Heap Sort

arr [] $\Rightarrow$ 7 8 1 5 2 4

$\downarrow$

Build a Heap $\Rightarrow$ T: $O(n)$ S: $O(1)$

$\downarrow$

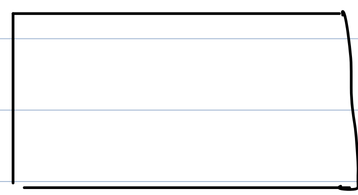
Extract min() & push into an new array

$\hookrightarrow$ T: $O(n \log n)$

S: $O(n)$

7 8 1 5 2 4

$\downarrow$



Min heap

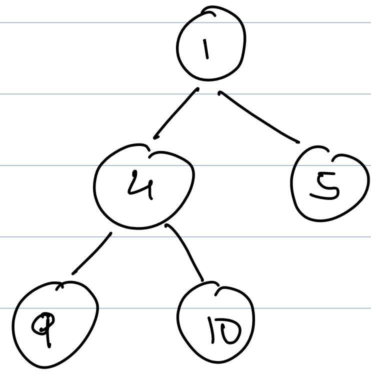
1 2 4 5 7 8

Heap Sort with a new array.

TC: $O(n + n\log n) \simeq n\log n$

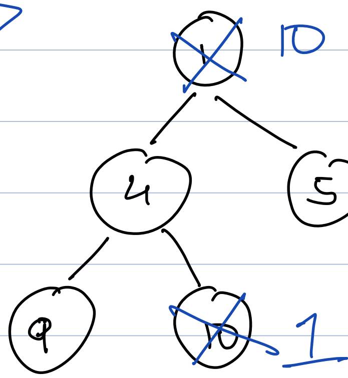
SC: $O(n)$

$\hookrightarrow$ if we use a new array



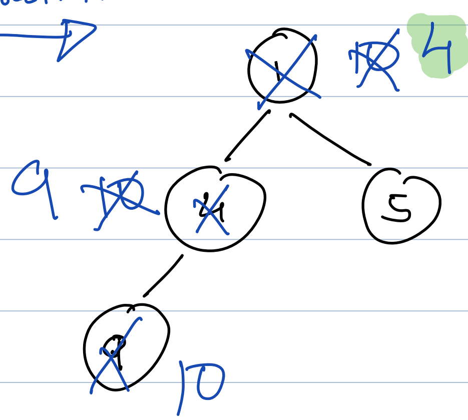
0	1	2	3	4
1	4	5	9	10

Extract min



0	1	2	3	4
~~1~~	4	5	9	~~10~~
10				1

Extract min



0	1	2	3	4
~~4~~	9	5	~~10~~	~~10~~
10			4	1

After extracting all minimums

10	9	5	4	1

reverse to get sorted array.

Approach 2: [Min Heap] Use no array. Reverse the array after extracting all minimum.

reverse.

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

$$SC: O(1)$$

Approach 2: [Max Heap] Use no array. Extract max at each step

$$TC: O(n + n\log n +) \simeq (n\log n)$$

$$SC: O(1)$$

Heap Sort

$\Rightarrow$ Inplace Sorting algorithm

$\Rightarrow$ Unstable Sorting algorithm

Q Find the k^{th} largest element in a array.

- → Build Max heap
- → Extract k element.
  The k^{th} extracted element is your answer.

$$TC: O(n + k\log n)$$

Q2 Find the k^{th} largest element in every window from $[0 - i]$. $[i \geq k-1)$

$K = 3$

	0	1	2	3	4	5	6
arr[] =	10	8	7	5	16	19	3

7

7

8

10

10

0	1	2	3	4	5	6
10	8	7	5	16	19	3

10, 8, 7 5

Max heap

$\simeq n \times k \log n$

→ 5 elements ⇒ 5th largest

↳ Smallest element

0	1	2	3	4	5	6
10	8	7	5	16	19	3

↓ insert k element

10, 8

7

$\xrightarrow{\text{min}}$ 7

5 (5 , root(minheap))

(5 < root (min-heap)
don't insert)

10, 8
7

$\xrightarrow{\text{min}}$ 7

$\downarrow$ 16

16 > root (min-heap)
→ Extract_min
→ insert 16

10, 8
16

$\xrightarrow{\text{min}}$ 8

$\downarrow$ 19

19 > root (min-heap)
→ Extract_min
→ insert 19

10, 19
16

$\xrightarrow{\text{min}}$ 10

Pseudo-code

→ Build heap with first k values $[0 \ldots (k-1)]$ $O(k)$

→ print (min (heap))

```
for (int i = k ; i < n ; i++) {
    if (arr[i] > root(min heap))
        → Extract-min
        → insert (arr[i]) in heap;

    print (root (heap))
}
```

$(n-k)\log k$

$$TC: (k + (n-k)\log k)$$

$\boxed{k = n/2}$ $TC = O(n\log n)$

K sorted array !

In an array every element is almost K distance apart from the sorted position. Sort the array.

$\boxed{K=3}$

	0	1	2	3	4	5	6
arr[] =	6	5	3	2	8	10	9
	2	3	5	6	8	9	10
Distance	3	1	1	3	0	1	1

0^{th} element $\Rightarrow$ $0 - k^{th}$ index.

$\xrightarrow{\text{min}}$ real 0^{th} element.

$\boxed{6 \quad 5 \quad 3}$ $\Rightarrow$ 2

1^{st} element $\Rightarrow$ $\boxed{6 \, 5 \, 3 \, 8}$ $\xrightarrow{\text{min}}$ real 1^{st} element

$\Rightarrow$ 3

$$TC: \; k + (n-k)\log k.$$

Pseudo Code

→ Build min heap { 0 ... k^th element }

```
for (int i = k+1 ; i<n ; i++) {
    int x = extract_min();
    arr.insert (x);
    insert_heap( arr[i]);
}

while ( !heap.empty() ) {

    arr.insert (extract_min ());
}
```

Q Running stream of input. Find the median of current set of element after a new element join.

Median ⇒ Middle element of a sorted array.

1) Odd Sized array.

2 3 4 5 6 7 8

1) Even Sized array.

2 3 4 5 6 7 8 9

$$\frac{5+6}{2} \Rightarrow \underline{\underline{5.5}}$$

9 8 7 6 5 4 - - - - - - - -

Median: 9 8.5 8 7.5 7 6.5

⇒ Insertion Sort TC: $O(n^2)$
SC: $O(1)$

9 8 7 6 5 4 . . . . - - - -

A
4 5, 6

Max heap

B
7, 8 9

Min heap

n element

1st half

2nd half

1

→ root (max-heap) $\geq$ curr. element

⇓

insert in 1st half

A 1,
4 5, 6

Max heap

B
7, 8 9

Min heap

n element

$$\text{Size}(A) - \text{Size}(B) \leq 1$$

2

A	B
2, 1, 4 5, 6	7, 8 9
Max heap	Min heap

[illegible]

Condition sacrificed. $Size(A) - Size(B) \leq 1$

because of max_heaps $\Rightarrow$ Remove the top from max-heap & insert into min heap.

A	B
2, 1, 4 5	7, 8 9, 6
Max heap	Min heap

[illegible]

median $\Rightarrow \frac{5+7}{2} = 6$

1n

10

1) root (max-heap) $\leq$ curr
insert - maxheap.
else insert - minheap

A	B
2, 1, 4 5	7, 8 9, 6 10
Max heap	Min heap

Condition sacrificed. because of min-heaps

$$\text{Size}(A) - \text{Size}(B) \leq 1$$

Size(B) can never be greater.

→ Remove the top from min-heap & insert into max-heap..

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

Pseudo Code

1) Insert 1st element inside max-heap.

↳ Print (root-max-heap);

2) For every new integer x.

```
if (x ≤ root-max-heap) {
        insert_max-heap(x);
        if ((size(A) - size(B)) > 1) {
            y = extract_max(max-heap);
            insert.min.heap(y);
        }
} else {
        insert_min-heap(x);
        if (Size B > Size A) {
            y = extract_min(min-heap);
            insert.max heap(y);
        }
}
```

}

if (size is odd)
print (root (max_heap));

else
print $\left(\frac{\text{root (max\_heap) + root (min\_heap)}}{2} \right)$;

$\boxed{\text{Priority\_Queue}}$