→ Today's Content:

a) k Smallest elements → { Interview Questions ✓

b) Median of every prefix Subarray

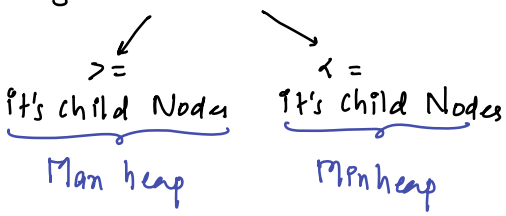
c) Merge N Sorted arrays } Wednesdays session

d) find k^{th} smallest pair sum }

Revison:

Heap: a) CBT → Complete Binary Tree

b) For every node value



Functionalities:

: insert() → $\log_2^N$

: getMin()/getMax() → 1

: deleteMin()/deleteMax() → $\log_2^N$

Given N distinct Elements, find k Smallest Elements in array k < N

ar[10] : { 8 3 10 4 11 2 7 6 5 1 }

k=4
↳ { 1 2 3 4 }

ar[9] : { -3 6 2 0 8 7 10 4 }

k=3
↳ { -3, 0, 2 }

Ideas:

1) In Every Step, Iterate in array, get smallest & swap with i^{th} index
Repeat above process by k times
TC: k * {N} SC: O(1)

2) Sort entire array & get first k Elements
TC: NlogN SC: ↱ O(N) : merge Sort
↳ O(1) : Quick Sort

3) Insert all array elements in min heap, & Apply del-min k times

min_heap : { -3, 6, 2, 0
8, 7, 10, 4 }

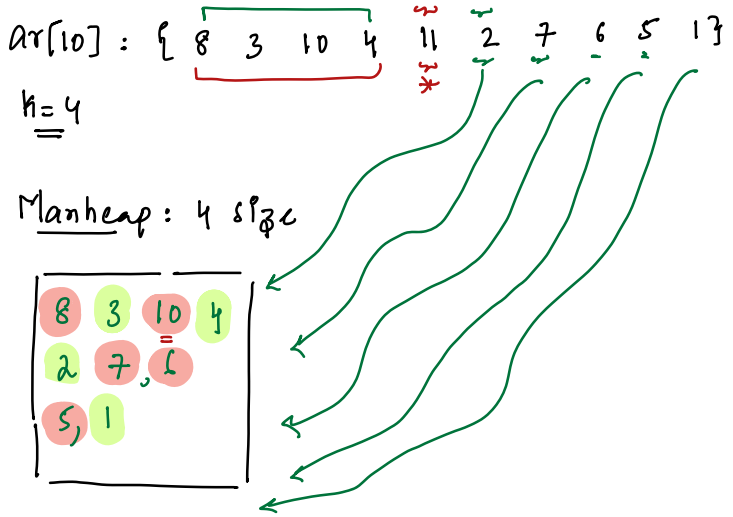
TC: NlogN + { k * 1 + k * logN }

TC: NlogN + klogN

SC: O(N)

get_Min(): -3, delete_Min(): -3
get_Min(): 0, delete_Min(): 0
get_Min(): 2, delete_Min(): 2

4) Using Manheap:

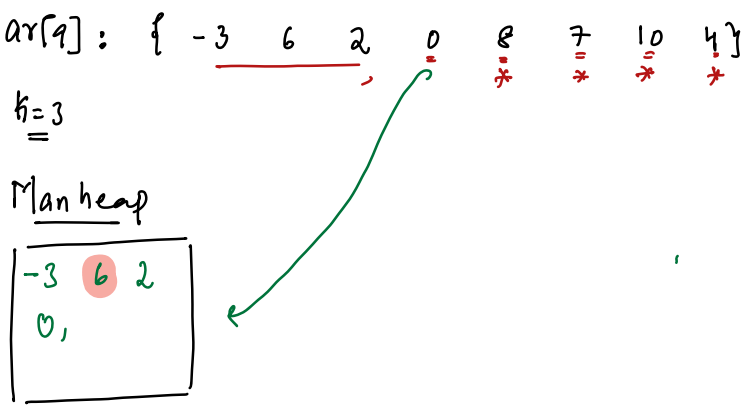


obs1: ele > get_man() in that ele cannot be our ans

obs2: ele < get_man() in that ele may be our ans

a) insert ele in heap

b) delete man from heap



Idea: Manheap <int> mh;

insert first k elements → TC: k log k

for all remaining elements (N-k) → TC: [N-k](log k)] TC: N log k

→ check if element in possible ans SC: O(k)

:No → Neglect

:Yes → {deleteMan() & Insert}

↳ O(log k)

{Better than above approach}

Median: point can divide data into 2 halves

ar[5] = { 2 9 6 4 5 }

↳ = { 2 4 5 6 9 }

1st < 2nd

ar[7] = { 2 -1 10 4 15 3 -2 }

↳ = { -2 -1 2 3 4 10 15 }

1st half, 2nd half

ar[6] = { -1 10 3 9 6 2 }

↳ = { -1 2 [3 6] 9 10 }

$\frac{3+6}{2} = \frac{9}{2} = 4$

1st < 2nd

ar[8] = { 2 4 -3 12 6 24 20 10 }

↳ { -3 2 4 [6 10] 12 20 24 }

$\frac{6+10}{2} = \frac{16}{2} = 8$

1st < 2nd

28) Given an array find median of all prefix Subarrays

```
          0  1  2  3  4
ar[5] = { 9  6  3  10 4 }
```

↳ Subarrays starting at index 0

Prefix Subarrays	Median
{9} → {9}	9
{9 6} → {6 9}	15/2 = 7
{9 6 3} → {3 6 9}	6
{9 6 3 10} → {3 6 9 10}	15/2 = 7
{9 6 3 10 4} → {3 4 6 9 10}	6

Solutions:

1) Sort every prefix Subarray & get median

TC: N * {N log N} SC: O(N)

2) ar[5] = { 9 6 3 10 4 }

Step1 : 9

Step2 : 9 6 → 6 9

Step3 : 6 9 3 → 3 6 9

Step4 : 3 6 9 10 → 3 6 9 10

Step5 : 3 6 9 10 4 → 3 4 6 9 10

After each step in insertion sort for prefix Subarray get median

TC: $O(N^2)$ SC: O(1), we are sorting in the given array itself

optimization:

ar[9] ⇒ { 3 1 6 10 14 2 17 12 9 }

Median ⇒ { 1 2 3 6 9 10 12 14 17 }

Include median in 1st half

{ 3 1 6 2 } { 9 } { 14 12 17 10 }

1st half 2nd half

obs1: Max element in 1st half ≤ Min element of 2nd half

obs2: |Size of 1st half − Size of 2nd half| = 1

Median: Max element in 1st half

ar[10] = { 3 4 16 12 10 14 8 9 2 1 }

Median: { 1 2 3 4 8 9 10 12 14 16 }

1st median 1st half

2nd median 2nd half

3 2 1 4 12 14 16 10

obs1: Max element in 1st half ≤ Min element of 2nd half

obs2: |Size of 1st half − Size of 2nd half| = 0

Median: $\frac{\text{Max of first half} + \text{Min of 2nd half}}{2}$

Ex: ar[] : { 4 9 6 2 1 10 9 7 3 5 }

4 (1st) > 1st half max (2nd) | 9 (=) 6 (2nd) > 1st half max | 6 (2nd) < 1st half max (1st) | 2 (1st) < 1st half max | 1 (1st) > 1st half max (2nd) | 10 (=) 9 (2nd) > 1st half max | 9 (2nd) > 1st half max (2nd) | 7 (=) 3 (1st) < 1st half max | 3 (1st) 5 < 1st half max (1st half)

1st half

4 ~~6~~ 2, 1, ~~10~~

3, 5

2nd half

9 ~~6~~ → 1st hat

~~9~~, 10, 9, 7

6

operations:

- Insert()
- getMax()
- deleteMax()

} Maxheap

Min heap { Insert(), getMin(), deleteMin()

Median:

4, 13/2, 6, 5, 4, 5, 6, (6+7)/2, 6, (5+6)/2

Take care:

- ✓ Max of 1st half <= Min of 2nd half
- ✓ size(1st half) − size(2nd half) = 0 or 1

If equal divided:

Median = (1st half max + 2nd half min)/2

If size diff = 1

Median = (1st half max)

→ 11:18pm

```
int[] Running_Median(int ar[]){
    int n = ar.length;
    int ans[n];
    Maxheap<int> maxh; maxh.insert(ar[0])
    Minheap<int> minh;
    ans[0] = ar[0]
    i=1; i<n; i++){
        // Insert ar[i]
        if(ar[i] < maxh.get_Max()){
            // ar[i] belongs to 1st half
            maxh.insert(ar[i])
        }
        else{ minh.insert(ar[i]) }
        if( maxh.size() < minh.size()){
            // Transfer min element from minh to maxh
            int ele = minh.get_Min(); minh.delete_Min()
            maxh.insert(ele);
        }
        else if( maxh.size() - minh.size() > 1){
            // Transfer max from maxh to minh
            int ele = maxh.get_Max(); maxh.delete_Max()
            minh.insert(ele)
        }
        int s = (i+1)  # Total elements inserted in both heaps
        if(s%2==0){
            ans[i] = [maxh.getMax() + minh.getMin()]/2
        }
        else{ans[i] = maxh.getMax()}
    }
    return ans;
}
```

TC: $N \times [\log N + 2\log N + 1]$

TC: $O(N \log N)$ SC: $O(N)$

Step1: Inserting element

Step2: Balancing

Step3: Ans