Lecture : Heaps 2

Agenda

- Heap sort
- K places apart
- Kth largest element in every prefix
- Running median.

Qu 1 Given arr[n] and k.

Every element is ==at max k== distance away from its sorted posⁿ, we have to sort the array.

Note: k is very small wrt n.

Eg:

$$\begin{array}{ccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ A = [& 3 & 1 & 5 & 6 & 2 & 7 & 4 &] \end{array}, \quad K = 3$$

output: $[\ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7\ ]$

Idea1: Arrays.sort(arr);

TC: $O(n \log n)$

Idea 2: Min heap

$A = [\overset{0}{3} \quad \overset{1}{1} \quad \overset{2}{5} \quad \overset{3}{6} \quad \overset{4}{2} \quad \overset{5}{7} \quad \overset{6}{4}]$ $\quad K = 3$

output: [1 2 3 4 5 6 7]

Sorted array	ip array
0th	(0 - 3) idx
1st	0 - 4 idx
2nd	0 - 5 idx
3rd	0 - 6 idx
4th	1 - 6 idx
5th	2 - 6 idx

~~3~~
~~1~~
~~5~~
~~6~~
~~2~~
~~7~~
~~4~~

Min heap

output'

1 2 3 4

5 6 7 Ans

Code

```
void sort( arr[] , k) {

    PriorityQueue<Integer> minHeap = new PriorityQueue<>();

    // Insert (0-k) idx in min heap

O(K) —— for(i=0; i<=k; i++) {
            minHeap.add( arr[i]);
        }

        idx = 0;

     —— for(i=k+1; i<n; i++) {
            arr[idx] = minHeap.poll();
            idx++;
            minHeap.add( arr[i]);
        }

        while( ! minHeap.isEmpty()) {
            arr[idx] = minHeap.poll();
            idx++;
        }
}
```

k el — add: log k

poll() — log k

min Heap

TC: O(n log k)

SC: O(k)

Qu2 Given arr[n], find $\boxed{\text{kth largest el}}$ from 0th − ith idx.

$\forall \; (i >= K-1)$ [Important]

	0	1	2	3	4	5	6	7
A[] =	10	18	7	5	16	19	3	17

, K = 3

7

7

10

16

16

17

Idea1: for every set of elements, store K largest elements and return smallest among them.

Logic

$$A[\,] = \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \end{array}$$
$$\left[\begin{array}{cccccccc} 10 & 18 & 7 & 5 & 16 & 19 & 3 & 17 \end{array}\right], \quad K = 3$$

7

7

10

16

16

17

~~10~~
18
~~7~~
~~16~~
19
17

output:

Red	Green	Yellow	Block
7	7	10	16

heap.peek()

Blue	Red
16	17

Code

```
void kLargest(arr[], k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();

    for(i=0; i<k; i++) {
        pq.add(arr[i]);
    }
    print(pq.peek());

    for(i=k; i<n; i++) {
        if(arr[i] > pq.peek()) {
            pq.poll();
            pq.add(arr[i]);
        }
        print(pq.peek());
}
```

TC: n log k

SC: O(k)

Qu: **Heap sort**

Given arr[n], sort array using heap.

Idea1:

arr[] — input

↓

Build a min heap — $O(n)$

↓

el = getMin() and deleteMin() — $O(n\log n)$

↓

add el in your ans[]

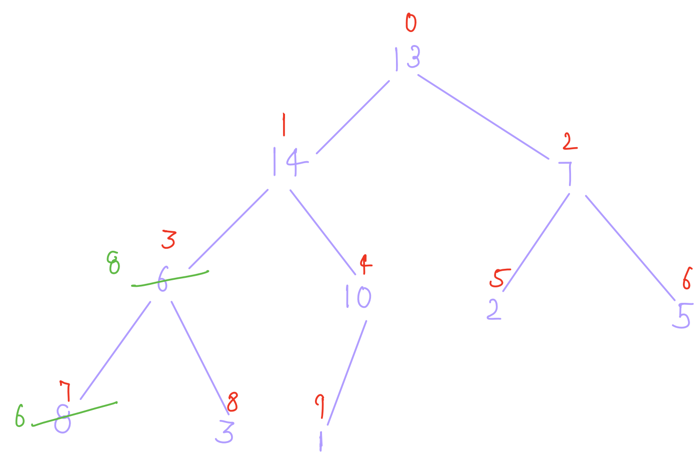
TC: $O(n\log n)$
SC: $O(n)$

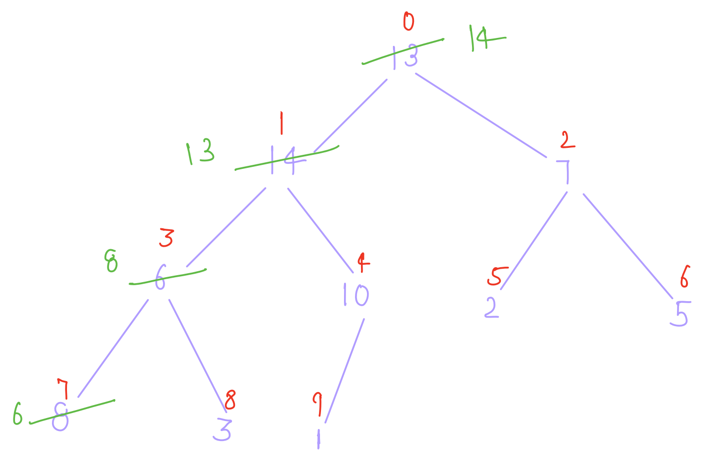
Qu: Can we optimise ~~this~~ space?

Idea2 Max heap

$arr[\,] = [\overset{0}{\cancel{13}}\ 14,\ \overset{1}{\cancel{14}}\ 13,\ \overset{2}{7},\ \overset{3}{\cancel{6}}\ 8,\ \overset{4}{10},\ \overset{5}{2},\ \overset{6}{5},\ \overset{7}{\cancel{8}}\ 6,\ \overset{8}{3},\ \overset{9}{1}]$

Visualisation





Dry run

4th idx: A[4] = 10

lc = 2*4+1 = 9 , A[9] = 1

rc = 2*4+2 = 10, out of bound

do not swap

3rd idx: A[3] = 6

lc = 2*3+1 = 7 , A[7] = 8

rc = 2*3+2 = 8 A[8] = 3

swap(3, 7)

2nd idx: A[2] = 7

lc = 2*2+1 = 5 , A[5] = 2

rc = 2*2+2 = 6 A[6] = 5

do not swap

1st idx: A[1] = 14

lc = 2*1+1 = 3 , A[3] = 8

rc = 2*1+2 = 4 A[4] = 10

do not swap

0th idx A[0] = 13

lc = 2*0+1 = 1 , A[1] = 14

rc = 2*0+2 = 2 A[2] = 7

swap(0, 1)

1st idx: A[1] = 13

lc = 2*1+1 = 3 A[3] = 8

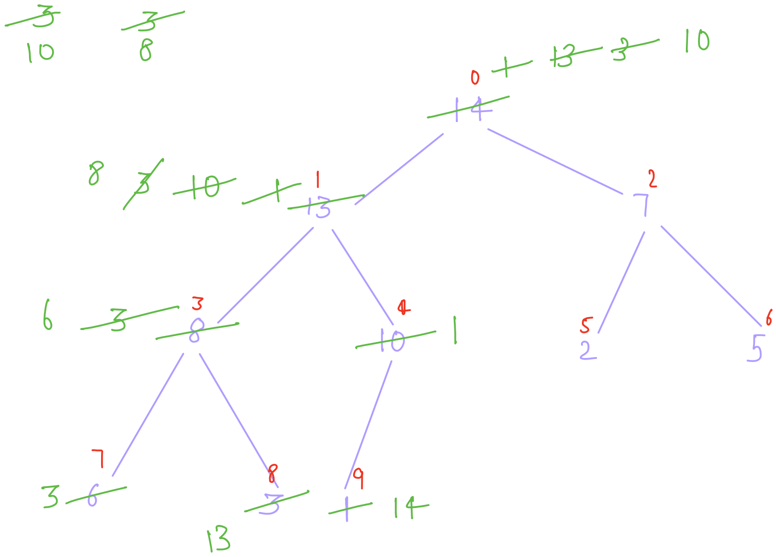
rc = 2*1+2 = 4 A[4] = 10

do not swap

Logic

A[] =

0	1	2	3	4	5	6	7	8	9
~~14~~ ~~1~~ ~~13~~ ~~3~~ 10	~~13~~ ~~1~~ ~~10~~ ~~3~~ 8	7	~~8~~ ~~3~~ 6	~~10~~ 1	2	5	~~6~~ 3	~~3~~ 13	~~1~~ 14



Dry run:

swap(0, 9)

14 is at its correct position

downheapify(arr, 0, 8)

0th idx: A[0] = 1

lc = 2*0+1=1 A[1] = 13

rc = 2*0+2=2 A[2] = 7

swap(0, 1)

1st idx: A[1] = 1

lc = 2*1+1=3 A[3]=8

rc = 2*1+2=4 A[4]=10

swap(1, 4)

4th idx:

lc = 2*4+1=9 → out of bound

swap(0, 8)

13 is at its correct position

downheapify(arr, 0, 7)

0th idx: A[0] = 3

lc = 2*0+1=1 A[1] = 10

rc = 2*0+2=2 A[2] = 7

swap(0, 1)

1st idx: A[1] = 3

lc = 2*1+1=3 A[3] = 8

rc = 2*1+2=4 A[4] = 1

swap(arr, 1, 3)

3rd idx: A[3] = 3

lc = 2*3+1=7 A[7] = 6

rc = 2*3+2=8 , out of bound

swap(3, 7)

⋮

To be contd..

Algo [H|W]

1. Build max heap

   inplace heap build [Discussed in prev class for min heap]

   challenge

2.

```
j = n-1
while( j>0) {
    swap(arr, 0, j);
    j--;
    downheapify(arr, 0, j); — Challenge.
}
```

Break: 8:35 - 8:45

Qu Given a running stream of integers, find median for all inputs. [Hard]

Median 1. A[] = [5 10 2 1 4]

↓ sort

1 2 $\boxed{4}$ 5 10 ⟶ median = 4

2. A[] = [5 10 2 3 1 4]

↓ sort

1 2 $\boxed{3 \quad 4}$ 5 10 ⟶ median = $\frac{3+4}{2} = 3.5$

ip A[] = [9 8 17 20 25 10 5 3]

9

8.5

9

{ 8 9 17 20 } ⇒ $\frac{17+9}{2} = 13$

{ 8 9 17 20 25 } ans = 17

{ 8 9 10 17 20 25 } ans = $\frac{10+17}{2} = 13.5$

10

9.5

Idea1: Insertion sort — H|W

TC: $O(n^2)$

SC: $O(1)$

Idea2 Case1: $A[\,] = [\, 5 \quad 7 \quad 4 \quad 3 \quad 6 \quad 2 \quad 1 \,]$

$$\downarrow \text{sort}$$

$$[\, \underbrace{1 \quad 2 \quad 3 \quad 4}_{\text{Part1}} \; < \; \underbrace{5 \quad 6 \quad 7}_{\text{Part2}} \,]$$

Median :- Max el of part 1.

Case2: $A[\,] = [\, \underbrace{1 \quad 2 \quad 3 \quad 4}_{\text{part1}} \quad \underbrace{5 \quad 6 \quad 7 \quad 8}_{\text{part}} \,]$

$$\text{Median :- } \frac{\text{Max(part1)} + \text{Min(part2)}}{2}$$

Observation

1. if no of elements are odd.

$$[\ \underbrace{1\quad 2\quad 3\quad 4}_{P1}\quad \underbrace{5\quad 6\quad 7}_{P2}\ ]$$

$Part1 = \frac{n+1}{2}$ el.

$part2 = \frac{n}{2}$ el.

Ans = max of part1. [achieve it using max heap]

2. if no. of elements are even.

$$A[\,] = [\ \underbrace{1\quad 2\quad 3\quad 4}_{part1}\quad \underbrace{5\quad 6\quad 7\quad 8}_{part}\ ]$$

$Part1 = \frac{n}{2}$ els.

$Part2 = \frac{n}{2}$ els

$$ans = \frac{\underset{\uparrow\ \text{max heap}}{\text{max of part1}} + \underset{\uparrow\ \text{min heap}}{\text{min of part2}}}{2}$$

3. size of max heap − size of min heap <= 1

Dry run:

A[] = {	9	8	17	20	25	10	5	3 }
	9	8.5	9	$\frac{9+17}{2}$ = 13	17	$\frac{10+17}{2}$ = 13.5	10	$\frac{9+10}{2}$ = 9.5

max heap:

- 3
- 5
- ~~10~~
- ~~17~~
- 9
- 8
- ~~9~~

min Heap:

- 10
- 17
- 25
- 20
- ~~17~~
- ~~9~~

1. P2 (min heap) > P1 (max heap)

2. s(max heap) − s(min heap) == 0 || 1

Code:

```
void  runningMedian( arr[]) {

    PriorityQueue<Integer> maxHeap = new PriorityQueue<>
                                        ( Collections.reverseOrder());

    PriorityQueue< Integer> minheap = new priority Queue<>();

    maxHeap.add(arr[0]);

    print (arr[0]);

    for(i=1; i<n; i++) {

        curr= arr[i];

        if( curr < maxHeap.peek()) {

            maxHeap. add( curr);

        } else{

            minHeap. add( curr);
        }

        // Balance — size(maxHeap) -size(minHeap) ==0 || 1

        if (maxHeap.size() - minHeap.size() > 1) {

            int el= maxHeap.poll();

            minHeap.add(el);
    }
```

```
if (maxHeap.size() - minHeap.size() < 0) {
        int el = minHeap.poll();
        maxHeap.add(el);
 }

 int size = maxHeap.size() + minHeap.size();

 if( size % 2 == 0) {
     print( (maxHeap.peek() + minHeap.peek()) / 2 );

 } else {
     print( maxHeap.peek());
 }
}
```

TC: $O(n \log n)$

SC: $O(n)$

Doubt

pfxor [a_0 $\quad$ $a_0 \wedge a_1$ $\quad$ $a_0 \wedge a' \wedge a_2$

$(0,0)$ i $\quad$ $(0,1)$ $\quad$ j

(1)

a_0 $\quad$ a_1 $\quad$ a_2 $\quad$ a_3 $\quad$ a_4

class Info{
int val;
int len;
int s;
int e;
}

0	1	2	3
a_0	a_1	a_2	a_3

pfxor

a_0	$a^0 \wedge a^1$	$a_0 \wedge a_1 \wedge a_2$	$a_0 \wedge a^1 \wedge a_2 \wedge a_3$

4 → $a_0 = a_0$ ✓
$a_1 = a_1$ ✓
$a_2 = a_2$ ✓
$a_3 = a_3$ ✓
0
→ $a_0 \wedge a_1$ ✓
→ $a_0 \wedge a_1 \wedge a_2$ ✓
→ $a_0 \wedge a_1 \wedge a_2 \wedge a_3$ ✓
$a_1 \wedge a_2$ ✓
$a_1 \wedge a_2 \wedge a_3$ ✓
$a_2 \wedge a_3$ ✓

Pair

$a_0 \wedge a^0 \wedge a^1 = a_1$ $\quad$ $(0,1)$

$\cancel{a^0} \wedge \cancel{a^0} \wedge a^1 \wedge a^2$ $\quad$ $(0,2)$

$\cancel{a^0} \wedge \cancel{a^0} \wedge a^1 \wedge a^2 \wedge a^3$ $\quad$ $(0,3)$

$\cancel{a^0} \wedge \cancel{a^0} \wedge \cancel{a^0} \wedge \cancel{a^1} \wedge a^2$ $\quad$ $(1,2)$

$\cancel{a^0} \wedge \cancel{a^1} \wedge \cancel{a^0} \wedge \cancel{a^1} \wedge a^2 \; a^3$ $\quad$ $(1,3)$

$\cancel{a_0} \wedge \cancel{a_1} \wedge a_2 \;\wedge\; \cancel{a_0} \wedge \cancel{a_1} \wedge \cancel{a_2} \wedge a_3$ $\quad$ $(2,3)$

0, 4 →
1, 4 →
2, 4 →
3, 4 →

xor pair

xorPair(
)