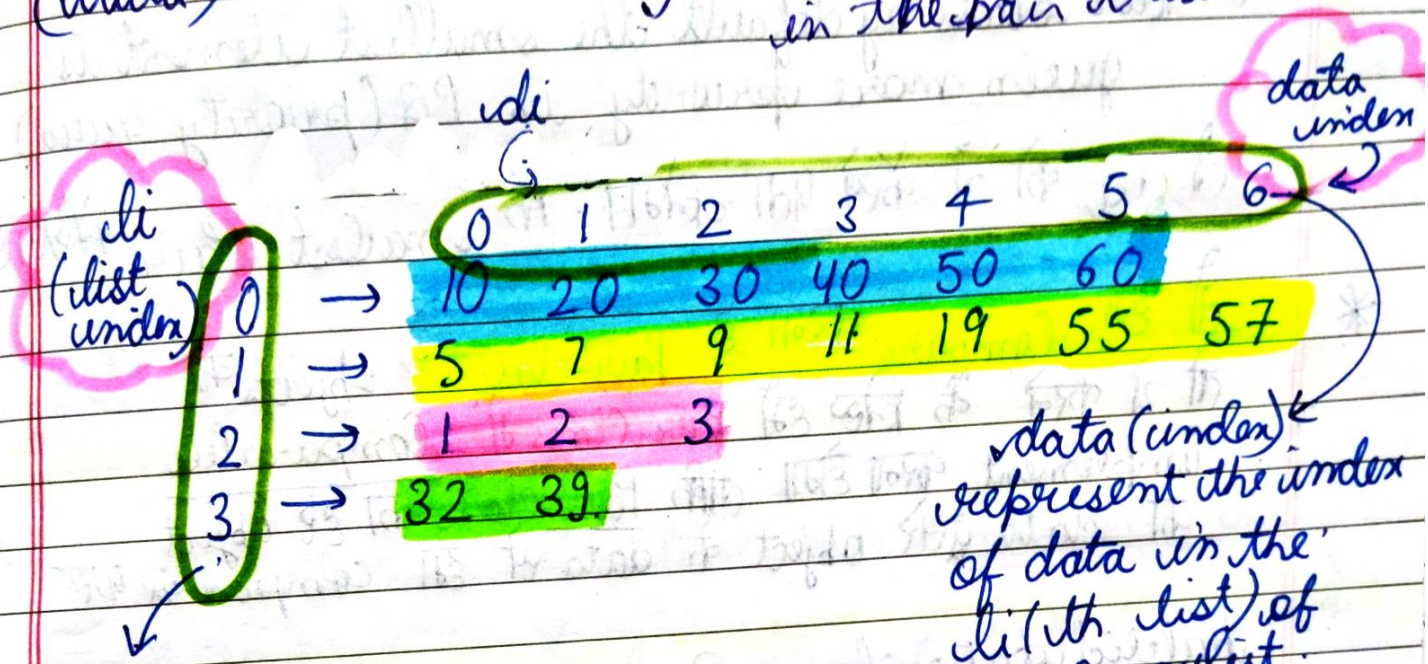
Because अगर list का next element हमें PQ में add करना है, और next element null है because this element is the last element (which is removed) तो nullpointer exception आ सकती है।

Approach.

① इसमें Pair class बनाएंगे

li (list index) → li
di (data index) → di
data → data

} We have these 3 things in the Pair class



K list are provided in Arraylist. So li (list index) will store the index of li(th) list of arraylist. And therefore, it help us to keep track of the position in the Arraylist. (इस Arraylist में कौन सी list पर travel करना है)

data (index) represent the index of data in the li(th list) of Arraylist.

Data is the value, which is an element of li(th) list of Arraylist

Pair class में 3 चीज़ store होती है → li (list index)
→ di (data index)
→ data

data is the value at di th index of ith list

② हम एक PriorityQueue बनाएँगे of type Pair class. तो हम priority queue में Pair class के objects डालेंगे But Priority Queue में हमे ऐसा data डालना होता जिसकी priority PriorityQueue को पता चल सके।

तो ..... by default the smallest element is given more priority in PQ (priority queue)

तो PQ को ये कैसे पता चलेगा कि smallest Pair कौनसा है

* तो हमे Compare करना है Pair class के objects को तो ये करने के लिए हमें Pair Class में Comparable implement करना होगा ताकि Pair class का हर object का data दूसरे object के data से हम compare कर पाए

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

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

अगर हम किसी class में comparable implement करते है तो उसमे compareTo function आना ही चाहिए

o means other object of class Pair

अगर ये difference +ve मिला तो this is Bigger..

difference is -ve तो this is smaller

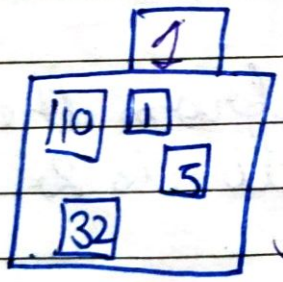
Difference is 0 तो this and other is equal

Smaller elements | Pair are given more Priority

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

अगर हम other.data - this.data return करते हैं, so this means that we are giving priority to greater element.

⑤ So, first of all, we will create a Priority Queue और उसमे हर list के first element के लिए Pair बनाके PQ में डालेंगे।



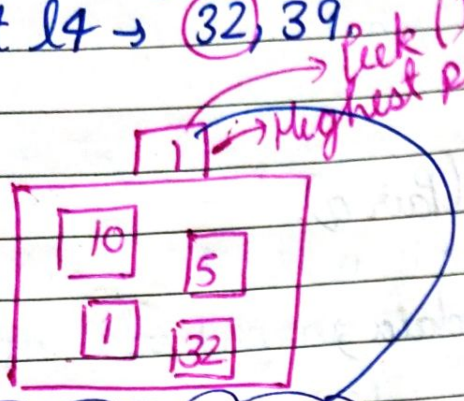
तो हमने हर element के लिए Pair बनाया where the li (list index) will be the index of list to which the element belong.

④ Now, we will also create a result Arraylist. अब हम PQ में एक element remove करेंगे और एक new element PQ में डालेंगे (from the same list to which removed element belong).

Only if the removed element was not the last element of the list it belonged to.

DRY RUN

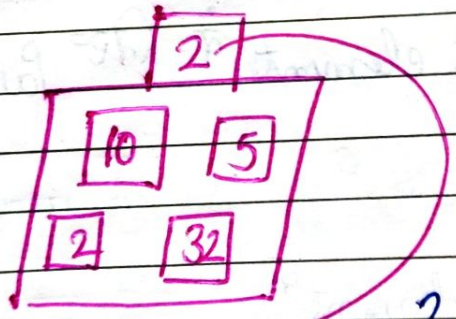
li=0 list l1 → 10, 20, 30, 40, 50.
li=1 list l2 → 5, 7, 9, 11, 19, 55, 57.
li=2 list l3 → 1, 2, 3.
li 3 list l4 → 32, 39.



Result Arraylist 1

→ 1 is removed from list where index is 2.

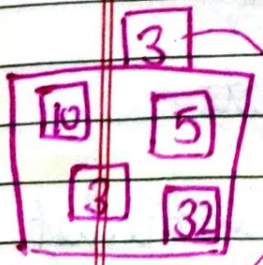
Now; next element will be added from the same list where li=2.



2 is removed from the list where li is 2.

Result Arraylist 1 2

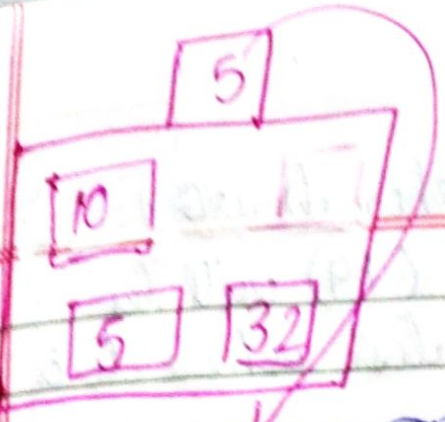
→ Now, 3 with be added. from list where li is 2.



3 is removed from the list where li is 2. Now 3 is the last element from the's list. So now no element will be added.

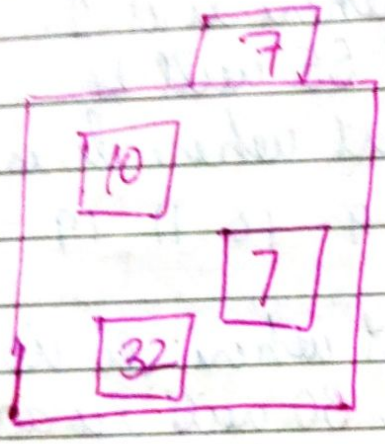
Result Arraylist 1 2 3

5 is removed from AL where li is 1, now next element will be added from the same list
So now 7 will be added in the list.



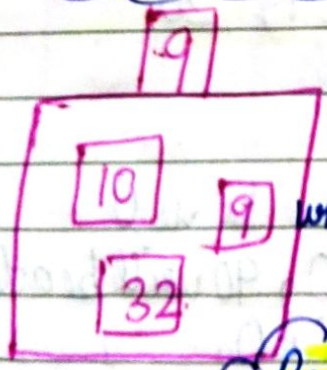
Result AL → 1 2 3 5

7 is from list where li is 1.
Now 7 is at the peek of priority in the PQ, so 7 will be removed now next element 9 will be added in the PQ which is from the same list.
So now, 9 will be added.



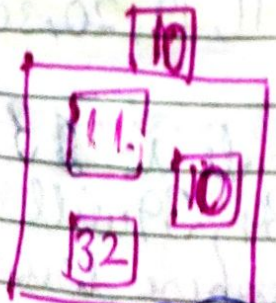
Result AL → 1 2 3 5 7

9 is from list where li is 1.
So 9 will be removed & now 11 is added which is next element from the list where li is 1.

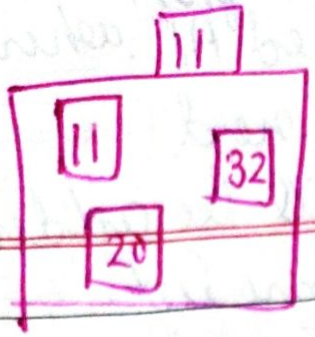


Result AL → 1 2 3 5 7 9

10 is from list where li is 0.
So, 10 will be removed from PQ & now next element (20) will be added from the list where li is 0.

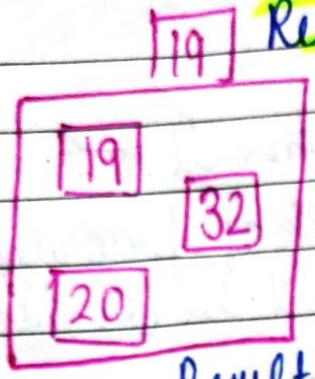


Result AL → 1 2 3 5 7 9 10



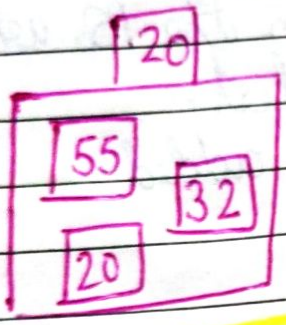
11 is from list where li is 1.
So next element (19) will be added from list where li is 1.

ResultAL → 1 2 3 5 7 9 10 11



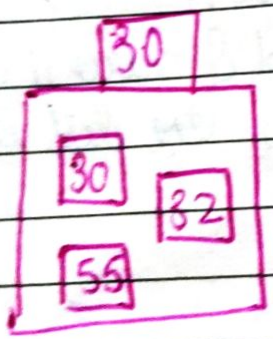
19 is from list where li is 1.
So next element 55 will be added from list where li is 1

ResultAL → 1 2 3 5 7 9 10 11 19



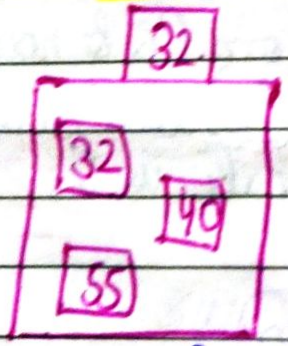
20 is from list where li is 0.
So, next element 30 will be added from list where li is 0.

ResultAL → 1 2 3 5 7 9 10 11 19 20.



30 is from list where li is 0.
So, next element is 40, 40 will be added from list where li is 0.

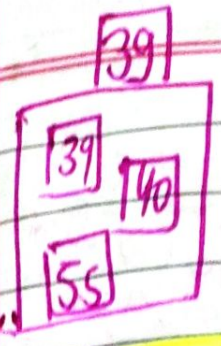
ResultAL → 1, 2, 3, 5, 7, 9, 10, 11, 19, 20, 30.



32 is from list where li is 3
So, next element is 39, 39 will be added from list where li is 3.

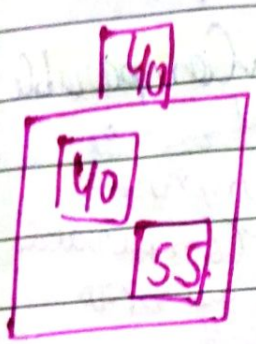
Result AL → 1 2 3 5 7 9 10 11 19 20 30 32

39 is removed from PQ



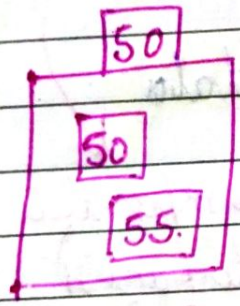
39 is the last element of list where li = 3.
So now, no new element will be added.

Result AL → 1, 2, 3, 5, 7, 9, 10, 11, 19, 20, 30, 32, 39.



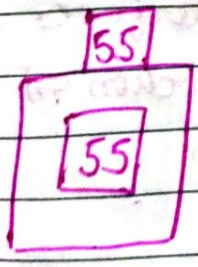
40 is removed from PQ.
40 is from list where li is = 0.
So, next element 50 will be added in pq from list where li = 0.

Result AL → 1 2 3 5 7 9 10 11 19 20 30 32 39 40



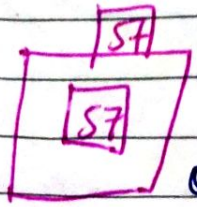
50 is removed from PQ.
50 is from AL where li is 0.
Now no new element is added in PQ because 50 is last element of list where li is 0.

Result AL → 1, 2, 3, 5, 7, 9, 10, 11, 19, 20, 30, 32, 39, 40, 50.



55 is removed from PQ, the li of 55 is 1
57 is the next element to be added in PQ as li of 57 is 1.

Result AL → 1 2 3 5 7 9 10 11 19 20 30 32 39 40 50 55



57 is removed from PQ. 57 is the last element of list where li is 1. So there is no next element to be added in PQ, so PQ becomes empty

Result AL → 1 2 3 5 7 9 10 11 19 20 30 32 39 40 50 55 57

We get our final arraylist

```
import java.util.*;

public class Main
{
    public class Pair implements Comparable
    {
        int data;
        int di;
        int li;

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

Comparalele is an interface (Interface is a contract of function)

Pairs को compare करने के लिए Comparalele interface का use करना पड़ेगा

Comparalele Interface अगर हम implement करते हैं तो उस function को भी mention करने पड़ेंगे

Comparalele Interface में compareTo function का बस नाम है।

Interface में function की Body नहीं होती, Interfaces में बस नाम होता है function का

अगर हम PQ में integer data नहीं डालते तो PQ assume करती है कि PQ के data की class में हमने Comparalele implement किया होगा क्योंकि Comparalele implement करना जरूरी है ताकि PQ Pairs को compare कर पाए

Date
Page No.

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

→ हमने result AL बनाई (↳ ये Arraylist की Arraylist pass की गई है as a Parameter

```
{
ArrayList<Integer> resultAL = new ArrayList<>();
PriorityQueue <Pair> pq = new PriorityQueue<>();
```

↳ ये एक Priority Queue है जिसमें हम Pair store करेंगे

```
for(int li = 0; li < lists.size(); li++)
{
    Pair p = new Pair();
    p.li = li;
    p.di = 0;
    p.data = lists.get(li).get(0);
    pq.add(p);
}
```

हमने सभी list के first element के लिए Pair बनाए और उन Pairs को PQ में add करदिया

```
while (pq.size()>0)
{
    Pair removedPair = pq.peek();
    pq.remove();
    resultAL.add(rp.data);

    removedPair.di++;
    if(removedPair.di < lists.get(removedPair.li)
                              .size())
    {
        removedPair.data = lists.get(removedPair.li)
                              .get(removedPair.di);

        pq.add(removedPair);
    }
}
} return resultAL;
```

→ जब तक Pq empty न हो जाए तब तक loop चलेगा और हम 1 element को remove करेंगे with highest priority और इसी element वाली list का next element डालेंगे।

हम removed element के Pair में ही new element का data डालके उसे add करदो

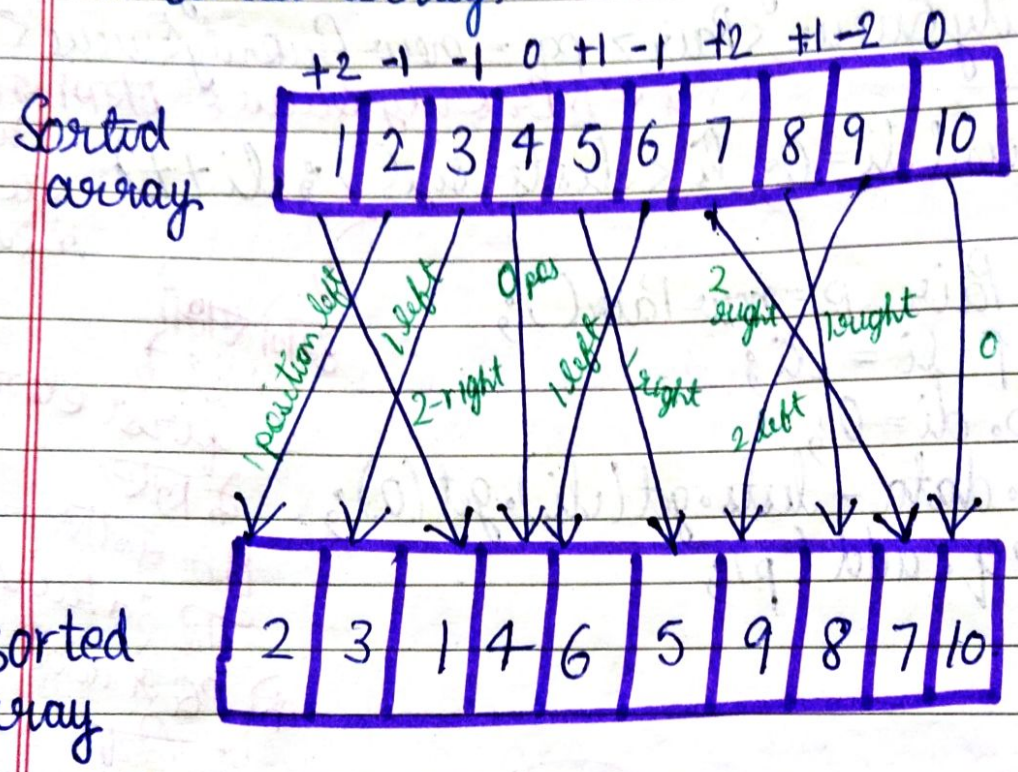
पहले check करोकि क्या उस list में element बचे है, अगर element बचे है तो next element को PQ में डालो

→ last में AL को return करदो।

Ques-2. # SORT K-SORTED ARRAY

हमे एक array given होगा और एक integer K given होगा

The integer K denotes that the array is a K-sorted array.



हमे एक K-Sorted array given होगा

K-sorted array में हर element अपनी position से (0 to K) K-position left या K-position right पर move हो सकता है।

If K=2, तो हर element को 1 या 2 position left या right move कर सकते है।

It is also possible that element does not move from ither position.

i.e. We can move 0/1/2 positions if K=2.

Approach

1. We create a priority queue of size K+1. (K+1 size की ArrayList बनाते हैं)
2. Now, we will remove 1 element from the priority queue. (Priority Queue में से 1 element remove कर दिया और उस removed element को print कर दिया)

This will be the smallest element as the removal from the priority queue gives the smallest element or the element of highest priority. Now, we have K elements remaining in the priority queue. The removed element will be

3. So, we insert the next element in the Priority Queue. and the size again becomes K+1.

Unless the PQ becomes empty of

Now, we will keep on removing the element from PQ. and adding next element of the array (unless we have no next element in the array to be added in the PQ).

* So, we repeat the same procedure until we cover the entire array.

→ इस Priority Queue में से 1 element remove करेंगे और Array का 1 element PQ में add करते चलेंगे जब तक Array के सब elements को visit न कर लें।

जब तक PQ empty न हो जाए & print the removed element.

4. If the array is completely visited & some elements are left in the Priority Queue, we empty the PQ completely (& print the removed element) and then this procedure stops here.

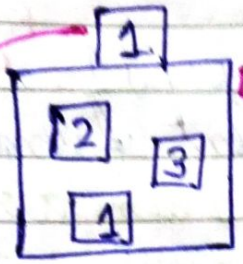
DRY RUN.

Date
Page No.

① 1 element will be removed & printed

2	3	1	4	6	5	9	8	7	10

K=2.



} Added K+1 i.e 3 elements in the PQ.

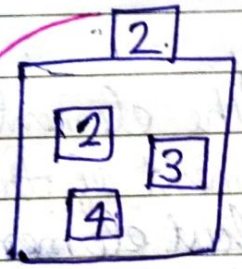
1

} 1 is removed & printed
Next element (4) will be added in the PQ.

2	3	1	4	6	5	9	8	7	10

1 has already been printed

② 1 element will be removed and printed

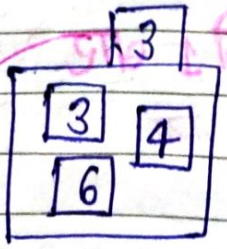


} We still have 3 elements in the PQ.
2 has the maximum Priority.
So we will remove 2 from PQ & print it.
Now next element (6) will be added in the PQ.

1 2.

↳ 2 is also printed

③ 1 element will be removed & printed



2	3	1	4	6	5	9	8	7	10

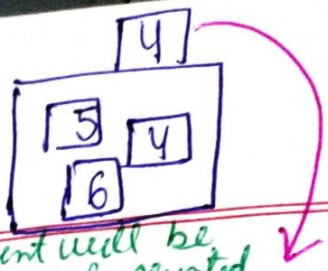
1 and 2 has already been printed

Now 3 is printed & removed.
So the next element of Array (5) will be added in the PQ.

1 2 3.

Date / /
Page No.

④



1 element will be removed & printed

1 2 3 4

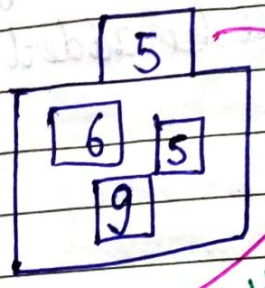
2	3	1	4	6	5	9	8	7	10

1,2,3 have been printed

Now 4 is also printed & removed from the PQ

So next element of Array (9) will be added in the PQ

⑤



1 element will be removed and printed

1 2 3 4 5

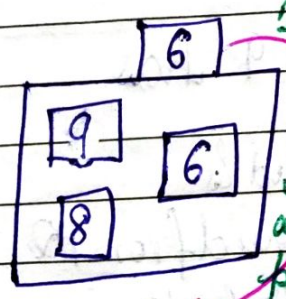
2	3	1	4	6	5	9	8	7	10

1,2,3,4 have already been printed.

Now, 5 is printed and removed from PQ.

So, next element of Array (8) will be added in the PQ.

⑥



1 element will be removed and printed

1 2 3 4 5 6

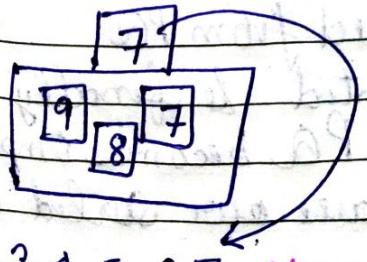
2	3	1	4	6	5	9	8	7	10

1,2,3,4,5 have already been printed

Now, 6 is printed and removed from PQ.

So, next element of Array (7) will be added in the PQ

⑦



1 2 3 4 5 6 7

2	3	1	4	6	5	9	8	7	10

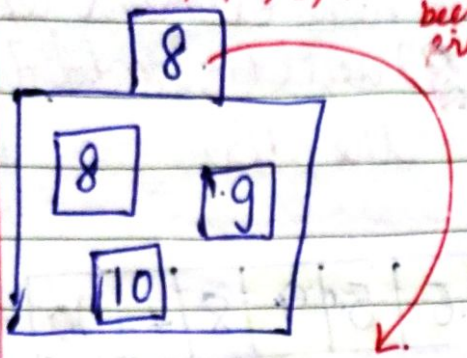
1,2,3,4,5,6 have already been printed

Now, 7 is printed and removed from PQ.

So, next element of Array (10) will be added in the PQ.

(8) | 2 | 3 | 1 | 4 | 6 | 5 | 9 | 8 | 7 | 10 |

1, 2, 3, 4, 5, 6, 7 have been printed



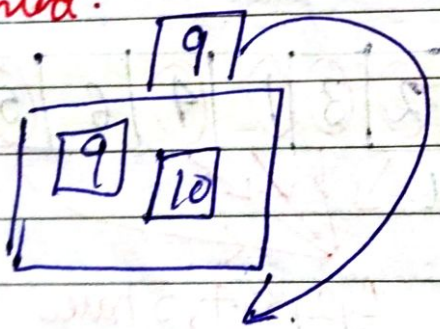
Now, 8 will be printed and removed from PQ.
So, next element of array will be added.
So, nothing will be added.

1 2 3 4 5 6 7 8

(9) | 2 | 3 | 1 | 4 | 6 | 5 | 9 | 8 | 7 | 10 |

Now, 9 will be printed & removed from PQ.

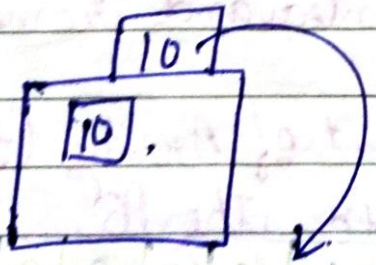
1, 2, 3, 4, 5, 6, 7, 8 have been printed.



Now 9 has been printed & removed from PQ so Only 10 is remaining in PQ.

1 2 3 4 5 6 7 8 9

(10)



Now 10 will be removed from PQ & printed & finally our PQ becomes empty. and we have our sorted array

1 2 3 4 5 6 7 8 9 10.

So we have pushed element in PQ & then removed them

So the tc for removing an element from PQ

↓ Time complexity → $O(\log k)$.

↳ from k element we remove element

So, TC. for K elements → $O(n \log k)$

Space Complexity → $O(k)$.

↳ Because only K elements were stored in PQ.

SORT K-SORTED ARRAY
<code>

```
import java.util.*;
public class Main{

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

    int k= scan.nextInt();

    PriorityQueue <Integer> pq= new PriorityQueue
                                <>();
    for(int i=0; i<=k; i++)
      {
        pq.add(arr[i]);
      }

    for(int i= k+1; i<arr.length; i++)
      {
        int val= pq.peek();
        System.out.println(val);

        pq.remove();
        pq.add(arr[i]);
      }
  }
```

हमने K sorted array input लिया।

हमने k की value input की।

एक Priority Queue बनाई

array के K+1 elements को pq में डाला

pq का peek element दे दीजिए और उसे pq से remove करो

जब तक array के सभी element remove नहीं pq में से डालते तब तक loop चलेगा।

array के next element को pq में डालो

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

अगर pq में अब भी कुछ elements बचे हैं तो print करदो और उन्हें pq में से remove करदो

(3)

MEDIAN PRIORITY QUEUE

We have to create our own priority queue but this is not a normal priority queue. This is a median priority Queue.

मतलब इस PriorityQueue में smaller या greater element को पहले priority नहीं मिलेगी। Median Priority Queue में median को ही सबसे पहले priority मिलेगी।

तो हर बार remaining elements का median ही return किया जाएगा।

10 20 30 40 50

NORMAL PRIORITY QUEUE	Median Priority Queue
peek() → 10.	peek() → 30
remove() → 10.	remove() → 30.

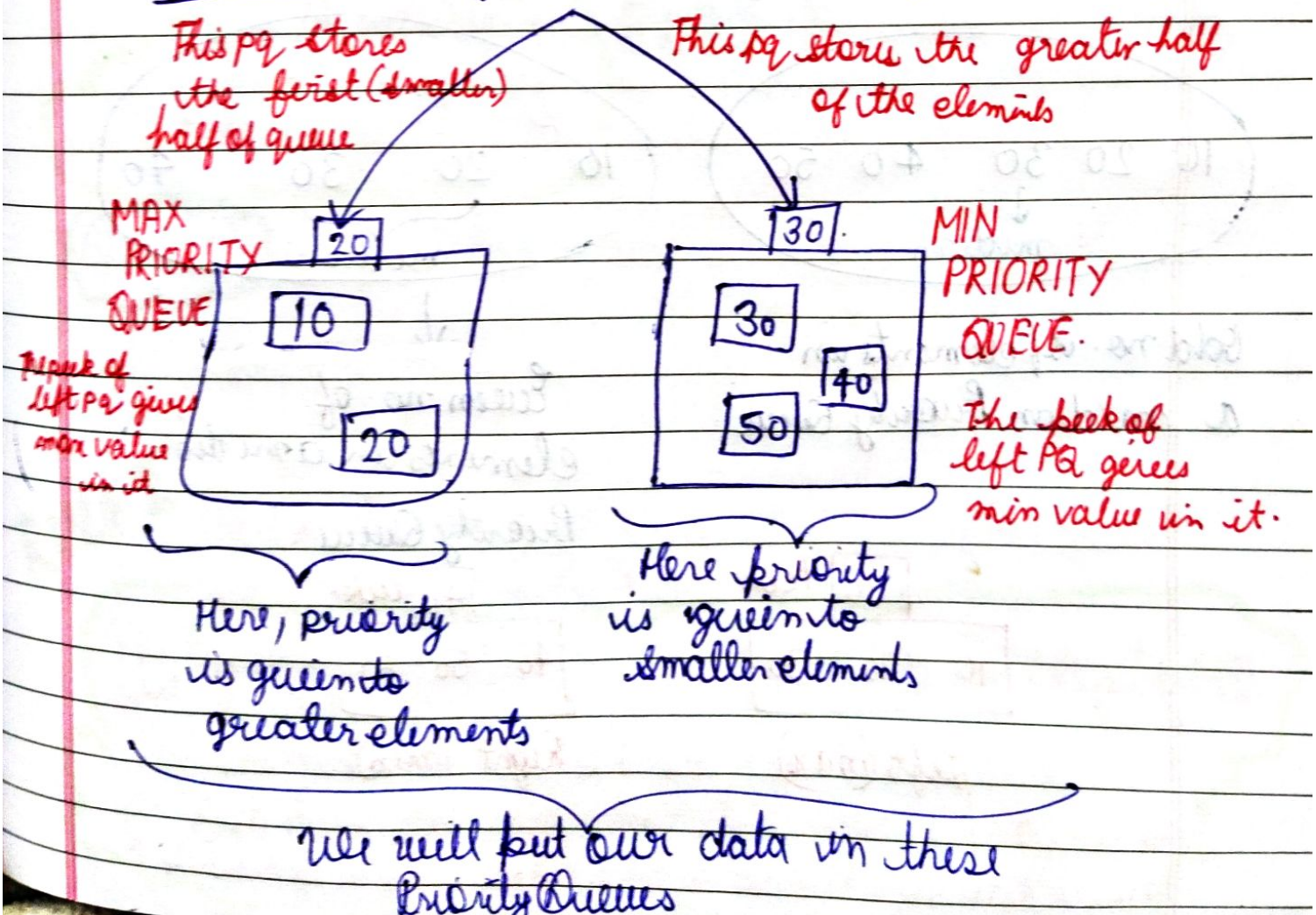
In A median priority will see just like a normal priority queue, but in the normal priority queue when we call the remove function, it removes the smallest element (10) from the queue because By default In Priority Queue, the smallest element has the highest priority queue.

But, In case of Median Priority Queue, when we call the remove() function, the remove function will remove the median of all elements similarly the peek() function returns the median of all elements.

* The Time Complexity of peek() and remove() function should be same as of a normal priority queue. peek() → O(1), remove() → O(log n)

Approach

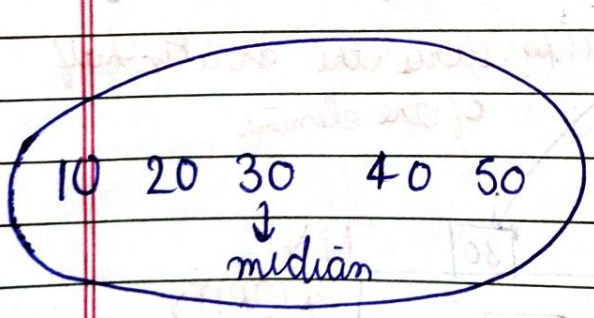
1. Median Priority Queue is basically 2 priority queues



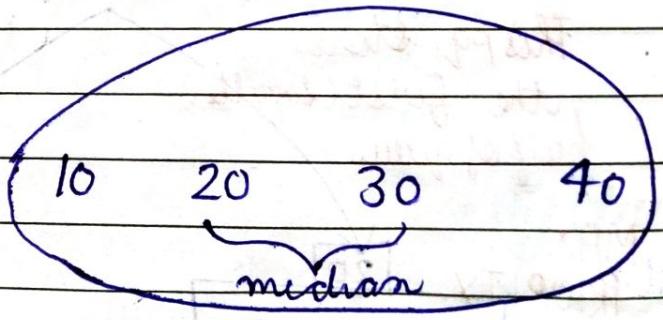
	Left PQ.	Right PQ.	
peek	n	n	(left)
	n+1	n	(left)
	n	n+1	(right)

→ If the median queue has even no. of elements i.e. If both Left PQ & Right PQ has equal size then Left PQ से element remove होगा and that element will be printed as a median.

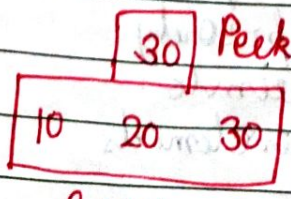
→ If there are odd number of elements in the pq if the left priority / right priority queue has n+1 elements & the other priority queue right / left pq has n elements, then in this case, the pq with n+1 element will remove an element.

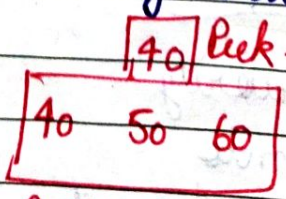


Odd no. of elements in a median Priority Queue.



Even no of elements in a median Priority Queue.





DRYRUN

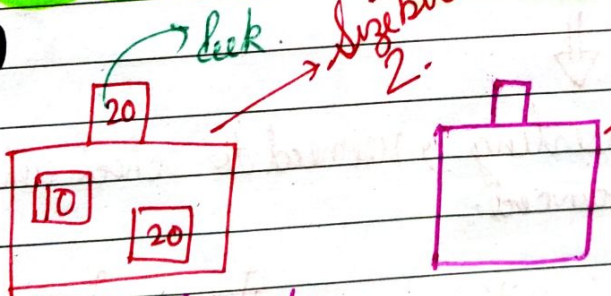
Our median priority queue is empty.
Initially, both our left and right priority were empty.

ADD(10)

By default, an element is inserted in the left PQ.

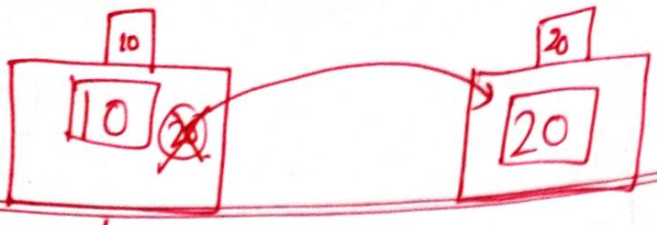
RULE 1

ADD(20).



RULE 2 By default, element is inserted in left PQ

But if, The difference is size of Left and Right PQ. Should be 1
But here the differnce becomes 2. So, remove 1 element from queue having more element & put it in another



Since left PQ is max PQ
So 20 will be removed from it & it will be pushed in right PQ.

तब Left और Right PQ का Size equal है।

RULE 3

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

element > Right.peek() ⇒ Insert in Right PQ.

When the element being added is lesser than the peek() of left PQ, then the element will be inserted in left PQ.

element < left.peek() ⇒ Insert in left PQ.

⇓

After Inserting, we need to check whether the left & right PQ is Balanced.

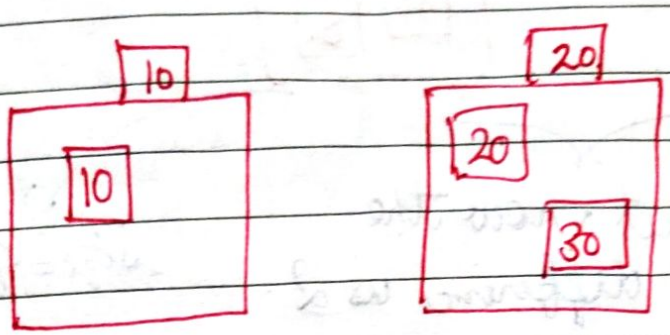
↳ तब Size Difference 2 से ज्यादा न हो

जिस queue में ज्यादा element है, उस queue में से element remove करके दूसरी queue में add करदो।

If the element < Right.peek() & element > left.peek()
तो By default, उस element को left PQ. में add करेंगे।
Then, we will check if the PQ has size differ <= 1

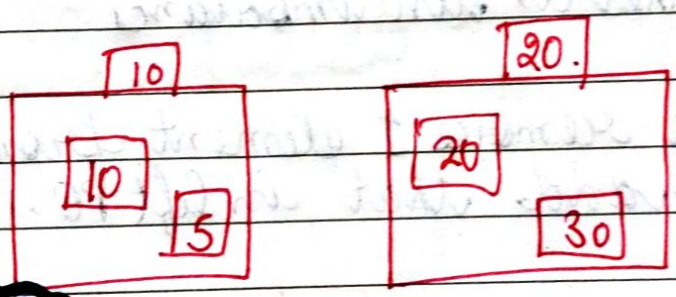
ADD(30)

↳ 30 is greater than Right.peek().
So 30 will be inserted in the Right PQ



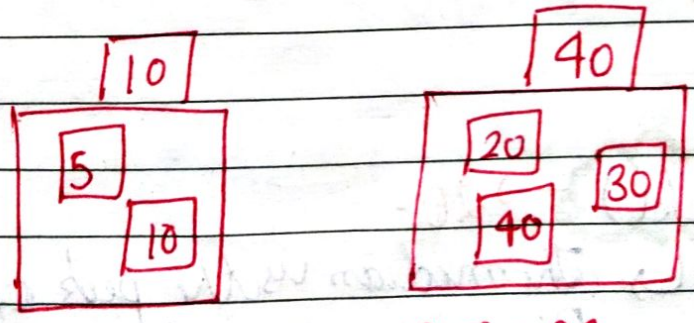
ADD(5)

↳ 5 is lesser than Left.peek()
So 5 will be inserted in the Left PQ.



ADD(40)

↳ 40 is greater than Right.peek()
So 40 will be inserted in Right PQ.

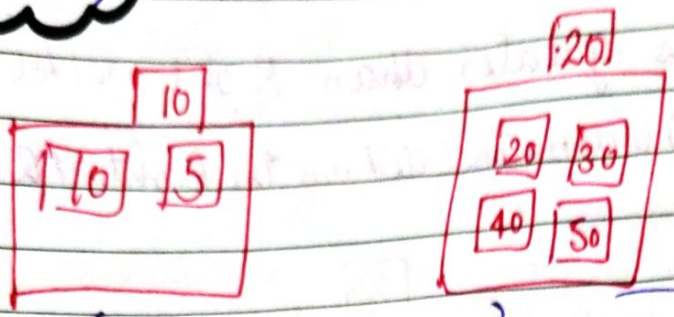


Left PQ
Maximum Priority Queue

Right PQ
Minimum Priority Queue

ADD (50)

50 is greater than Right.peek(). So 50 is added to the Right PQ.



But now the size difference is 2.

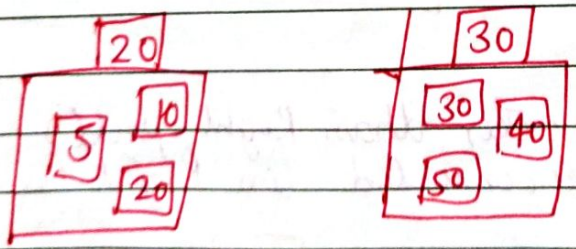
Size = 4.

Size = 2

So there isn't the accurate balance between size.

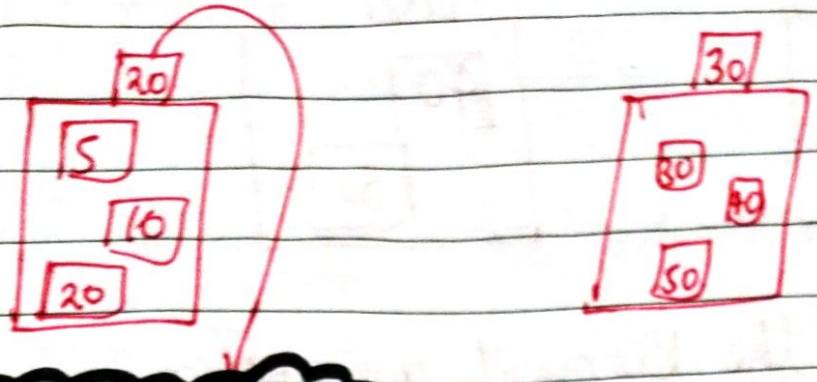
So the Right PQ has Size 4.
and the Left PQ has Size 2.
So, there is an imbalance.

So, just remove 1 element from Right PQ and add that in left PQ.



MedianPQ.peek() → 20.

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



Median PQ. Remove() → 20

Median. Remove() → When there are equal elements in both Left & Right PQ (even no. of elements in total).

Then we will remove the element from left PQ.

Right. remove() → 20 is removed.

MEDIAN PQ. REMOVE() → 30.

Size = 2 Size = 3

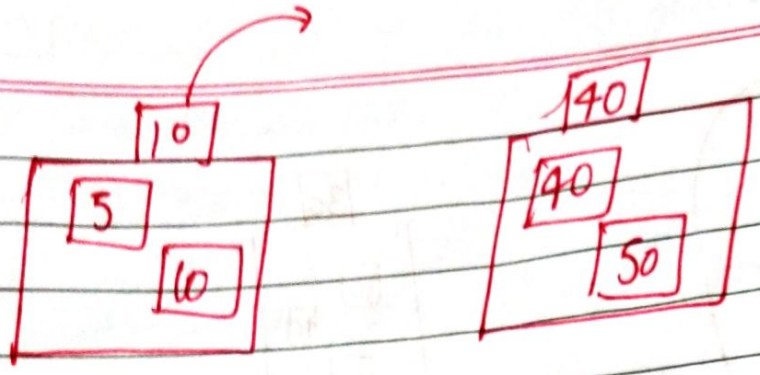
5 element → When the Size is odd (when size of left & right PQ is not equal)

Then remove the element from PQ having more elements.

Right Size > left Size

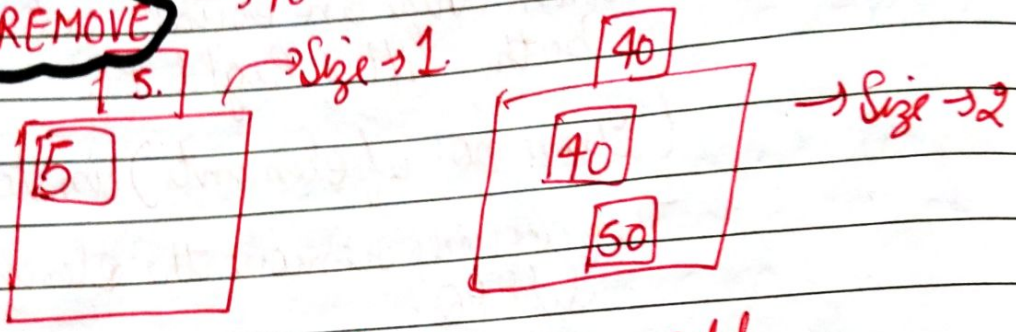
So Remove element from Right PQ..

MEDIANPQ.REMOVE() → 10.



Now again the Elements are even.
So remove the element from left PQ.

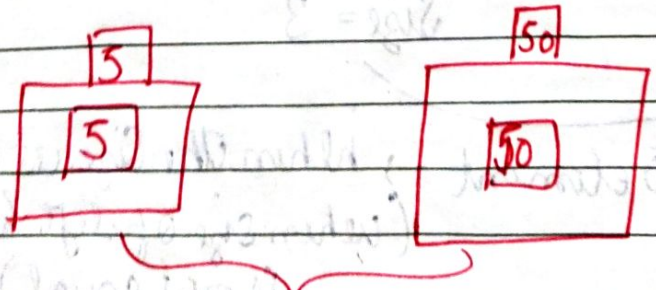
MEDIANPQ.REMOVE → 40



Now, the elements are odd.
So the element will be removed from PQ where the Size is more.

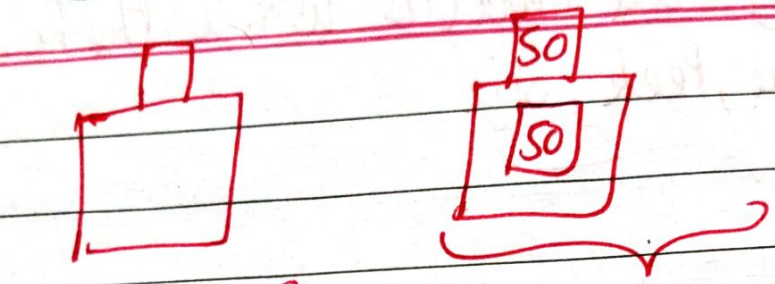
SizeofRightPQ > SizeofLeftPQ

MedianPQ.Remove() → 5



Even element So, element will be removed from left PQ.

MedianPQ.remove() → 50.



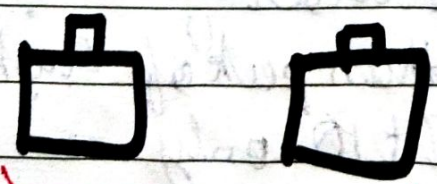
Left PQ has 0 element

Right PQ has 1 element

Size of Right PQ > Size of Left PQ.

So remove the element from Right PQ.

Now, Both our left & Right PQ is empty



Empty PQ's

In the Question, we need to write the following functions
Add, Remove, Peek, Size

Add() → Add an element in Median PQ.
Basically we have 2 PQ behind

1. The elements are added by default in left PQ.
2. The difference b/w size of 2 PQ should be maintained to 1
   If the difference increases, shift one element from the queue with higher no. of elements to the one with the lower.
3. If the input element is greater than peek of right PQ, then it is added to Right PQ only.

Peek() → 1. If there are even numbers of elements (equal in both PQ) then the median is the peek() of the left Priority Queue. else if there are odd number of elements, then the median is the peek of the Queue with more elements.

Remove() → 1) If there is equal no. of elements in left & right PQ. (total even no. of elements in both PQ)
then the median will be removed from the left PQ
else we remove from the PQ which has more number of elements.

Size() → If We will just to add the Size of left PQ & Right PQ.

Date / /
Page No.

Time Complexity

1. Add() → O(log n)
2. remove() → O(log n)
3. peek() → O(1)
4. Size() → O(1).

} Similar to normal pq.

Space Complexity

↳ The Space Complexity is O(1) as we have not used any extra space for any function (add, remove, peek, size). We have just taken 2 PQ but they will not be counted in Space Complexity as it is not extra space.

The question asked to create a Median Priority Queue and we have just done that using 2 PQ's.

MEDIAN PRIORITY QUEUE <code>

Page No.

```
import java.util.*;
public class Main {


public class Main
{
  public static class MedianPriorityQueue
  {
    PriorityQueue<Integer> left;
    PriorityQueue<Integer> right;

    public MedianPriorityQueue()
    {
      left = new PriorityQueue<>(Collections.reverseOrder());
      right = new PriorityQueue<>();
    }

    public void add(int val)
    {
      if(right.size() > 0 && val > right.peek())
      {
        right.add(val);
      }
      else
      {
        left.add(val);
      }
      handleGap();
    }
```

Annotations:

- left → ये left PQ. है having Max Priority
- right → ये Right PQ है having min priority
- constructor of medianpq class
- left = new PriorityQueue<>(Collections.reverseOrder()) → left PQ. having max priority
- right = new PriorityQueue<>() → giving priority to smaller element
- add(int val) → function of median pq
- अगर medianPQ के object पर add func call है
- right.size() > 0 → Right PQ की peek check करने से पहले ये check करना है कि उस PQ में कोई element है या नहीं, otherwise if the Right PQ is empty we will get null pointer exception
- right.add(val) → We have called add func of right pq.
- val की value अगर right PQ की peek से ज्यादा है तो Right PQ में add करेंगे otherwise left PQ में
- handleGap() → अगर left और Right PQ का size diff 1 से ज्यादा हो गया है तो हमें Balance करना पड़ेगा

```
private void handleGap()
{
  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();
  }
}
```

Date / / Page No.

अगर left का size right से 2 element ज्यादा है तो left से एक element right में store & vice versa.

अगर remove function call किया गया है तो पहले check करेंगे कि median PQ में कोई element है (left और Right PQ अलग से size check होगा) अगर size 0 है तो we have no element to remove so we will simply print underflow. & return -1 as the removed element

else और Median PQ का size 0 नहीं है और left PQ का size right PQ से ज्यादा है या equal है तो we will remove the element from left PQ otherwise from right PQ.

```java
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();
}

}
```

हमने peek function में भी बिलकुल remove function वाला logic लगाया है

अगर size 0 है Median PQ का so there is no element to remove/peek. So, we will simply print Underflow & return -1 as peek

अगर left pq में right pq से ज्यादा element या equal है तो we will return the peek of left pq otherwise we return the peek of right pq

The Size of median PriorityQueue is the sum of size of left PriorityQueue & right PriorityQueue.

→ Here this is the closing Bracket of Median Priority Queue Class

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

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 ("puk"))
    { int val = queue.remove();
      if (val != -1)
        { System.out.println ( val);
        }
    }
```

Date. / /
Page No.

↳ इसमें MedianPriorityQueue class का object बनाया

Object का add function call किया

इसमें Median PQ. class में add, remove, size, peek function define किये हैं

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

str=ler.readLine();
  }
}
```