Today's Content :

a) Intro

b) k Smallest Elements

c) Median of every prefix Subarray

Sunday 18th : Heaps Implementation : 10:30 AM → optional / Recording

Note: N is no: of elements

Req functions	getmin()	deletemin()	insert()	size()
list/Dy array	O(N)	O(N)	O(1): at last	O(1)
linked list	O(N)	O(N)	O(1): start last pointer	O(1) size var
Stack	*	*	doesn't satisfy req	
hashset	O(N)	O(N)	O(1)	O(1)
BinaryST	O(H)	O(H)	O(H)	O(1)
Treeset/ map	log(N)	log N	log N	O(1)
min heap	O(1)	log N	log N	O(1)

heap
- minheap: getMin() deleteMin() insert() size()
  TC: O(1) O(log N) O(log N) O(1)
- manheap: getMan() deleteMan() insert() size()

obs1: Traversing on a heap is not possible by above are only functions we

obs2: In heaps we can store duplicates

obs3: Heap memory is different from heap datastructure

Syntax prefer language: TODO: Read in your language of choice

```
min_heap<int> minh
                  // Type of data we store
man_heap<int> manh
```

Given N distinct Elements, print k Smallest Elements in inc from array k < N

ar[10] : { 8 3 10 4 11 2 7 6 14 1 }

k=4: {1 2 3 4}

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

k=3 : { -3 0 2 }

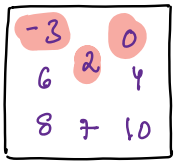
Ideas:

1. Sort arr[] & print first k elements TC: O(nlogn + k)
2. Insert all ele in min_heap & k times getMin() + deleteMin()

TC: N * logN + k * {1 + logN} ≈ O(N+k)logN SC: O(N)

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

min_heap

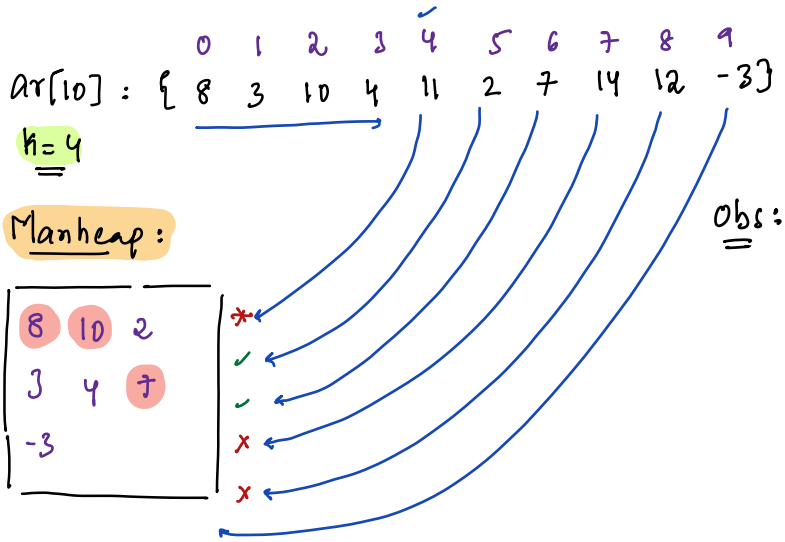


get_Min(): -3 delete_Min(): -3 deleted

get_Min(): 0 delete_Min(): 0 deleted

get_Min(): 2 delete_Min(): 2 deleted

3. Using Manheap:



Obs: If an ele > man
: ele cannot be ans
If an ele < man
: ele can be ans

Idea: TC: ~~k log k~~ + 2N log k - ~~2k log k~~ + ~~k log k~~ + N-k + ~~k~~ + k

TC: O(N log k) SC: O(k)

→ Manheap: Insert first k ele in manheap: TC: k log k

→ Iterate on rem arr[]: (N-k) * {1 + log k + log k}

: ele < man in Manheap:

: Insert ele in man heap } // we can interchange
: delete man from man heap } 2 steps.

: ele > man in Manheap:

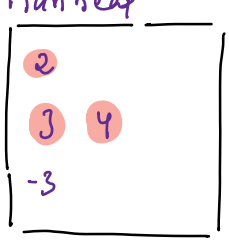
: no action

10:10 → 10:20 pm

→ For k times → getMan() & deleteMan() from heap & store in arr[]

: TC: k * {1 + log k}

→ Print data in inc order: TC: O(k)



getMan : 4
getMan : 3
getMan : 2
getMan : -3

dec order
Store in list/stack/ given arr
print in rev order

deleteMan(): 4 is deleted
deleteMan(): 3 is deleted
deleteMan(): 2 is deleted
deleteMan(): -3 is deleted

Median: point can divide data into 2 halfs { 1st half = 2nd half / Odd elements / Odd elements }

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

Sort ar[]: { 2 4 **5** 6 9 }

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

Sort ar[]: { -2 -1 2 **3** 4 10 15 }

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

Sort ar[]: { -1 2 **3** **6** 9 10 }
(c_1 c_2)

$$\text{median} = \frac{c_1 + c_2}{2} = \frac{3+6}{2} = 9/2 = 4$$

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

Sort ar[]: { -3 2 4 **6** **10** 12 20 24 }
(c_1 c_2)

$$\text{median} = \frac{c_1 + c_2}{2} = \frac{6+10}{2} = 8$$

28) Given an array print median of all prefix subarrays:

{Subarray start at index = 0}

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

Subarray : median

[0 0] : {9} → 9

[0 1] : {9 6} → (9+6)/2 = 7
↳ {6 9}

[0 2] : {9 6 3} → 6
↳ {3 6 9}

[0 3] : {9 6 3 10} → (6+9)/2 = 7
↳ {3 6 9 10}

[0 4] : {9 6 3 10 4} → 6
↳ {3 4 6 9 10}

Idea1: For every prefix subarray
: sort it & get median
merge sort
↳ TC: N * N log N = O(N² log N)

Idea2: Sorting prefix subarray
using InsertionStep ⇒ TC → O(N)
Insert ele in sorted data
& make overall data sorted

```
ar[5] = { 9 | 6 | 3 | 10 | 4 }
          9 | 6
          6   9 | 3
          3   6   9 | 10
          3   6   9   10 | 4
          3   4   6   9   10
```

↳ TC: N * N ⇒ O(N²) SC: O(1)

optimization:

ar[9] ⇒ { 3 1 6 10 14 2 17 12 9 }

(0 1 2 3 4 5 6 7 8)

sort

ar[9] = { 1 2 3 6 | 9 | 10 12 14 17 }

1st half | 2nd half

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

man(a) < 9 < min(b)

Obs: 1. man ele in 1st half ≤ min ele in 2nd half

2. Take median in 1st half

3. 1st half size − 2nd half size = 1

→ Odd ele: median = man ele in 1st half

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

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

(0 1 2 3 4 | 5 6 7 8 9)

1st half | c_1 | c_2 | 2nd half

Obs: 1. man ele in 1st half ≤ min ele in 2nd half

2. c_1 → 1st half c_2 → 2nd half

3. 1st half size − 2nd half size = 0

→ even: c_1 → man of 1st half c_2 → min of 2nd half = $\frac{c_1 + c_2}{2}$

Combine both:

1. man ele in 1st half ≤ min ele in 2nd half
2. 1st half size − 2nd half size ≤ 1
3. Odd ele: median ⇒ man of 1st half

   even ele: median ⇒ avg (man of 1st half + min of 2nd half)

	0	1	2	3	4	5	6	7	8	9
ar[10] : {	4	6	9	2	1	10	14	7	3	5 }

size 1st - size 2nd <= 1

man of 1st box <= min of 2nd box

1st half : box1

2	4	1
	3	5

manheap

getman()
delete_man()
insert()
size()

getmin()
delete_min()
insert()
size()

2nd half : box2

9	7
10	14
6	

minheap

ele : Insert	box1 size box2	Transfer	median
4 : insert in 1st half	1 0 = 1	no transfer	4
6 > man of box1 : insert box2	1 1 = 0	no transfer	$\frac{4+6}{2} = 5$
9 > man of box1 : insert box2	1 - 2 = -1 2 - 1 = 1	min 2nd → 1st box	6
2 < man of box1 : insert box1	3 - 1 = 2	man 1st → 2nd box	$\frac{4+6}{2} = 5$
1 < man of box1 : insert box1	3 - 2 = 1	no transfer	4
10 > man of box1 : insert box2	3 - 3 = 0	no transfer	$\frac{4+6}{2} = 5$
14 > man of box1 : insert box2	3 - 4 = -1 4 - 3 = 1	min 2nd → 1st box	6
7 > man of box1 : insert box2	4 - 4 = 0	no transfer	$\frac{6+7}{2} = 6$
3 < man of box1 : insert box1	5 - 4 = 1	no transfer	6
5 < man of box1 : insert box1	6 - 4 = 2	man 1st → 2nd box	$\frac{5+6}{2} = 5$

```
void running_median (int ar[]){
    min heap<int> minh
    max heap<int> maxh
    maxh.insert(ar[0])   } 1st ele
    print(ar[0])
    for(i=1; i<N; i++){
        ele = ar[i]
        if (ele < maxh.getMax()){ // Insert in left
            maxh.insert(ele)
        }
        else{
            minh.insert(ele)
        }
        if (maxh.size() - minh.size() > 1){
            // Extra ele present in maxheap
              Transfer max from maxheap → min heap
        }
        if (maxh.size() - minh.size() < 0){
            // Extra ele present in minheap
            Transfer min from minheap → maxheap
        }
        int s = minh.size() + maxh.size()
        if (s%2 == 1){ print(maxh.getMax())}
        else{
            print( [maxh.getMax() + minh.getMin()]/2)
        }
    }
}
```

TC: $N * \{\log n + \log n + \log n\}$

$\Rightarrow O(N \log N)$

SC: $O(N)$

```
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 = maxh.size() + minh.size()
        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