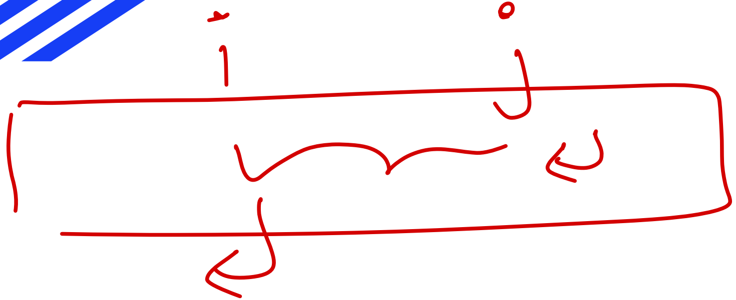
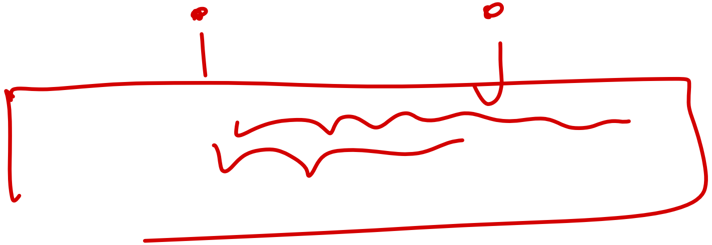
fixed length

Sliding Window

{variable length}



Two Pointers



$j = i+k-1$

- Priyansh Agarwal

Problem 1

Distance between two coordinates **x and y** is defined as absolute difference between the two. $A_3 > A_1$

$\{A_1 - A_3$, $A_3 - A_1\}$ $(A_1 - A_3)$

Given an array **nums** having **n** positive integers $\mathbf{A_1, A_2, \ldots, A_n}$, and an positive integer **k.** $\{1 \le k \le {}^nC_2\}$ $(A_3 - A_1)$

Return the **k**th smallest distance among all the pairs of integers **nums[i]** and **nums[j]** where $0 <= i < j < n$.

Constraints: $\mathbf{1 <= n <= 10^5}$, $\mathbf{1 <= A_i <= 10^5}$.

$\log(10^5 - 1)$

$\text{dist}(\text{point}_i, \text{point}_j)$

(i, j)

(j, i)

$0 \le i < j < n$

$0 \to n-1$, $1 \to n-2$

$^nC_2 \to n \cdot (n-1)/2$

1 2 3 100

1 1 97

consecutive pairs

$1 \leq k \leq n$

2

3rd smallest

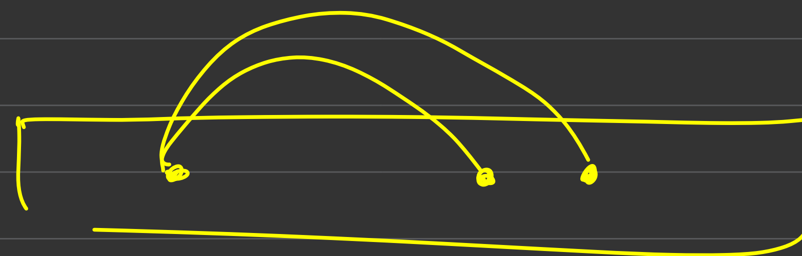
( ) ( ) ( ) ( ) ( )

iterate on 1st 2nd —— —— kth

$1 \le k \le \boxed{{}^{n}C_2}$

sorting all differe $\longrightarrow$ picking the kth

${}^{n}C_2 =$



$> x$

Consider diff b/w pair of elements

$\{(0 \le i < j < n)\}$

not sort

dif

kth smallest diff among all pairs

- Binary Search on Answer
- Find number of pairs in array whose diff < k → $\underline{O(n)}$

$i \longrightarrow j$

$$\boxed{a[j] - a[i] \le k}$$

how many xs $=$ $(j - i)$

$a[x] - a[i] \le k$

$0 \quad 5 \quad 7$

$i \quad 1$

$a[i]$

$(i - j)$

$a[x] - a[\square] \leq k$

$(5 - 0 = 5)$

$(7 - 1) = 6$

$a[x] - a[1] \leq k$

$a[j] - a[i] \leq k$

$$\boxed{a[j] - a[i+1] \leq k}$$

(x, y)

number of pair with diff $\leq$ k

$O(n)$

while (i < n) {

$O(n)$

while(j<n) {

if(a[j+1] - a[i] $\leq$ k)

j++

else

break)

ans += (j-i) }

O(n) → find # of pair with diff ≤ x

O(log(max diff))

kth smallest diff

pf → TTTTTFFFFFF

T → # of pair with diff < x or < k

F → # of " " " < x or ≥ k

1st smallest $< x$

$\longrightarrow$ 0 pairs

2nd smallest $< x$

$\longrightarrow$ 1 pair

Ans = 0

Binary Search on Diff → $\boxed{x}$

{

if ($f(\overset{mid}{x})$ == T) → $\underline{O(n)}$

{ ans = max (ans, mid)
start = mid + 1
}

end = mid − 1

}

$< k$

Binary search , 2 pointer

$\{ i \qquad j \}$ $\qquad a[j] - a[i] < x$

$(j - i)$

$a[y] - a[i] < x \qquad j-1$

$\longrightarrow \qquad i \quad j \qquad (j-i)$

kth smallest diff

$\rightarrow (a[j] - a[i]) \rightarrow$ x

$\{ a[1]-a[0]$, $a[2]-a[1]$, $a[2]-a[0]$

$- - - a[n] \}$

$\underline{\underline{< k}}$

kth smallest

kth smallest diff = the diff for which

no. of pair with diff $\leq$ $\underline{< k}$

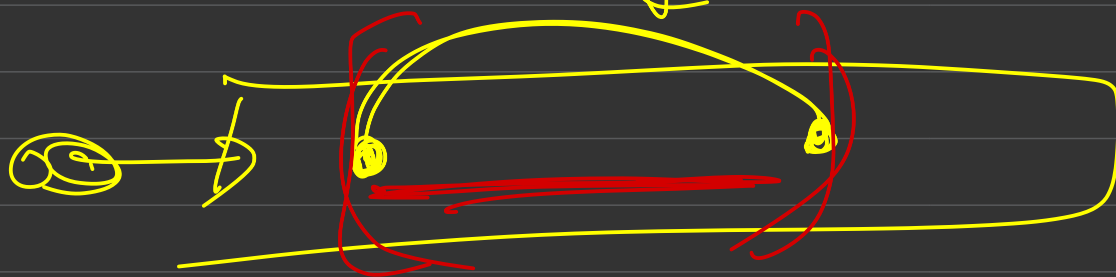
$\{ x \to \#$ of pairs with diff $< x \}$

y

$\boxed{y < k}$

ff TTTTTFFFFF $\boxed{< k}$

Binary → (2 pointer)



$a[i] - a[j] < x$

$O(\log n \cdot (n \log n))$

$O(n \log n \cdot n)$

Binary Search idea $\rightarrow$ 2 point

$n \log n \longrightarrow O(n)$

Sliding Window

fixed length

- Useful for array based problems - subarray
- When to use?
- Optimization Technique
- Use of 2 pointers.
- Super useful for interviews too

$O(n^2) \rightarrow O(n)$

fixed length segment

$i \longrightarrow j$

$j = i + k - 1$

k

Given an array, what is the maximum sum of a subarray of size k

1	3	4	2	7	6	3	1	2	4

K = 3

$i \rightarrow i+k-1$

x

$i \rightarrow i+k-1$

sum available

ans = 0

for (i=0, i<n; i++)

{ sum = 0

for (j=i; j < i+k ; j++)

sum += a[j]

}

ans = max (ans, sum)

}

O(n x k)

k < n

O(n^2)

$\boxed{n \times k}$

i

subarrays of size $k \longrightarrow \underline{\underline{n-k+1}}$

$\downarrow$

sum$(i, \; i+k-1)$

$O(n)$

for $(i=0, \; i<n, \; i{+}{+})$ $\underline{\underline{}}$

$O(k)$ (ans = max(ans, sum(i, i+k-1)))

$O(n) \rightarrow$ space

$[0 \text{ —— } k-1]$

sum $\rightarrow$ $O(k)$

$[1 \text{ —— } k]$

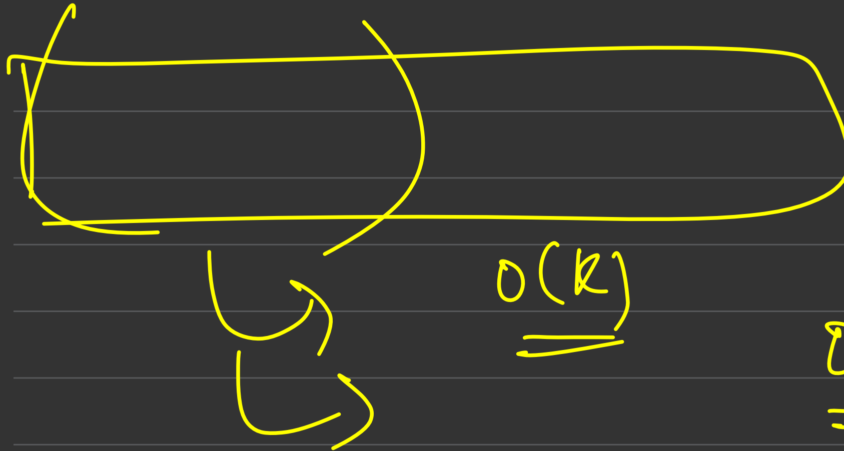
$[2 \text{ —— } k+1]$

$O(1)$ $O(1)$

sum + a[k] − a[0]

$+\,a[k+1] - a[1]$

$O(1)$



$O(k)$

$O(1)$

$O(k) + O(n)$

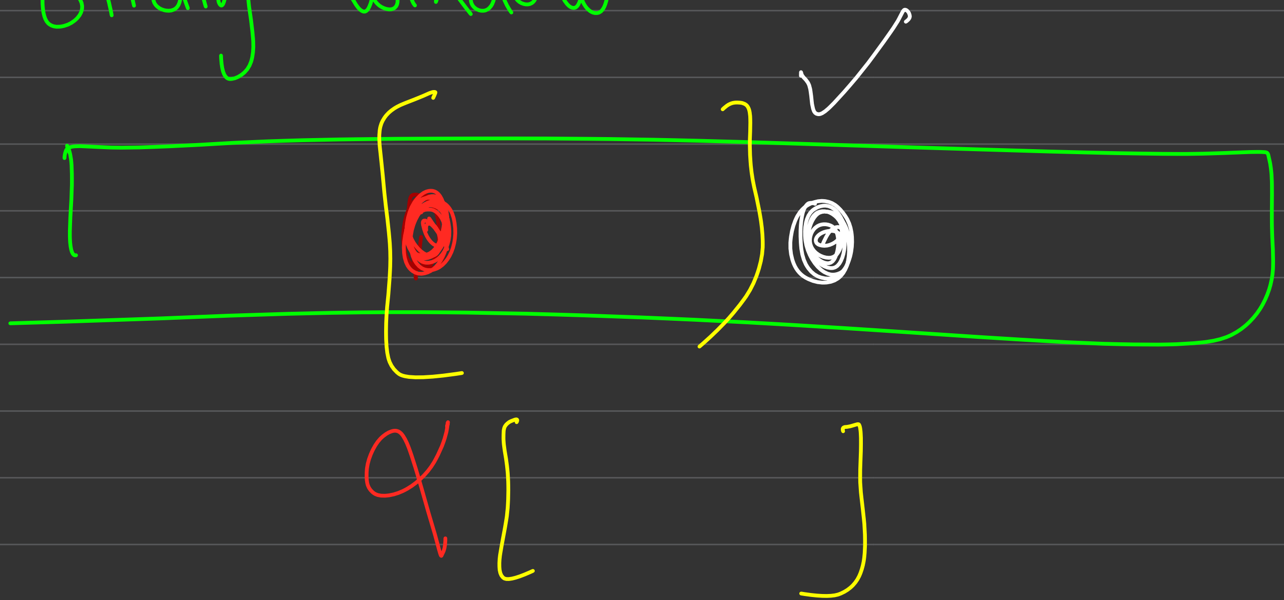
```
sum = 0
for (i=0; i<k; i++) {
    sum += a[i]
}

ans = sum

for (i=1; i+k-1 < n ; i++)
{
    sum += a[i+k-1]
    sum -= a[i-1]
    ans = max(ans, sum)
}
```

$i \longrightarrow i+k-1 \le n-1$

$\downarrow$

$< n$

Sliding window



Given an array, find the first negative number in every subarray of size k

1	-2	-3	4	-6	-5	1	2	3	-1

-2

-2

-3

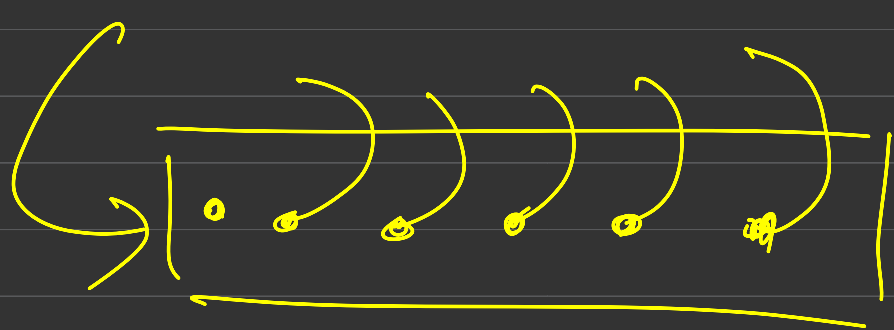
$n-k+1$

$O(n)$

O(1) space $\leftarrow$ $\underline{\underline{O(n)}}$

O(n) time

deque



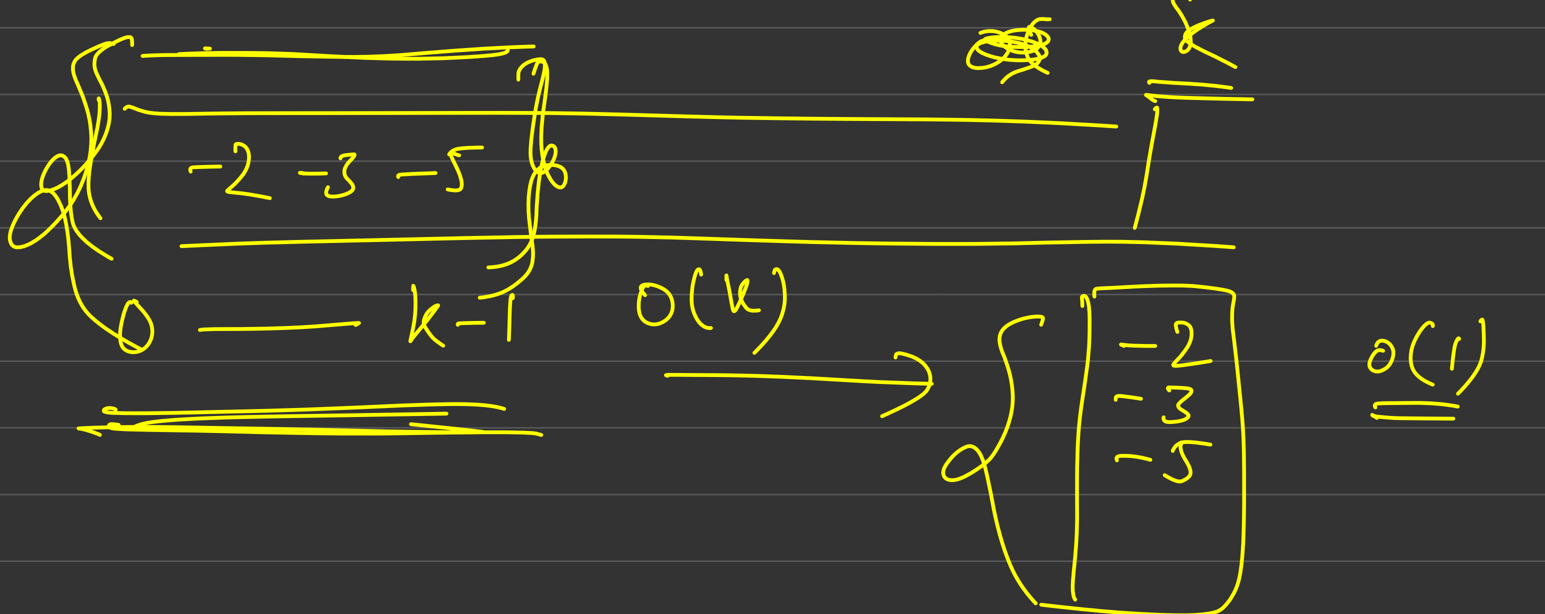
4 operations → O(1)

push_back   pop_back → O(1)

→ push_front → vector

pop_front → O(1)

first negative element in every subarray of size k



deque d ;

for (i = 0 ; i < k ; i++)

{ if (a[i] < 0)

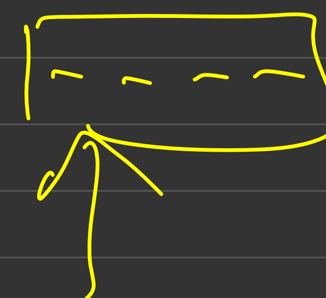
d.push_back(i) }

2	-3	-1	2	1	3	4	5

1
2

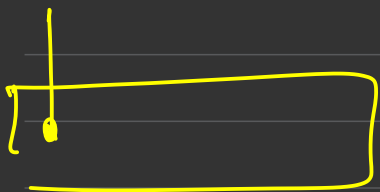
0	∞

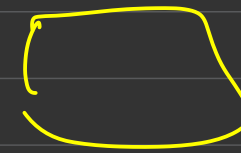
```
vector<int> ans(n-k+1, -INF)
deque d;
for(i=0, i<k, i++)
    if(a[i]<0)
        d.push_back(i)
if(!d.empty)
    ans[0] = a[d.front()]
```



```
for ( i=1 ; i+k-1 < n , i++ )
{
    if (a[i+k-1] < 0)
        d.push_back(i+k-1);
    if (d.front() == i-1)
        d.pop_front()
    if(!d.empty())
        ans[i] = a[d.front()]
}
```

i-1





$i-1$

$i+k-1$

0

i

0

time complexity → $O(n)$

space complexity → $O(n)$ → $O(n)$

3 pointers

$[i \text{ --- } j]$

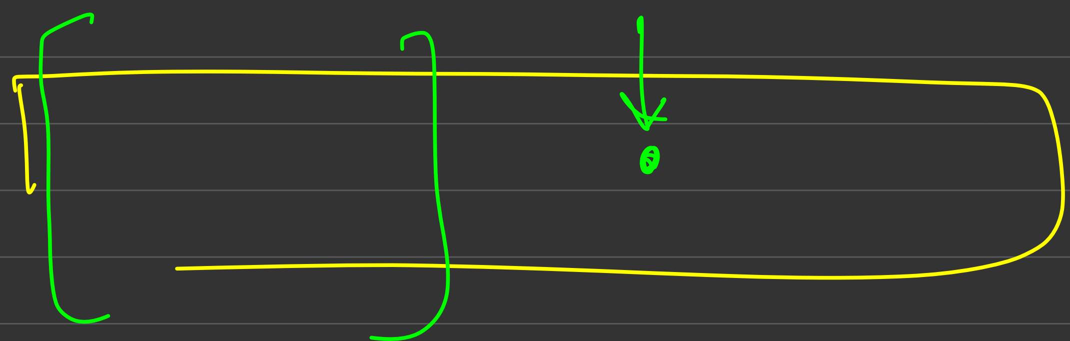
i , $i+k-1$, l l l

$a[l]$

$(l \geq i)$ and $a[l]$ is the first negative element from $(i$ to $n-1)$

find a valid $l = n$

```
for (i=0 , i<n , i++)
    if (a[i] < 0)
        l = i , break
```



find l → $O(n)$ once

$(l > i)$

for (i = 0 , i + k - 1 < n , i++)

{ if (l < i + k - 1)

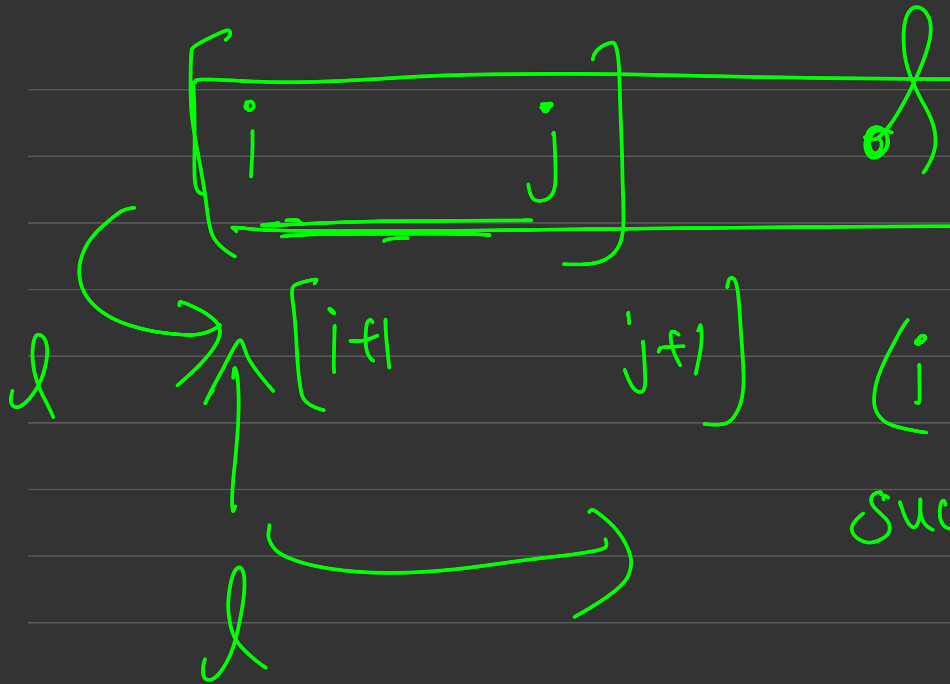
arr[i] = a[l]

i — i+k-1

(i , i+k-1)

l

(i+1, i+k)



$(i \le l \le n-1)$ such that $a[l]$ is the first negative element from $(i \text{ to } n-1)$

if $(l == i)$

$l = -1$

```
for (i=0 ; i+k-1 < n ; i++)
{
    if (l < i)
        l++;
    while (a[l] >= 0 && l < n)
        l++;
    if (l <= i+k-1)
        ans[i] = a[l]
}
```

$O(n^2)$

$i \longrightarrow O(n)$

$O(n)$

$l \longrightarrow O(n)$

i and l independent

queue , deque

2

time — $O(n)$

space — $O(1)$

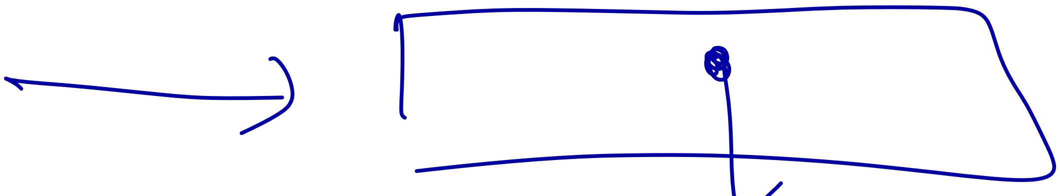
Given an array, find the median of each subarray of size k

k is odd here

No

i

i+k-1



$O(n)$

Ordered set → pbds

policy based datastructure

multiset

5th

O(logn)

O(n)

Youtube

insert → O(logn)

query to index → O(logn)

remove → O(logn)

0   k-1   ~~m~~   set

pbds A.
(pair<int, int>)

for (i=0, i<k, i++)
{
A.insert({a[i]}, i)
}

ans[0] = A[k/2].first
}

Concept

1 2 1 1 2 3

set (pair <int, int>)

↓

(2 , index)

```
for ( i = 1 , i+k-1 < n ; i++ )
{
    A.insert ( { a[i+k-1], i+k-1 } )      // O(1)
    A.erase ( { a[i-1], i-1 } )
    ans[i] = *A [k/2].first
               A.find_by_order (k/2)
```

Sliding window

→ fdsf

→ pairs in set

Pbds solutions

→ Fenwick Tree + Coordinate Compression

JAVA

Time complexity → $O(n \log k)$ → $O(n \log n)$

Space → $O(k)$ → $O(n)$

Given an array, find the minimum number in each subarray of size k

set(pair<int, int>)

multiset

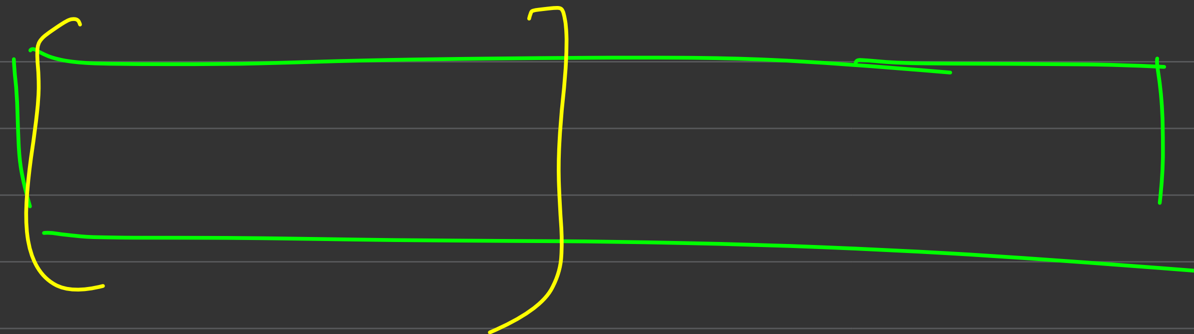
set → priority_queue

O(nlogn)

O(n)

priority queue faster than set

↳ Dijkstra



$\theta(n\log n)$

$\theta(n) \longrightarrow o(n\log$

deque

6 2 3 5 4

2 3 4

ans [0] = d

[6 2 0 5 0 3]

(6 5 4 3 2 0)

ans[0] = d.front()

d [2 0 0 3]

deque d

for (i = 0 , i < k , i++)

1

{ while (! d.empty && a[d.back()] > a[i])

{ d.pop_back() }

d.push_back(i)

}

ans[0] = a[d.front()]

```
for ( i=1 , i+k-1 < n , i++ )
{
    while( !d.empty() && a[d.back()]
O(n)                  > a[i+k-1] )
        d.pop_back()
    d.push_back(i+k-1)
}
```

{ { if (d.front() == i-1)

(d.pop_front())

ans[i] = a[d.front()]

}

[2 3 1] $O(n)$

~~(3 2)~~ ~~$O(n)$~~

(2, 3) 1

1 0 1 1

Solution:

- Sliding window
- Use of deque

```
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    deque<int> d;
    vector<int> ret;
    for(int i = 0; i < k; i++){
        while(!d.empty() && nums[i] > nums[d.back()]){
            d.pop_back();
        }
        d.push_back(i);
    }
    for(int i = k; i < nums.size(); i++){
        ret.push_back(nums[d.front()]);
        if(!d.empty() && d.front() <= i-k){
            d.pop_front();
        }
        while(!d.empty() && nums[i] >= nums[d.back()]){
            d.pop_back();
        }
        d.push_back(i);
    }
    ret.push_back(nums[d.front()]);
    return ret;
}
```

max element