PSP 55.59 → 60

Nov23_PSP_15Apr
sudhakar venkatachalam
Mohammad Mateen
Vijay V A
Piyush Kumar
kameswarreddy Yeddula
Kevin Theodore E
Suraj Devraye
Sai Sharath
Yash Malviya
Rajeev
Manjunatha I
Harshil Dabhoya
manikandan m
Gobika K
Mayur Hadawale

Nov23_PSP_15Apr
Robin Dhiman
Vigneshwaran K
Sarat Patel
MD JASHIMUDDIN
Nitendra Rajput
Pushkar Deshpande
Prashant Kumar Soni
Pradeep Kumar Chandra
Shaurya Srivastava
SIJU SAMSON
ALLEN GEOSHAN M
Pranadarth S
Mohammed Arshad
Tushar Desarda
Rsr Ram

Agenda

Heap Sort

K^{th} largest element

Sort nearly sorted array

Median of stream of integers

Q → Sort an Array

We want to sort an array in increasing order using a heap

Idea 1: Create a min heap and continuously extract min for n times

1. Build a heap
2. Extract min and store ans in array

Quiz 1

What is the time complexity to convert an array to a heap ?

O(N)

Quiz 2

Root element in heap is _____ ?

depends on either min or max Heap

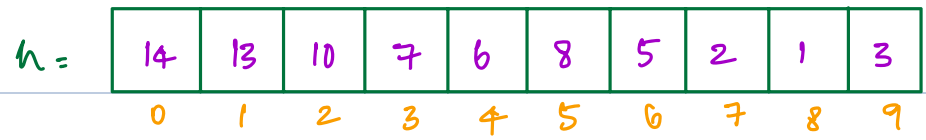
Sorting an array using heap

1. Build a Heap → O(n)
2. Extract Min for n times → O(nlogn)

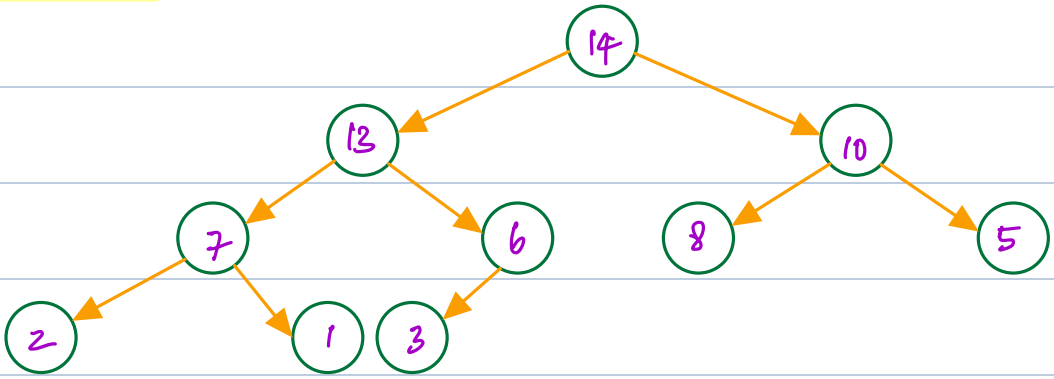
T.C = O(n log n) S.C = O(n)

Can we optimize Space complexity ?

Hint: Use a max Heap

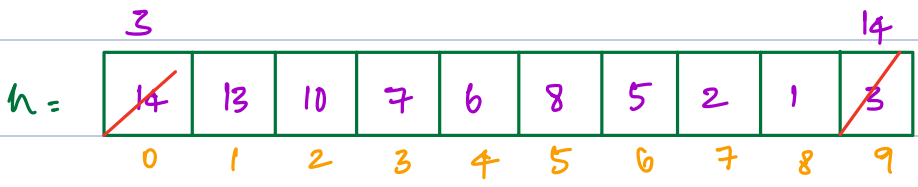


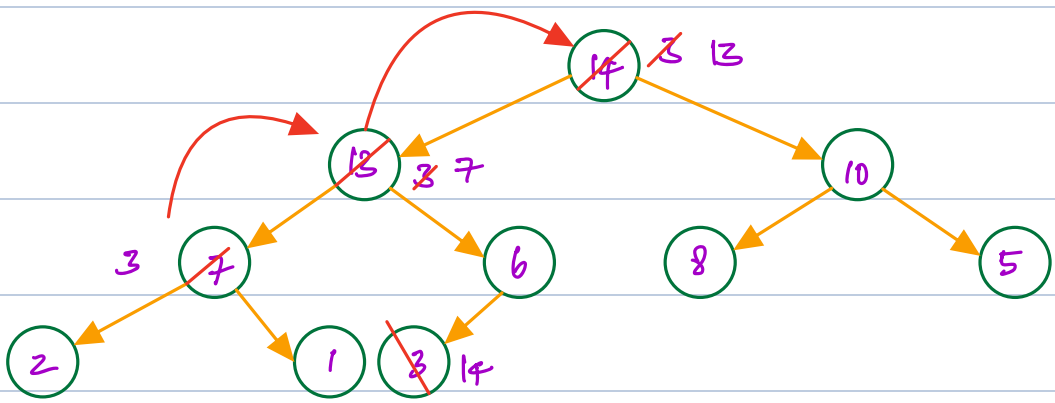
Visualize

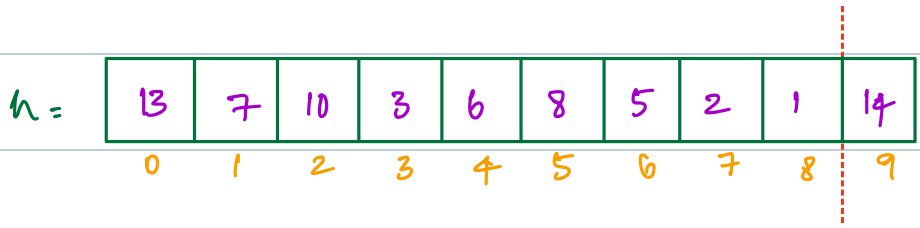


we can extract the max element from Heap

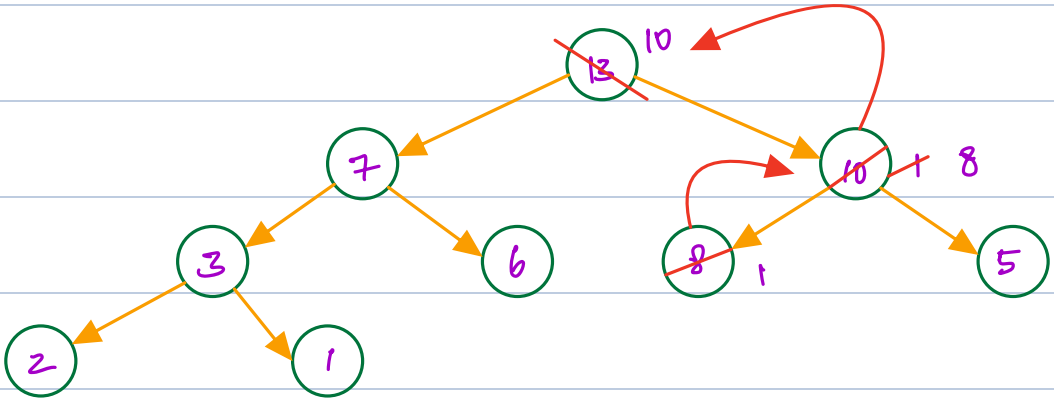
= 14

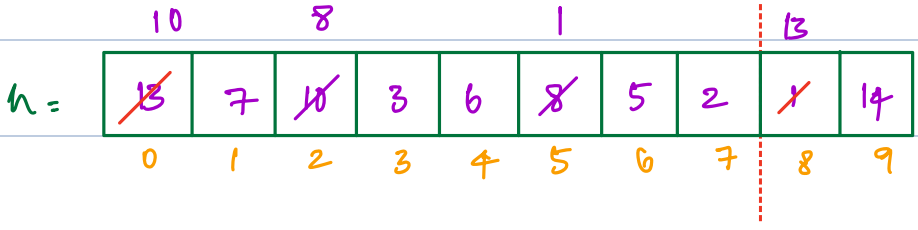






Reduce the size of array by 1, subarray [0,8]





extract max & heapify (root)

Repeat the following steps N times

① extract max & heapify 0th index

② Reducing size of heap and performing above step N time

psendo code

```
// Build max heap from array
j = n-1;
while (j>=0)
    swap (A[j], A[0]);
```

```
    for(int i = n/2 - 1; i >= 0; i--)
        heapify(i, heap, n);
}
```

T.C = O(n log n)

S.C = O(1)

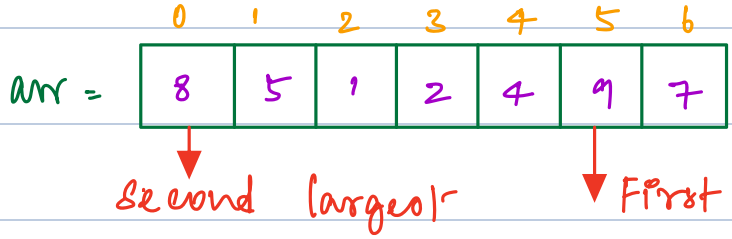
In place sorting → no additional space, Heap sort is in place

Stable sorting → are you maintaining order of input

Heap Sort is not stable

Q → Kth Largest Element

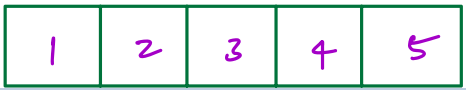
Given arr[N], find the Kth largest element



ans = 7

Quiz 3

What is the 5th largest element of an array



ans = 1

Idea 1 : Sort the array in decreasing order

return arr[k-1]

T.C = O(n log n)

Idea 2: Using heap sort k steps (max)

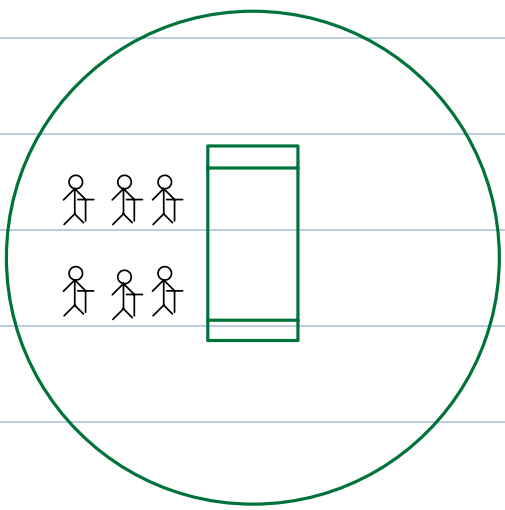
a) Build a max heap ⟶ O(n)

2) For k times extract max ⟶ O(k log n)

T.C = O(k log n)

Idea 3 : using min heap

example: Imagine you are in a cricket selection and you have to pick 2 batsmen out of 6 batsmen



12	8	9	4	6	11

when you see third batsman score more than lowest score so far, replace him with high score

when you see a person score less than existing selected least score, you will ignore them

The min element in our selection represents 2nd largest high scorer from the pack

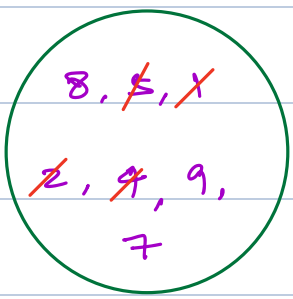
Example

arr :

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

k = 3

Create a min heap of size k



Min Heap

a[i]	h[0]	a[i] > h[0]	Remark
2	1	2 > 1	Replace 1
4	2	4 > 2	Replace 2
9	4	9 > 4	Replace 4
7	5	7 > 5	Replace 5

Third greatest is root of heap → 7

pseudo code

```
int getKthLargestElement(arr[], k) {
    heap[]
    for(i=0; i<k; i++) {
        heap.append(arr[i]);
    }
    heapify(heap); // min heap
    for(i=k; i<n; i++) {
        if(arr[i] > heap[0]) {
            heap.extract_min();
            heap.insert(arr[i]);
        }
    }
    return heap[0];
}
```

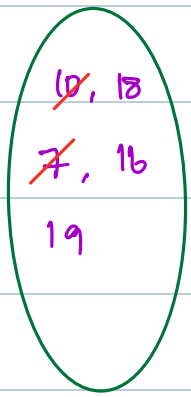
T.C = O(n-k log k) S.C = O(k)

Q → Find the kth largest element for all windows of an array starting from 0th index

	0	1	2	3	4	5	6
arr =	10	18	7	5	16	19	3

K = 3

s	e	array elements	ans array
0	2	10, 18, 7	7
0	3	10, 18, 7, 5	7, 7
0	4	10, 18, 7, 5, 16	7, 7, 10
0	5	10, 18, 7, 5, 16, 19	7, 7, 10, 16
0	6	10, 18, 7, 5, 16, 19, 3	7, 7, 10, 16, 16



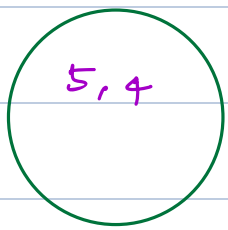
Min Heap

Ques 4

Find the kth largest element for all windows of an array starting from 0 index

	0	1	2	3	4
arr =	5	4	1	6	7

K = 2



Min Heap

a) create a Min heap of size k

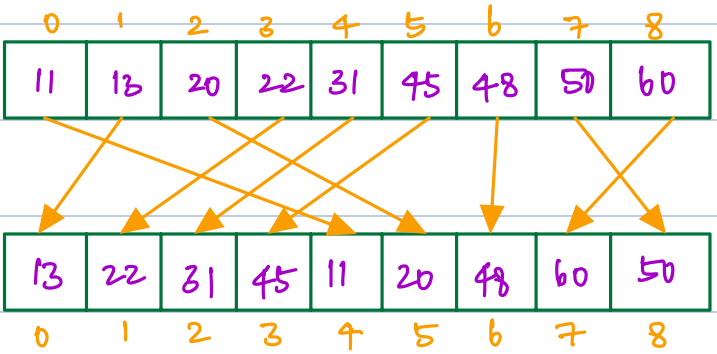
b) for all i, check if arr[i] > heap[0]
extract min & insert arr[i] to heap

c) root will give the ans

Q → Given a nearly sorted array. You need to sort the array

Nearly sorted array → Every element is shifted away from its correct position by at most k steps

Example



k = 4

Idea 1 Sort the array T.C = O(nlogn)

S.C = O(n)

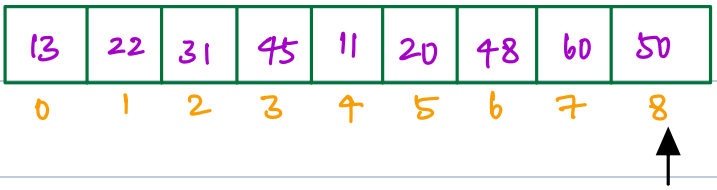
Idea 2

Since each element can be k steps aways the smallest element will be in range [0, k+1]

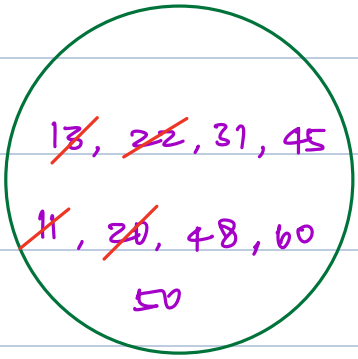
MIN HEAP of size k+1

to store all elements in window

DRY RUN



k = 4



① Add all elements from [0, k] to my heap

② Iterate from k+1 to N a) extract min

b) add arr[i] to heap

③ Till heap is not empty, keep extracting min

pseudo code

```
heap : []
for (i=0; i<=k; i++) {
    heap.append(arr[i]);
}

heapify(heap) // min heap        O(k)
for (i= k+1; i<n; i++) {
    ans.append ( heap.extractmin() );
    heap.insert (arr[i]);
}

while (! heap.empty) {
    ans.append (heap.extractmin());
}
```

T.C = O(n log k)

Q → Given an infinite stream of integers. Find the median of the curr set of elements.

Median is the middle element in sorted array

The median of

1	2	5	4	3	6

① sort the array

1	2	3	4	5	6

middle ele

② median $\frac{3+4}{2} = 3.5$

Quiz 5

Median of

1	2	4	3

median $= \frac{2+3}{2} = 2.5$

Brute Force

I/P → 9 8 4 6 7 12 15 ...

Median → 9 8.5 8 7

For every input add it to end of array and sort it.

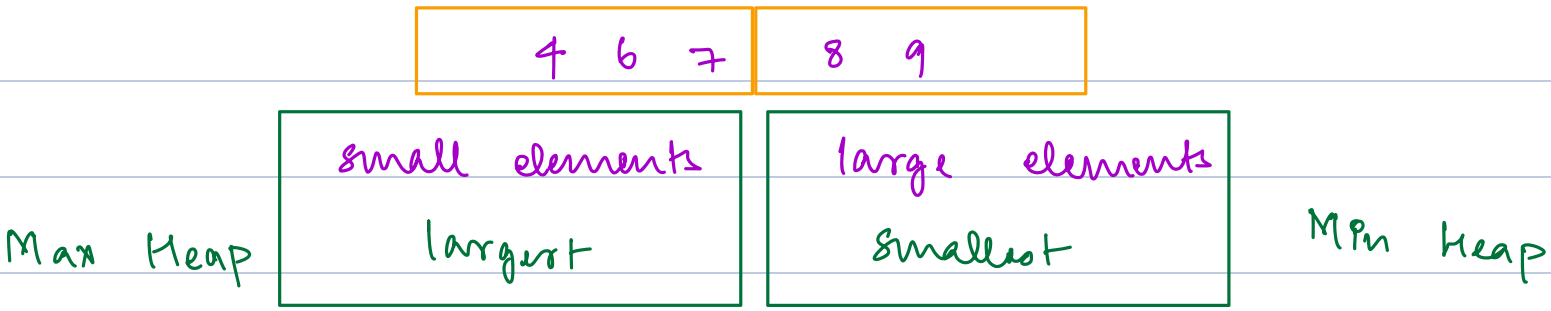
T.C = $O(n^2 \log n)$

Idea 2 — for every input insert it in correct position and calculate median

T.C = $O(n^2)$

Idea 3

I/P → 9 8 4 6 7 12 15 ...



(odd)

Ans → try to store my ans in root of Max heap (or)

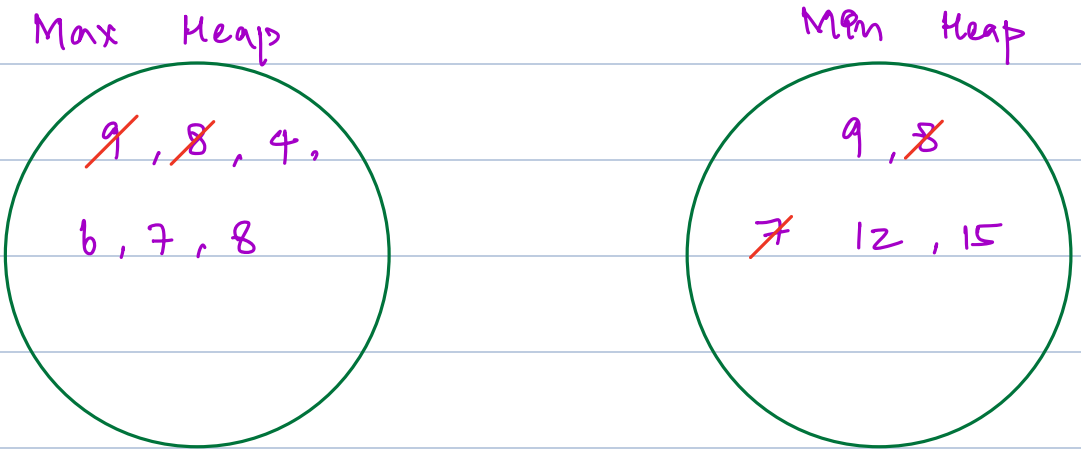
$$\frac{\text{root of max heap} + \text{root of min heap}}{2}$$

(even)

size of max Heap − size of min Heap = {0, 1}

I/P → 9 8 4 6 7 12 15 ...

9 8.5 8 7 7 7.5 8



∀ Input x

If (x <= root of max Heap)

→ Insert x in max Heap

else → Insert x in min heap

ds = max heap size - min heap size

If (ds > 1) remove root from max heap
& Insert in min heap

If (ds < 0) remove root from min heap
& Insert in max heap

length (odd) → root of max heap
ans

length (even) → $\frac{\text{root (max)} + \text{root (min)}}{2}$