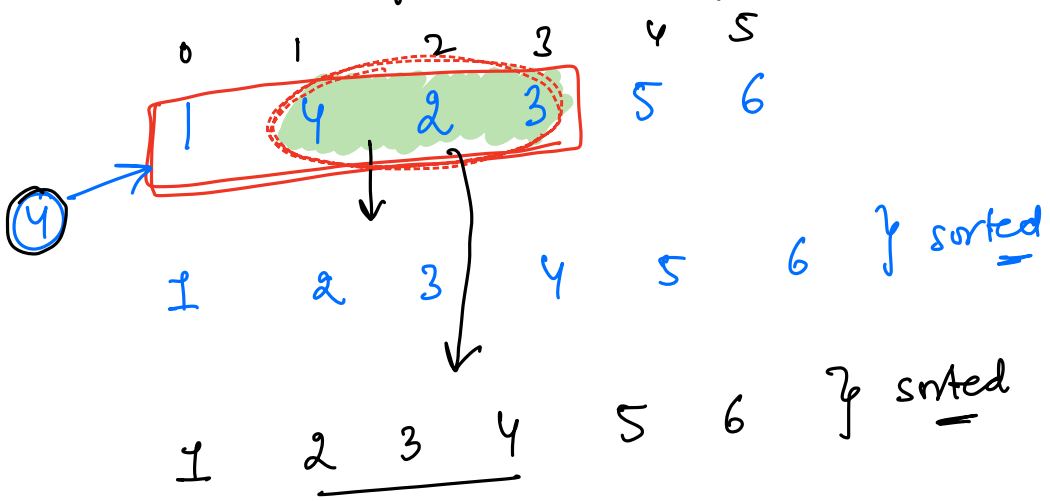
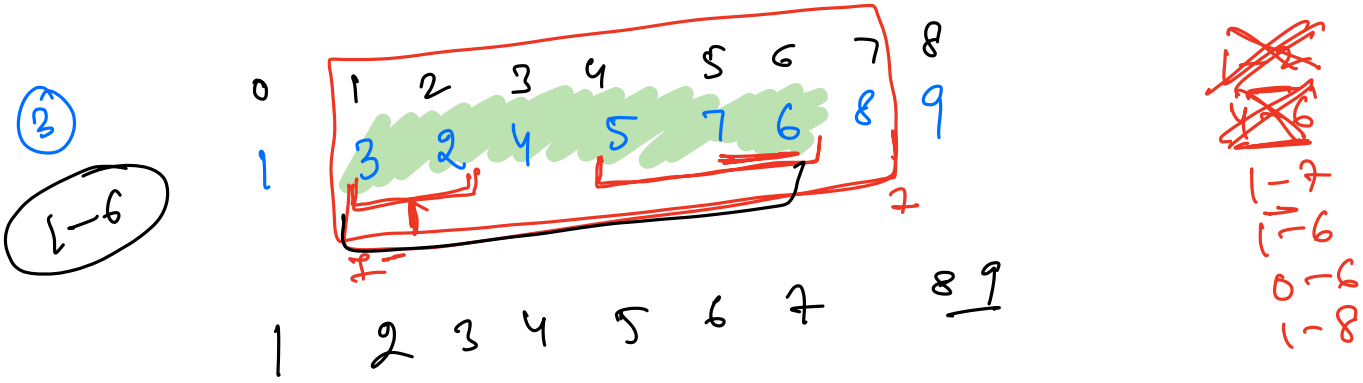
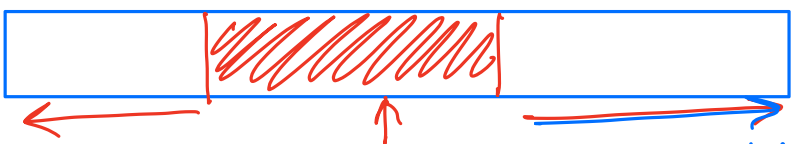
① you are given with an array of size N.
Find minimum length subarray such that if you sort that subarray, whole array will be sorted in asc order.



② 0 1 2 3 4 5
2 1 3 4 6 5
whole array
1 2 3 4 6 5 } sorted? ✗

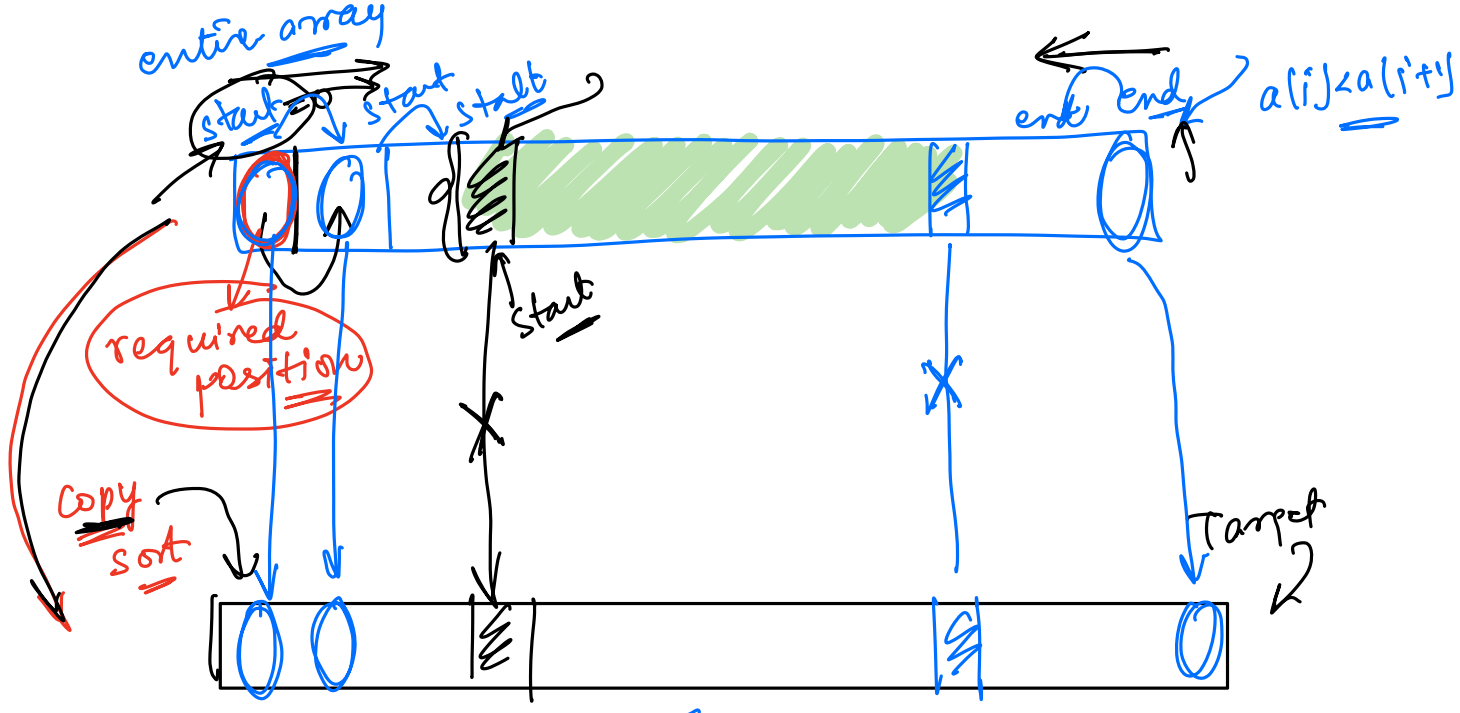


Soln

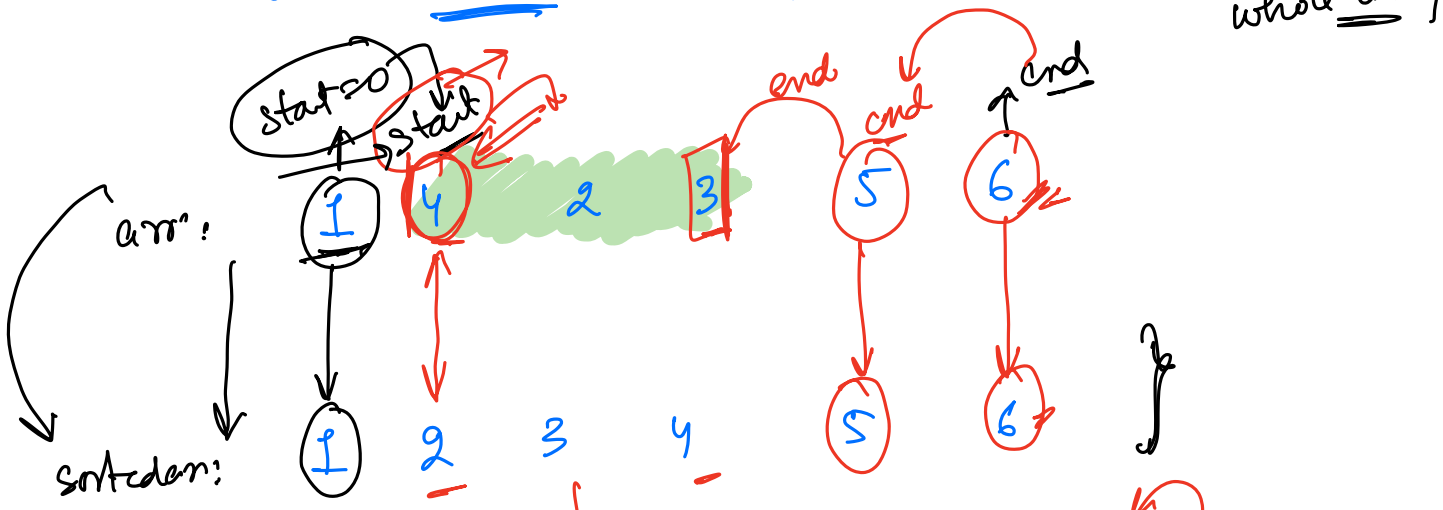


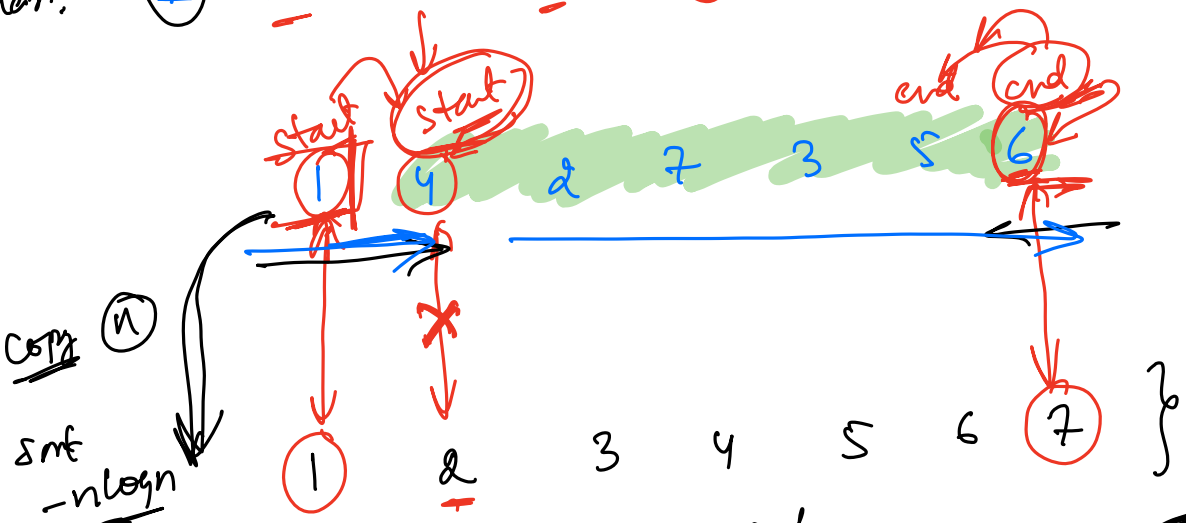
all on left → will be present in required pos.

all on right will be present on their required.

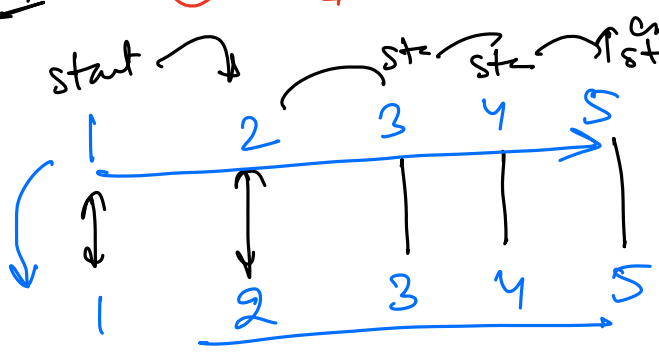


arr[i] == sotedarr[i]





N
nlogn
N



T.C: nlogn

if (start == n)
sorted

Manish

(2) array ⇒ N integers — choose B elements — $^{N}C_{B}$

max(B) − min(B) ↓ minimise

B <= N

diff

return minimum diff

N = 6 10 15 6 9 13 8 — B = 4

15 6 9 13 — max = 15, min = 6 — (9)

6 9 10 13 — max = 13, min = 6 — (7)

6 9 8 10 — max = 10, min = 6 — (4)

✗ Sort & first B elements — B = 4

1 2 3 100 101 102 103 — (3)

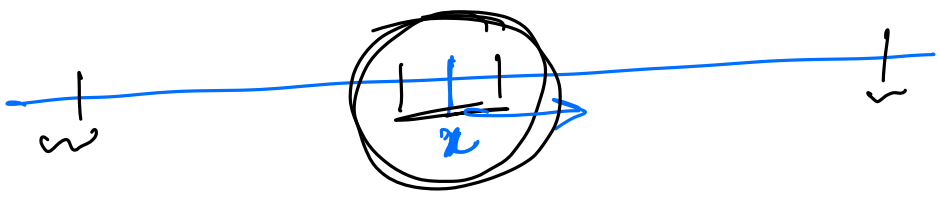
99

✗ least B elements

1 2 3 4 100 120 160 180 — 80

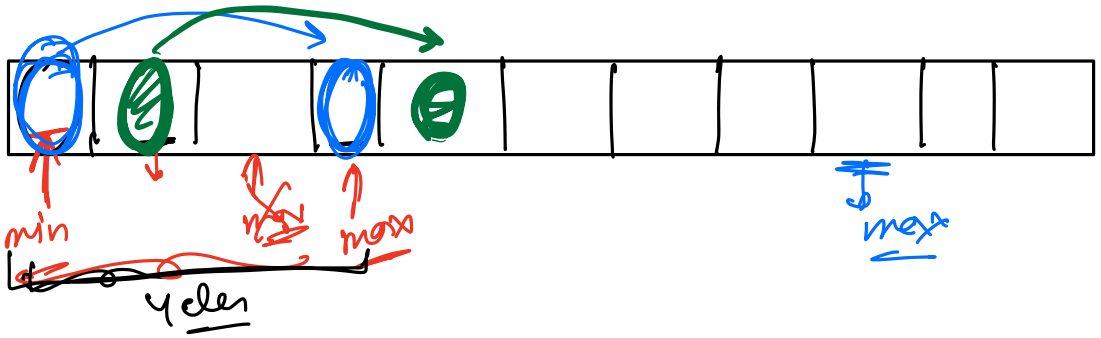
(3)

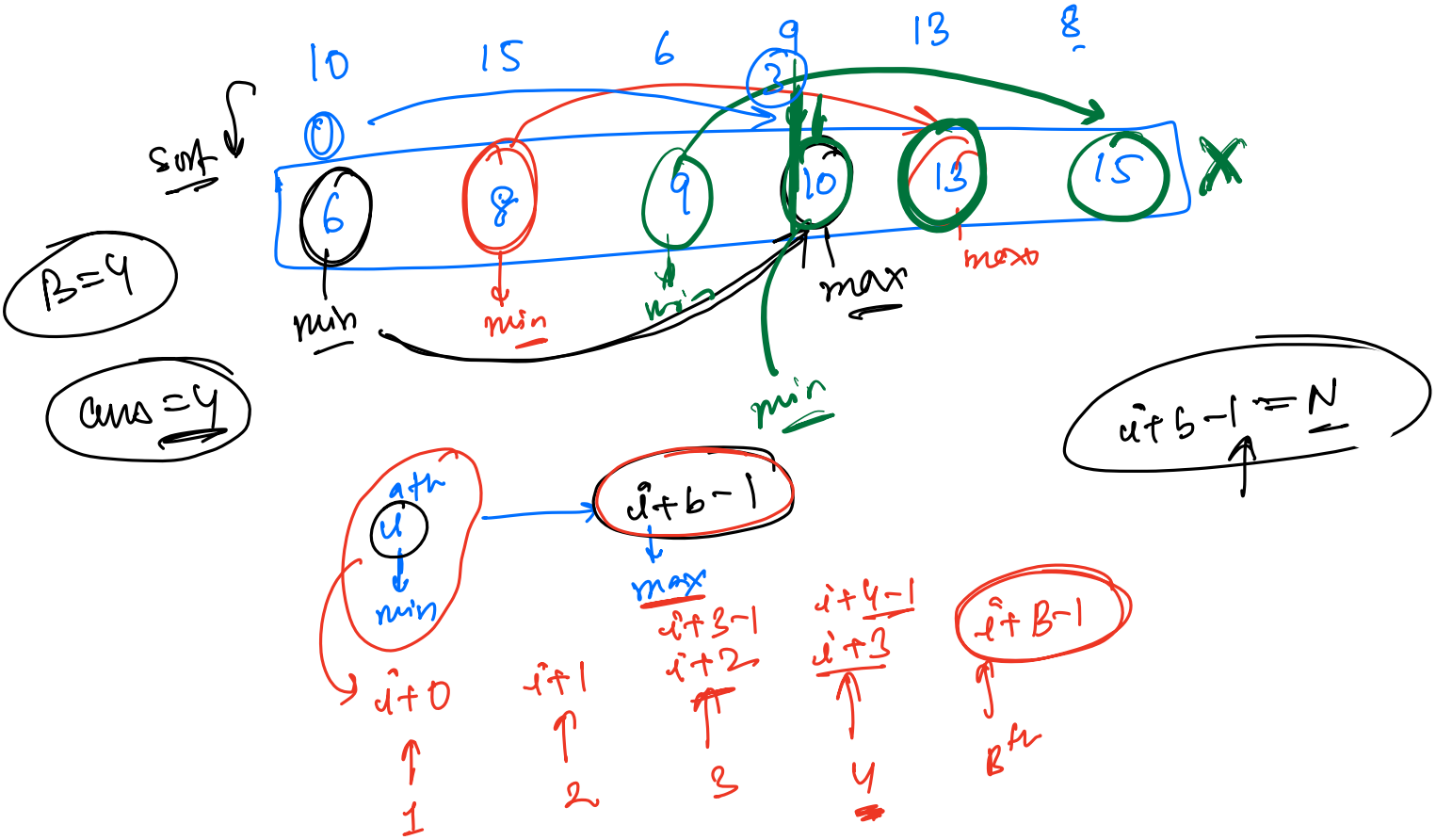
soln

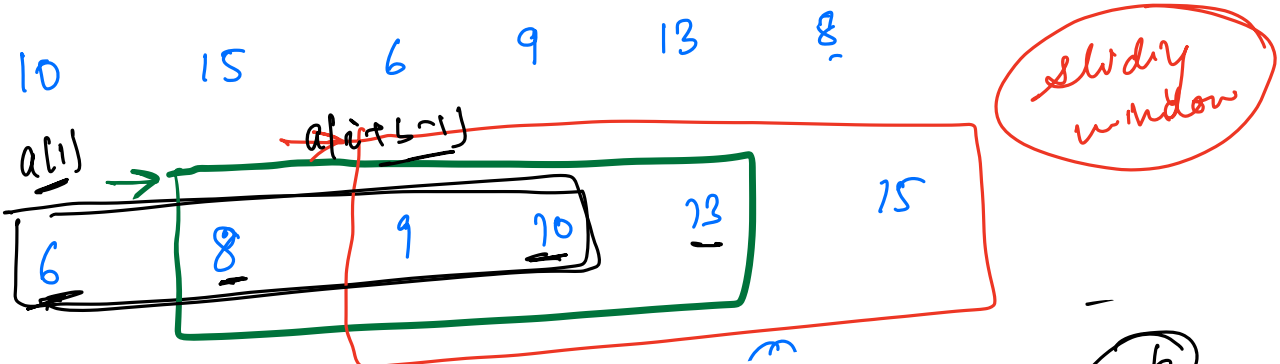


sort the array

B 4







1) sort

O(N)

ans = min(ans, a[i+b-1] - a[i]) i=0 →

n-b

T.C: nlogn

S.C: O(n)

subarray — continuous

subset — order

subsequence — order

O(n)

{5 7 1}

2 1 3 5 6 7

1 5 7

(3) Given array → N elements

distribute these elements into two, sets non-empty

A B

return A & B

|max(A) − min(B)| ↓ minimise

-3 1 6 2 4

A B

-3 1 2 — 2

4 6 — 4

diff = |2-4| = 2

diff = 2

-3 1 2 6 — 6

4 — 4

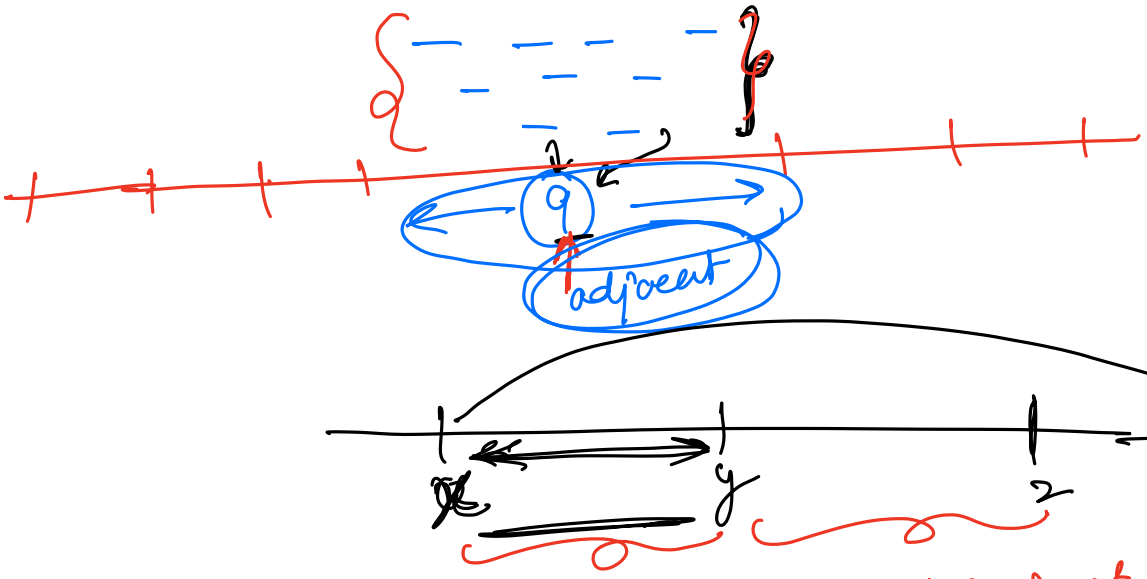
-3 — 1

2 4 6 — 2

diff = 1

0

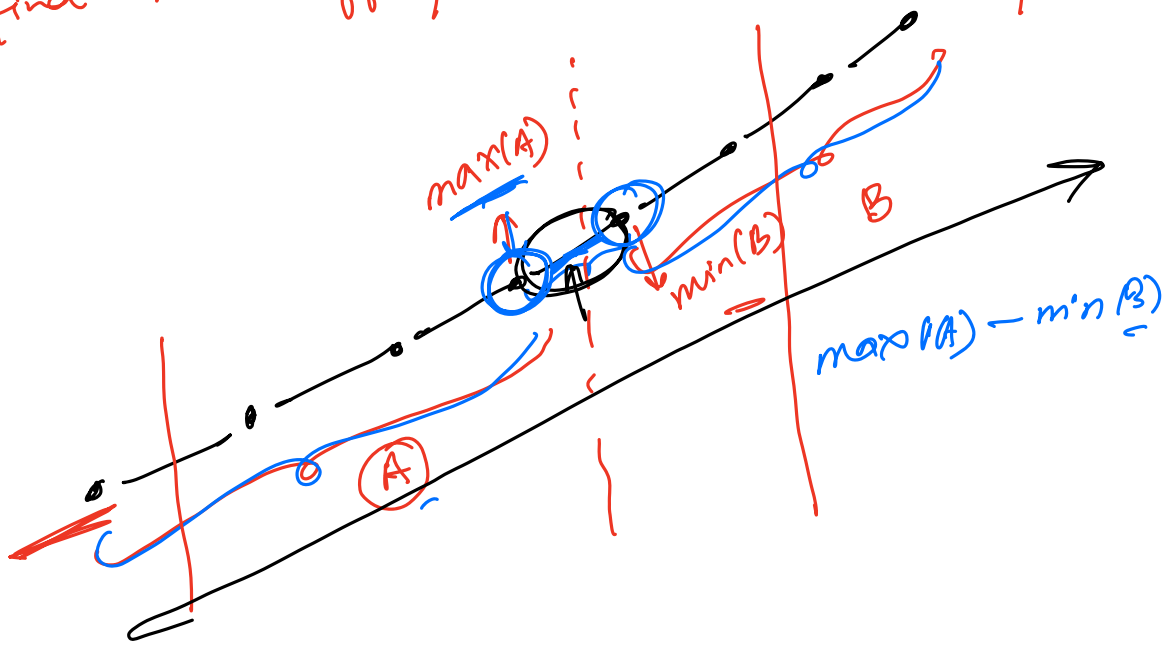
1) looking for a pair which would give minimum abs diff



n^2

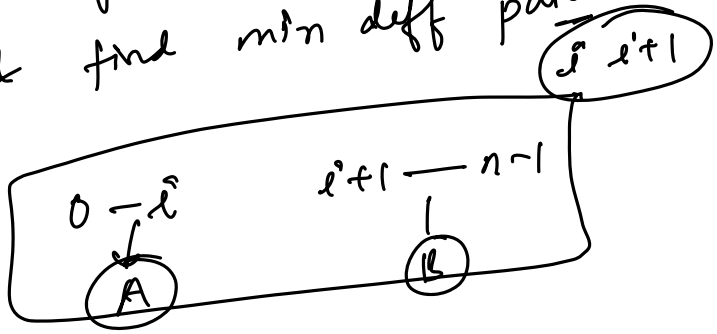
n pair

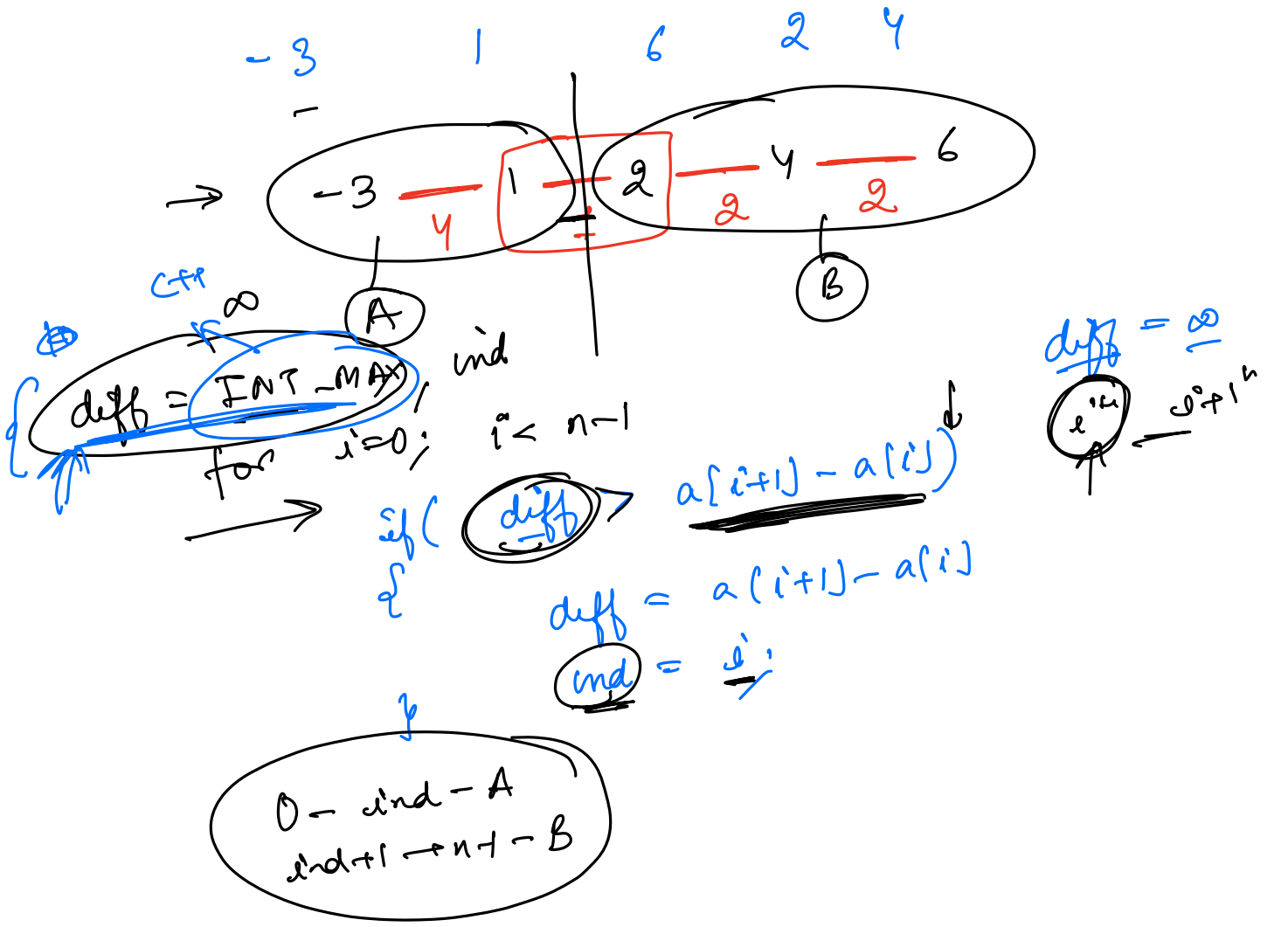
* to find min diff pair — sort & just check adjacent elems



1) sort the array
2) Traverse & find min diff pair
3) distribute

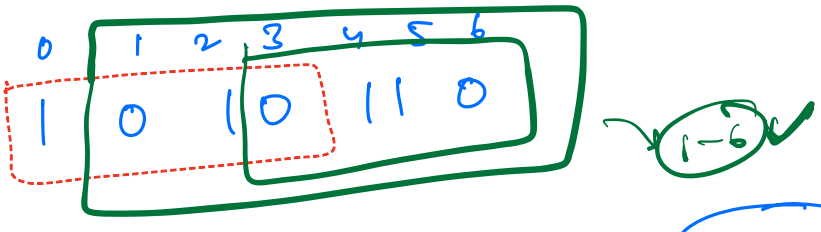
diff index





(4) Array given with n elements — 0's & 1's

length of largest subarray with equal 0's = 1's



1-6 ?

consider every possible subarray

start end

B.F

start end

```
for start = 0 → n
    for end = start → n
        (start end)
        for start → end
```

T.C: n^3

3 3 →

n is odd

only 1 will be contributing to sum

#0 = #1

sum = n/2

sum = len/2

→ prefix sum

O(1)

pfsum for

ans = 0

start = 0 → n

end = start → n

[start, end]

len = end - start + 1; — O(1)

if (len % 2 == 0) — O(1)

sum = len/2; — O(1)

actual sum = pf[end] - pf[start-1] — O(1)

if (sum == actual sum) — O(1)

ans = max(ans, len); — O(1)

T.C = n^2

1 1 1 1 0

6 = 6/2

0 0

2

4

3

1 0 1 0 1 0

1 1 2 2 3 4 4

4 → 2

2

3

3-1 = 2

sum

n^3 → n^2 → n → _

0 → 1

-1

#0's - #1's

Sum subarray = 0

0

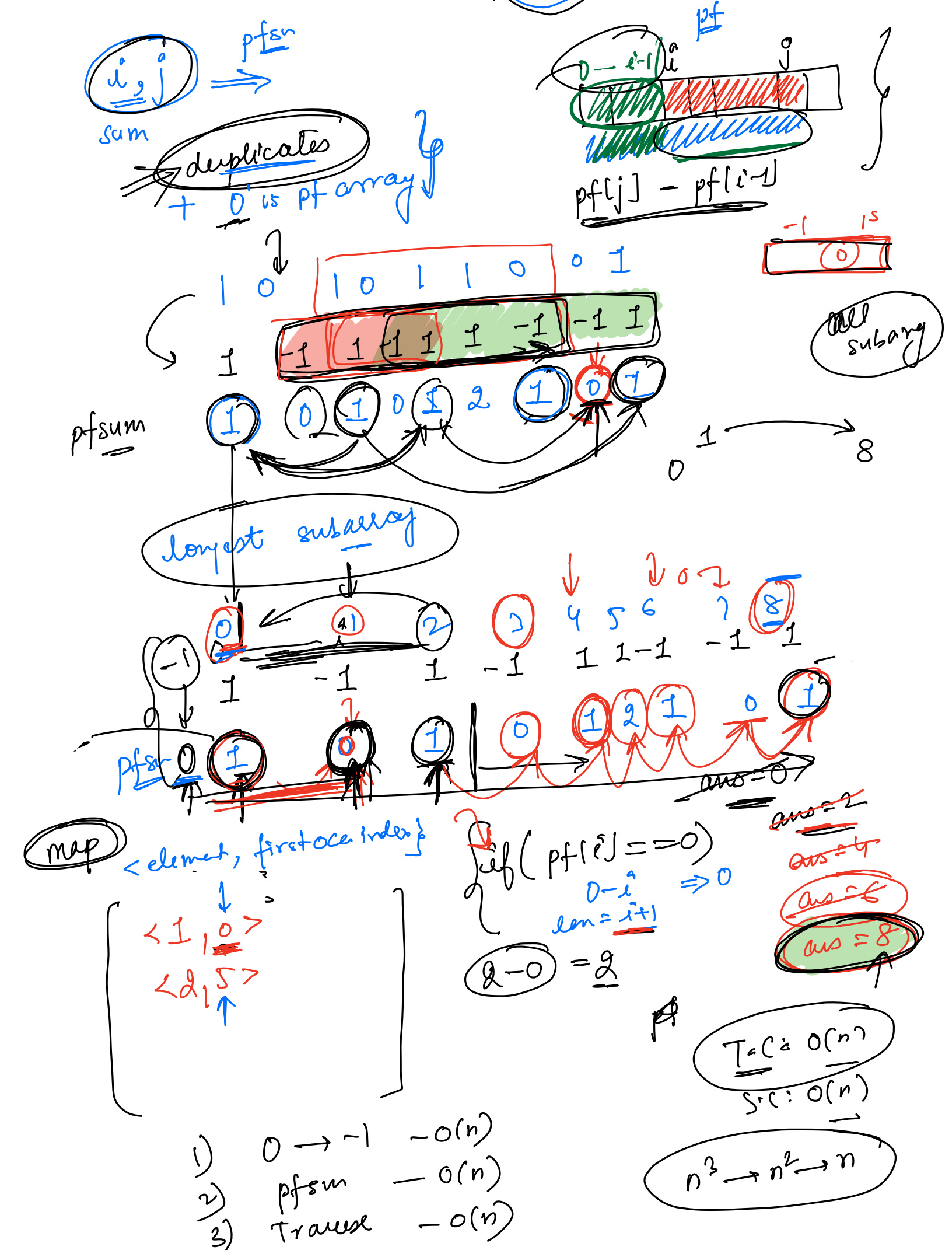
-1

i, j

subarray sum

pf[j] - pf[i-1] = 0

pf[j] = pf[i-1]


i, j
pfsm
sum
duplicates
+ 0 is pf array
pf
0 — i-1
i
j
pf[j] - pf[i-1]
-1 is
0
all subarray
pfsum
1 0 1 1 0 0 1
-1 1 1 1 1 -1 -1 1
1 0 1 0 1 2 1 0 1
0 1 8
longest subarray
0 1 2 3 4 5 6 7 8
1 -1 1 -1 1 1 -1 -1 1
pfsm
0 1 2 1 0 1
ans=0
map
<element, firstoccindex>
<1, 0>
<2, 5>
if(pf[i]==0)
0-i
⇒0
len = i+1
8-0 = 8
ans=2
ans=4
ans=6
ans=8
T.C: O(n)
S.C: O(n)
n³ → n² → n
1) 0 → -1 — O(n)
2) pfsm — O(n)
3) Traverse — O(n)