→ Today's Content:

- → k smallest elements
- → Running Median
- → k-apart from Sorted Array

→ complete Binary Tree

Heaps → CBT + (Element >= children) : Max Heap
CBT + (Element <= children) : Min Heap

Min Heap Max Heap TC

:getMin() :getMax() → O(1)
:insert() :insert() → O(log N) (N → number of elements)
:deleteMin() :deleteMax() → O(log N)

:search()
:delete()
} → O(N), Instead use BBST → (log N/2)

Heap Libraries in your language of choice.

MinHeap<data> h1; → h1.min()
h1.insert()
h1.delete() : It can delete min

MaxHeap<data> h2; → h2.max
h2.insert
h2.delete() : It can delete max

→ k smallest elements

IQ) Given N distinct array elements find k smallest elements

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

k=4 :

↳

output : [1 2 3 4]

Solutions :

→ (Pdeas): { Take 1st min & keep at a[0]
Take 2nd min & keep at a[1]
Take 3rd min & keep at a[2]
Repeat k times

TC: O(N*k)

SC: O(1)

Pdeas:

→ (sort the entire array & get first k elements)
[0 k-1]

TC: O(N log N + k)

SC: O(N)

Pdeas: → Insert every thing min Heap + k times { get min, delete min }

{ TC: N log N + k x {log N}
SC: O(N)

Idea: → Insert every thing max heap + →

→ (N-k) {delete max} → (Remove N-k max)

After above k elements in maxheap

→ k times { get max() / delete max } heap size = k

$k = N/2$

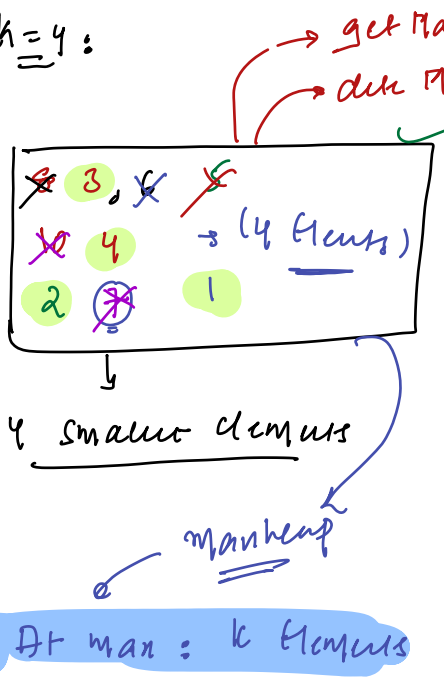
TC: $O(N \log N + (N-k)\log\frac{N}{2} + k \log\frac{k}{2})$

$\Downarrow$

TC: $O(N \log N)$

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

k = 4 :



At max : k Elements

10 < 11 : 11 Cannot ans

10 > 2 : 2 can be ans : delete 10

8 > 7 : 7 can be ans : delete 8

7 > 6 : 6 can be ans : delete 7

6 > 5 : 5 can be ans : delete 6

5 > 1 : 1 can be ans : delete 5

// Pseudo code → { Heap Size → k Elements }

```
ManHeap<Int> mh;

for(i=0; i<k; i++){                    } → k log(k)
    mh.insert(ar[i])                   }
}

for(i=k; i<N; i++){                                    (N-k)
    // ar[i] belongs in heap or not?                   }
    Int e = mh.getMax();                               } → (N-k) log k
    if(ar[i] < e){                                     }
        mh.insert(ar[i])   } → both are performed in   }
        mh.delete()        } heap of size k            }
    }
}
```

O(1)

log k

// In your manheap, we have k smaller elements

TC: $k \log k + (N-k) \log k = k \log k + N \log k - k \log k$

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

Better than >= $N \log N$ → Sorting

→ If $k = N/2$

→ $O(N \log N)$ SC: $O(N)$

// given N array elements find 3 max elements → { ___ }

→ You can do it above approach

→ min heap → size = 3

208) Given N distinct array elements, every element is atmax k index far from its actual sorted position sort given array.

Ex: ar[10] = { 3 4 7 9 14 21 27 30 39 44 }

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

given Input

k = 4

ar[]: { 4 9 27 3 39 44 7 14 30 21 }

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

→ sort the array

Every element is at max k distance from its actual sorted position

1) Approach →

Tc: O(NlogN)

2)

	0	1	2	3	4	5	6	7	8	9
ar[] : {	4	9	27	3	39	44	7	14	30	21 }

0	1	2	3	4	5	6	7	8	9

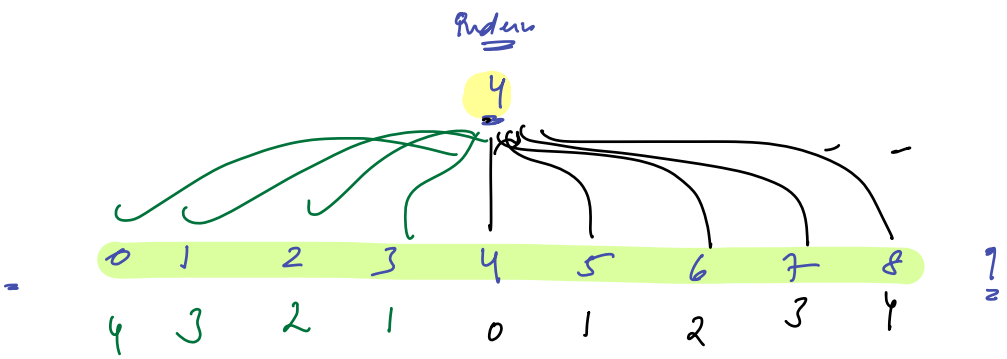
k=4 : 3 4 7 9 14 21 27 30 39 44

two pointers [0 - 8] + 9th → [1 - 8], 9th → [1-9] → [2 - 9] → [3 - 9]

as already removed on heap

~~4~~ ~~9~~ ~~27~~ ~~7~~ ~~30~~

~~3~~ 39 44 ~~14~~ ~~21~~



Pseudocode :

At most in heap = (k+1)

MinHeap<int> hm; →

{ hm.insert [0 k] in hm;
  print (hm.getMin())
  hm.deleteMin()

for (i = k+1; i < N; i++){
  hm.insert (arr[i])
  print (hm.getMin())
  hm.deleteMin()
}

while(hm.size() > 0)
  print (hm.getMin())
  hm.deleteMin()

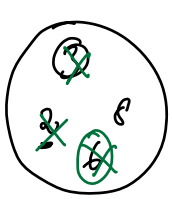
N * log k
↳ is better NlogN

SC: O(k)

Interview Questions

Smart ma

IIPM



2 → 3 → 6 → 8

Eg: median: centre element

ar[3] = { 1 4 2 }

→ 2 ⇒ median

Inc order: { 1 2 4 }

ar[5] = { 3 2 5 7 4 }

median

Inc odr = { 2 3 4 5 7 }

↓
/ → Integer

ar[6] = { 3 2 8 4 10 1 }

Inc odr = { 1 2 { 3 4 } 8 10 }

↳ (3+4) = 7/2 = 3

ar[4] = { 3 { 5 9 } 13 }

↳ 5+9 = 7

7 →

// given N array elements, calculate median for a { subarrays starting at index 0 }

Ex: ar[5] = { 9 6 3 10 4 }
(0 1 2 3 4)

{ # of N subarrays will start from index = 0 }

Subarrays | Median

[0-0] = 9 — 9

[0-1] = {9 6} — 7

[0-2] = {9 6 3} — 6
→ {3 6 9}

[0-3] = {9 6 3 10} — 7
→ {3 6 9 10}

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

Idea
- → for every subarray get median.
  - TC: N * { N log N + O(1) }
    - N log N → sorting
    - O(1) → getting
  - TC: $N^2 \log N$

0 1 2 3 4

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

// Insertion sort

[0-0] = 9

[0-1] = { 9 6 } = { 6 9 }

[0-2] = { 6 9 3 } = { 3 6 9 }

[0-3] = { 3 6 9 10 } → { 3 6 9 10 }

[0-4] = { 3 6 9 10 4 } → { 3 4 6 9 10 }

→ After every pass in insertion sort get median

TC: O(N²)

SC: O(1)

Because we use same array to sort

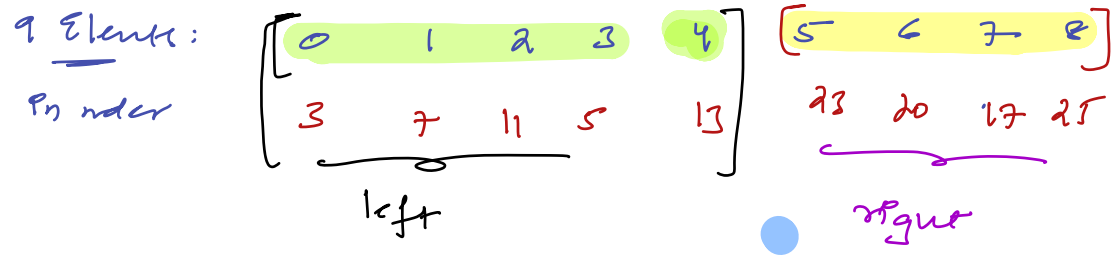
Thursday = { heaps + greedy }

Saturday = { greedy }

Greedy = { — }

Trees

observation: → median:

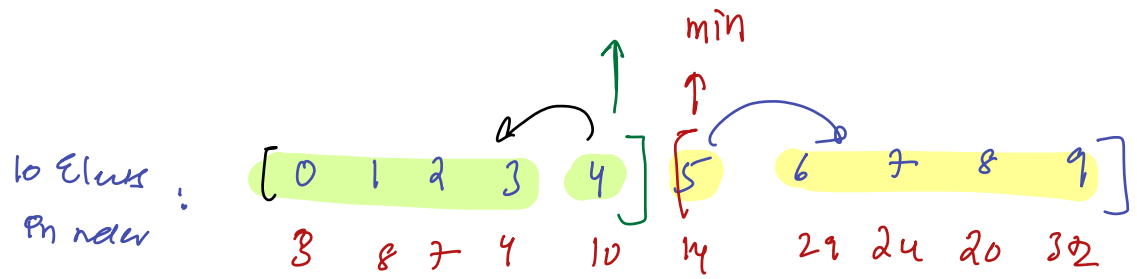


// left < right

size(left) - size(right) = 1

odd elements: median in left

Median = max in left

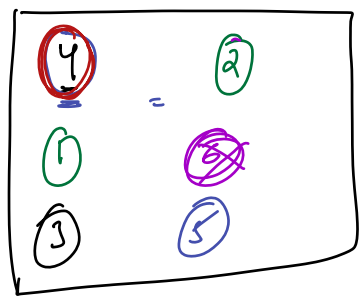


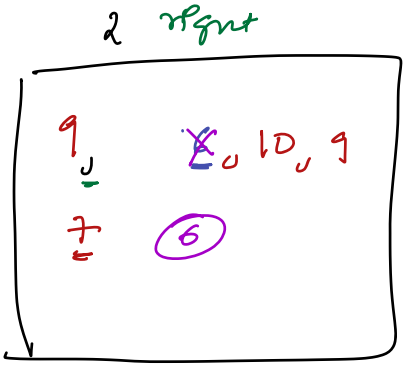
// left < right

size(left) - size(right) = 0

Median = $\frac{\max \text{ in left} + \min \text{ in right}}{2}$

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





→ pusch()
→ getMax()
→ deleteMax()
→ size()

} ⇒ Maxheap

⇒ push()
⇒ deleteMin()

} ⇒ minheap

⇒ size()

1) // left & right ✓ : ✓

2) // size(left) - size(right) = {0, 1}

After 4, 9 : (4+9)/2 ⇒ 6,

After → 6

After → 5

After ⇒ 4

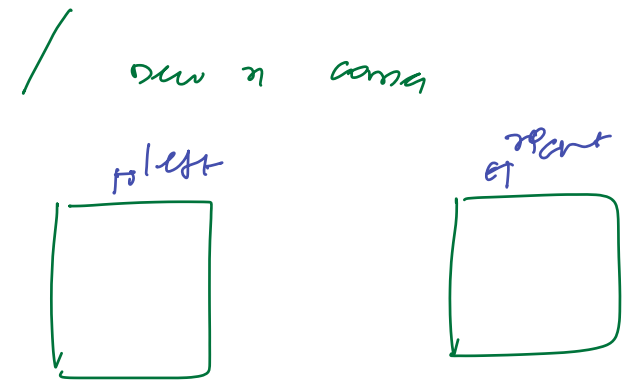
After ⇒ 5

```
Maxheap left ; Minheap right .
left.insert(ar[0])
print(ar[0])
for(i = 1; i < N; i++){
    if( ar[i] <= left.max){
        // ar[i] belongs to left side
        left.insert(ar[i])
    }
    else { right.insert(ar[i])
    }
    if( left.size() < right.size()){
        // Transfer from R-L
        Left.insert( right.getmin())
        right.delete()
    }
    if( left.size() - right.size() > 1){
        // Transfer L-R
        right.insert( left.getmax())
        left.delete()
    }
    if( left.size() == right.size()){
        print( left.getmax() + right.getMin()/2)
    }
    else { print (left.getmax())
    }
}
```

// left < = right

TC: $N * \{ 3 \log N \} \rightarrow O(N \log N)$

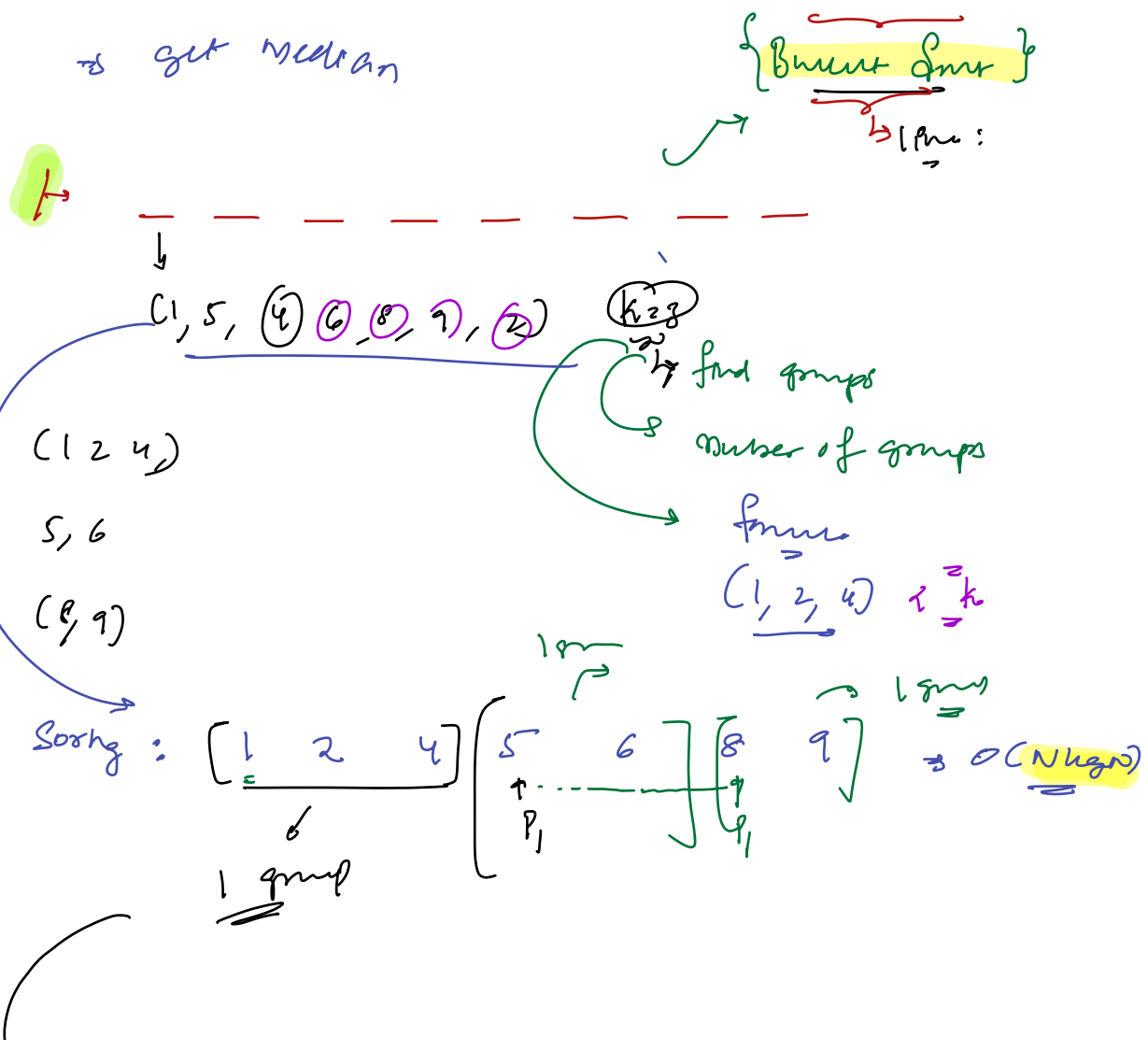
SC: $O(N)$



→ decide whether a pt in left or right

→ Balance left & right

→ get median



Heap: Sort:
- O(N log N)
- SC: O(1)

1:

k bucket — Each bucket k

[min + 0   min + k-1]

[min + k   min + 2k-1]

[   ]

[   ]

[max]

		min	max
{ [1	4] :	1	4
[5	8] :	5	8
[9	13] :	9	9
}			

k = 3

// {min ——— } : TC: $\frac{(max - min)}{k}$

{ ——— max}

Iterate on every element & in that bucket, min & max

mn ma

4 5 7 10

{ [1 4] : 4 4
[5 8] : 5 7 } 2! = Combin

↳ Ada adjar Dua

[9 13] : 10 10

}

// Constran

$1 \le 10^6$

$arr[i] \le 10^8$

$k \le 10^6$