Topics: 1- k sorted array
2- Heap sort
3- k places apart
4- kth largest element
5- running median

each array is sorted

a {2, 3, 11, 15, 20}^{5}
b {1, 5, 7, 9}^{4}
c {0, 2, 4}^{3}
d {3, 4, 5, 6, 7, 8}^{6}
e {-2, 5, 10, 20}^{4}

k array
n elements
X = n x k

{-2, 0, ... , 20}^{22}

X log k

TC: O(K + (X-K) log k) (II)

min_heap

(k)

add remove log(k)

X total TC: O(X) + O(X log X) (I)

merge after each other

sort quick sort

2n + 3n + ... + kn

n (2 + 3 + ... + k)

$n \times \frac{k(k+1)}{2} \sim$

X log X

nk log k

n x k x k

-2, 2, 3, 9, 10, 11

log(n) + log(n-1) + ...

log(1)

create heap

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

SC: O(n)

min_heap

{3, 9, -2, 10, 11, 2}

Sort using heap in increasing order.

How?

sort increasing small to large ← use max heap

decreasing ← min heap

Heap Sort

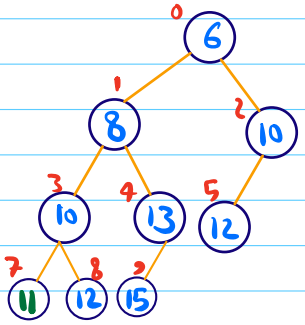
Quiz

* just for depiction

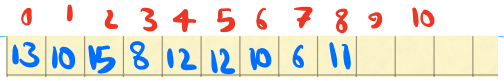
* actual heapify operation happens only on array SC: O(1)

step1

heapify



Convert by level order



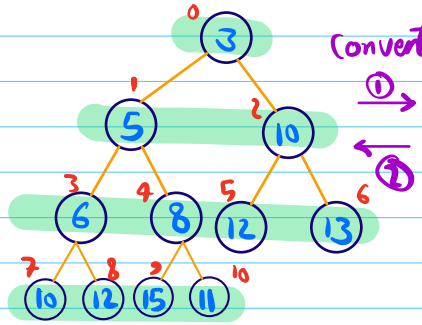
build heap SC: O(1)

TC: $O(n) + \log n + \log(n-1) + \log(n-2) \ldots$ (5, 3)

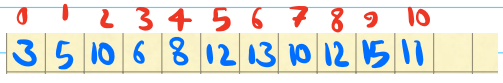
remove root from heap and readjust } log(n)

step2

Get & Remove min



Convert by level order



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

SC: O(1)

	TC	SC	stable?
Heap sort	nlogn	1	NO → Why?
merge sort	nlogn	n	Yes
quick sort	nlogn	O(1)	No

Comparison based sorts (i.e not count sort) cannot be better (average case) than $\Omega(n \log n)$

Microsoft

All windows starting from 0

P2 Given an integer array, $\forall i >= (k-1)$, find kth largest element from 0 to i.

bloom filter

A = { 10, 18, 7, 5, 16, 19, 3, 17 } k=3

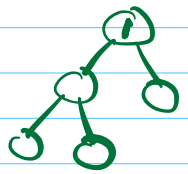
0 1 2 3 4 5 6 7

{ 7, 7, 10, 16, 16, 17 } ← ans

n-k+1

Quiz {1, 2, 3, 4, 5}

1st 5 2nd 4 3rd 3 4th 2 5th 1



{ 10 18 7 5 16 19 3 17 }

idea1 $3\log 3 + 4\log 4 + \dots + n\log n <= n^2 \log n$

idea2 binaray search after sort for finding kth largest (HW)

$O(n\log n + k\log n) \sim O(n\log n)$

idea3

optimized

10

min heap

19
18
16

size 3

build heap on first k item

ans.add(peaktop())

for i = k → n-1

if(A[i] <= root of min heap){ // ignore

else{

getMin() & remove

insert(A[i])

}

output is root of heap ← root of tree peaktop() O(1)

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

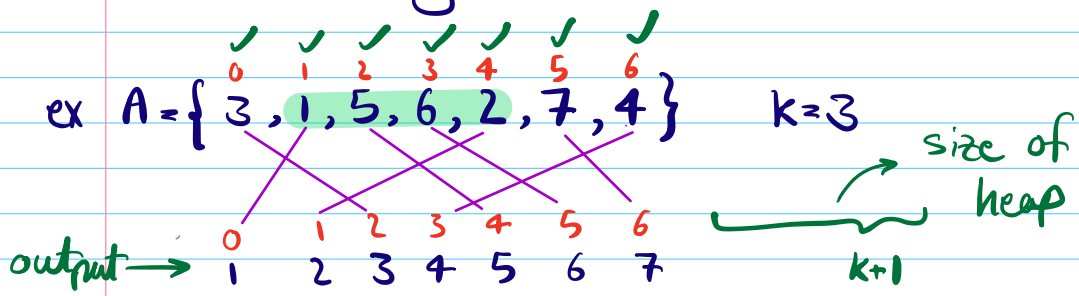
SC: $O(k)$

7

nearly sorted

P4 Given an integer array A with n elements & an integer k. In A, every element is at max k distance away from its position in sorted order. $|i-j| <= k$

Sort the array. $k <= 10^3$ $n <= 10^5$ $k << n$



idea pure sort

(I) TC: $n \log n$

ex A = { 1, 2, 5, 6, 3, 7, 4 } k=3

→ $O(k + (n-k)\log(k)) \sim O(n \log k)$ TC: $O(n \log k)$ II

(II) using heap

TC: $O(k)$

int

P5 Given a running stream of integers, find median of all elements, for each input. middle element in sorted order.

Quiz

ex {1, 2, 5, 4, 3} → Sort {1, 2, 3, 4, 5}

ex {5, 10, 2, 3, 1, 4} → sort {1, 2, 3, 4, 5, 10}

$\frac{3+4}{2} = 3.5$

ex 9, 8, 17, 20, 25, 10, 5, 3, ...

Sorted array

9	8	8	8	8	8	5	3
	9	9	9	9	9	8	5
	85	17	17	17	10	9	8
			20	20	17	10	9
				25	20	17	10
					25	20	17
						25	20
							25

outputs: o(2) o(3) o(4) . . . o(n)

$O(n^2)$

idea1 $O(n)$

idea2

idea3

n

8, 9, 17, 20, 25

$\sim n/2$ $\sim n/2$

$\frac{n}{2}$th largest item

$\frac{n}{2}$th smallest item

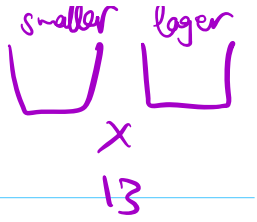
8
9
17
20
25

max heap of small item

min heap of large item

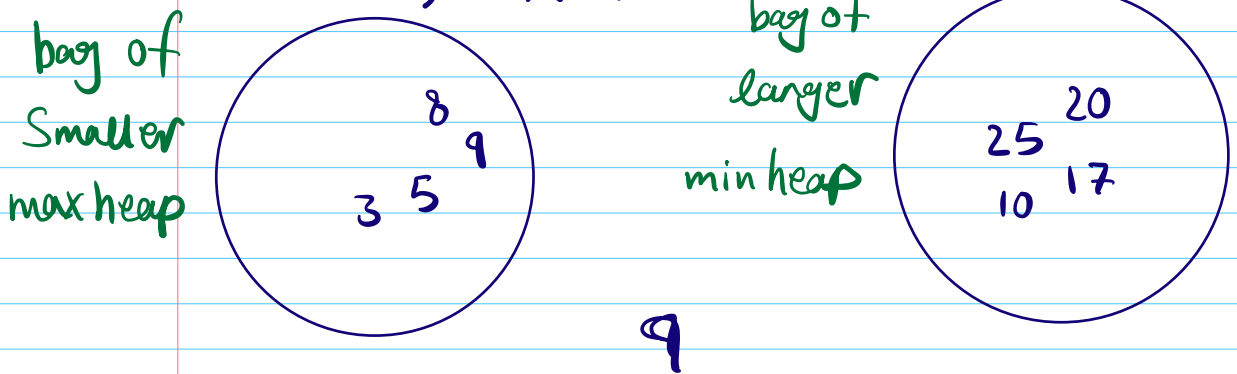
```
for each X{
    new number in stream is X
    if( X <= median){          // small items
        insert X in maxHeap
        if((size of maxHeap - sizeof MinHeap)>1){
            remove root from maxHeap
            & insert to minHeap
        }
    }
    else{ // try adding in larger bag ( min_heap)
        insert X in minHeap
        if((size of minHeap>sizeofmaxHeap) ){
            remove root from minHeap
            & insert to maxHeap
        }
    }
    median = maxHeap.root  → ans.add(median)
}
```

median = A[0]



check if(maxHeap.size+minHeap.size)%2==1

9, 8, 9, 9, 17, 10, 10, 9



9

TC: n log(n/2)