2ptr

* Arrays.sort(arr)

↳ $O(n \cdot \log_2 n)$ ✓

S5

S2-Class-3[Sorting-1]

$O(n^2)$?
✓

① Bubble sort.

② selection sort

~~BF~~ → 2ptr
s|w

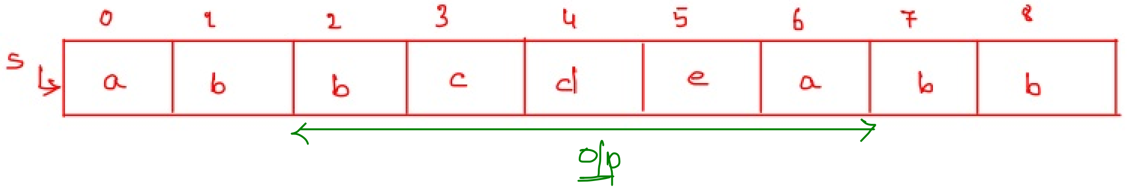
$a+b = k$

$O(n^2)$ x → nlogn x

↓

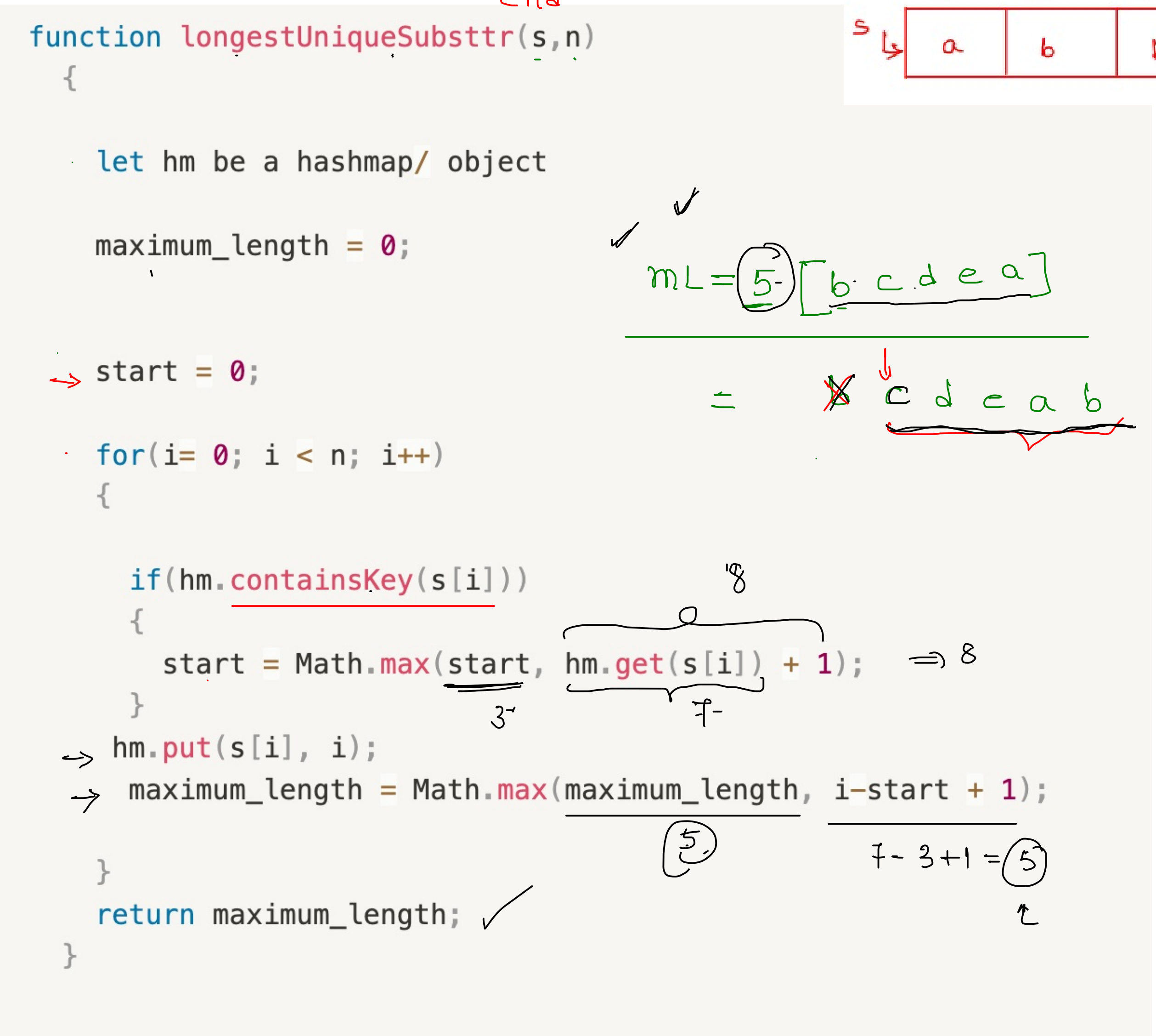
$\boxed{n}$ ✓

9) Find the size of largest sub-string which doesn't contains any repeated characters in given string



	0	1	2	3	4	5	6	7	8
s	a	b	b	c	d	e	a	b	b

length:- beg & end end-beg+1



```
function longestUniqueSubsttr(s,n)
 {

  let hm be a hashmap/ object

  maximum_length = 0;


  start = 0;

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

    if(hm.containsKey(s[i]))
    {
      start = Math.max(start, hm.get(s[i]) + 1);
    }
   hm.put(s[i], i);
   maximum_length = Math.max(maximum_length, i-start + 1);

  }
  return maximum_length;
 }
```

8

⇒ 8

3

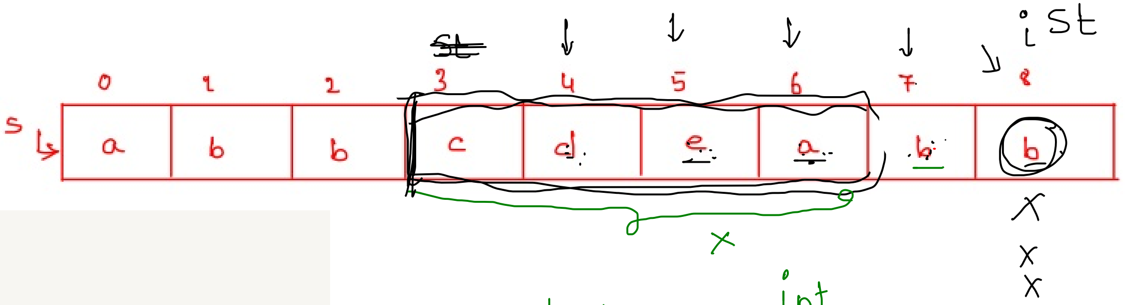
7

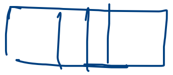
5

7 - 3+1 = 5

mL = 5 [b c d e a]

= ~~b~~ c d e a b





largest window + No repeating char

hm

Key	Value
~~a~~	~~0~~
~~b~~	~~2~~
c	3
d	4
e	5
a	6
b	7

char

int

index of array

✓1) Bubble sort

n=6 ✓

a ↳

0	1	2	3	4	5
9	7	6	4	2	1

o/p

a ↳

0	1	2	3	4	5
1	2	4	6	7	9

obj *

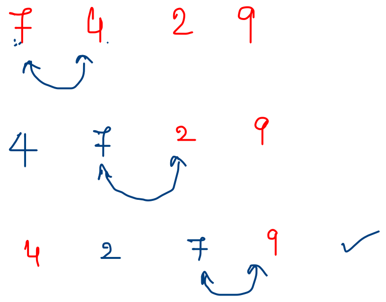
✓ ↑ order (1, 2, 3, 4 . - -)

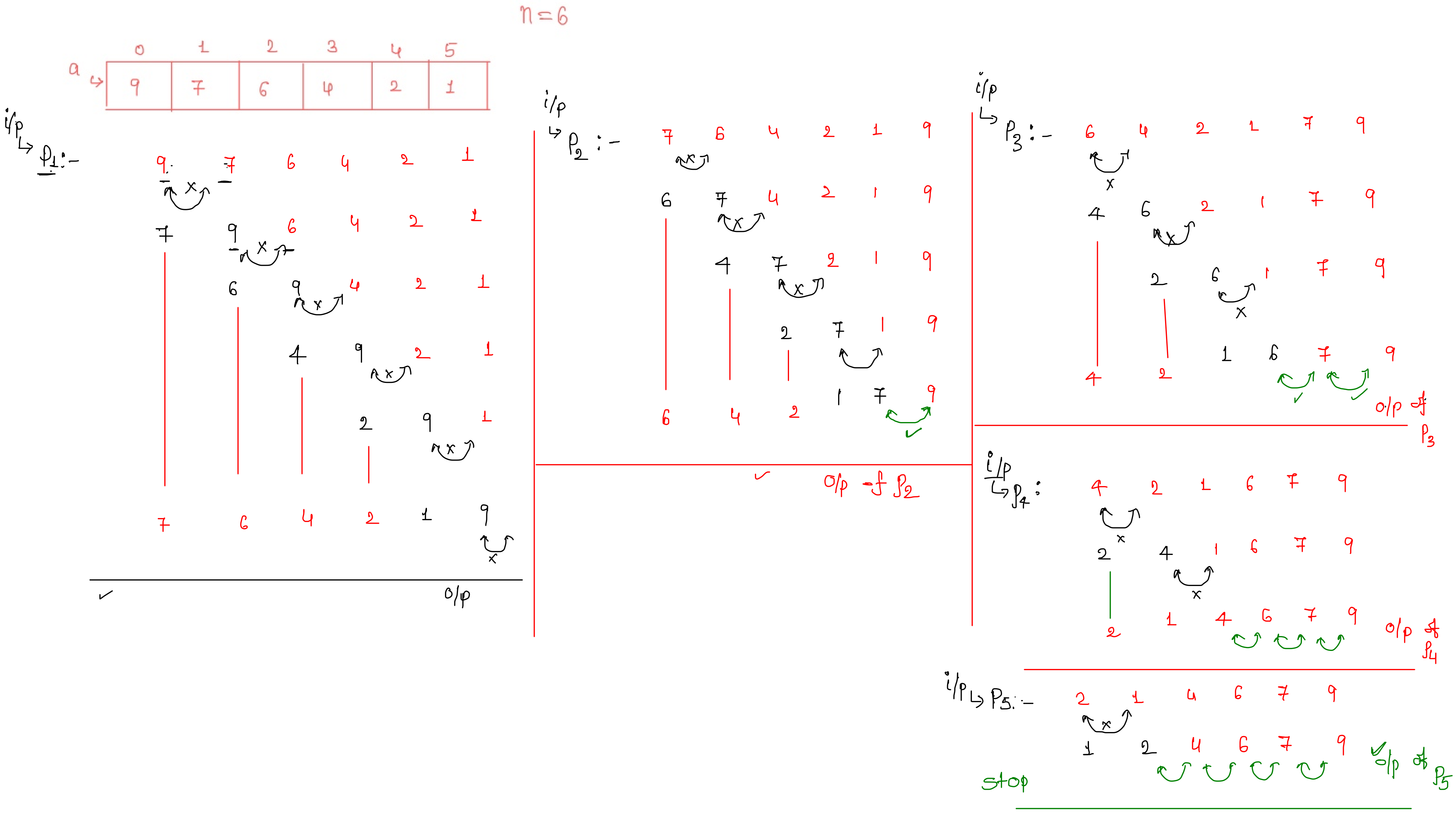
✓ ↓ order (4, 3, 2, 1 - - -)

2ptr :- ↑ order

. . . ↓ order

idea : works by repeatedly swapping the adjacent elements if they are "not in the proper order" ↑ order

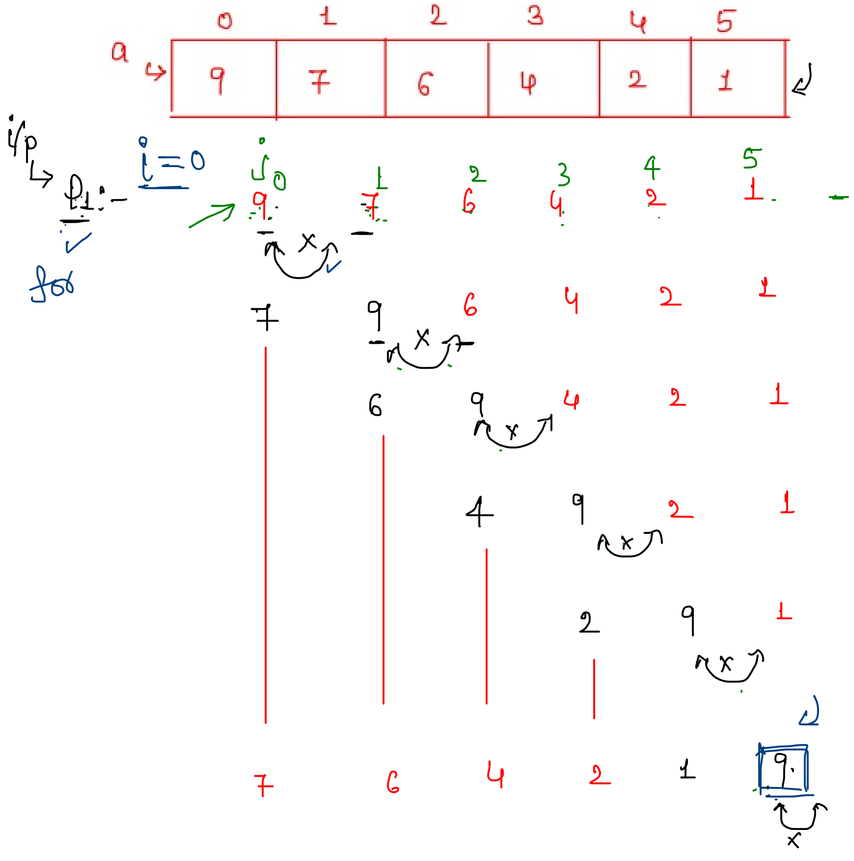



n=6
a
0 1 2 3 4 5
9 7 6 4 2 1
i/p ↳ P1 :-
9 7 6 4 2 1
7 9 6 4 2 1
6 9 4 2 1
4 9 2 1
2 9 1
7 6 4 2 1 9
O/p
i/p ↳ P2 :-
7 6 4 2 1 9
6 7 4 2 1 9
4 7 2 1 9
2 7 1 9
1 7 9
6 4 2 1 7 9
O/p of P2
i/p ↳ P3 :-
6 4 2 1 7 9
4 6 2 1 7 9
2 6 1 7 9
1 6 7 9
4 2 1 6 7 9
O/p of P3
i/p ↳ P4 :
4 2 1 6 7 9
2 4 1 6 7 9
2 1 4 6 7 9
O/p of P4
i/p ↳ P5 :-
2 1 4 6 7 9
1 2 4 6 7 9
O/p of P5
stop

n=6 ✓

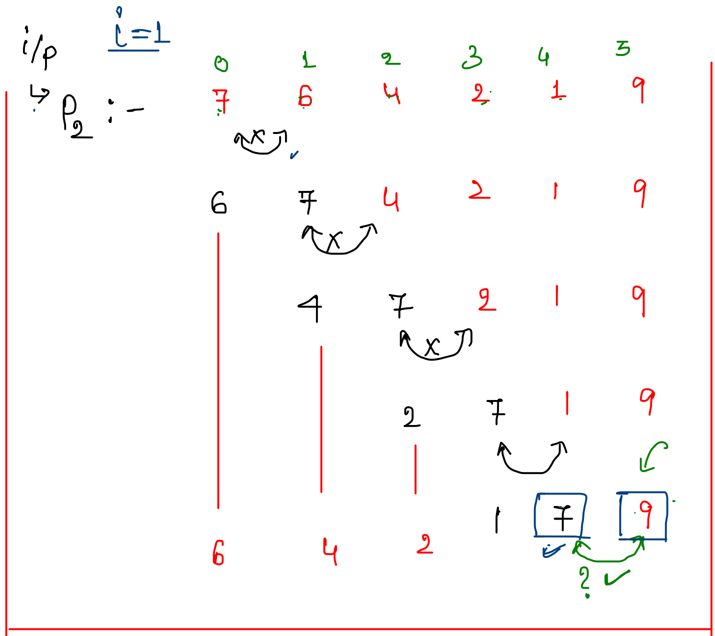
5 elets prop ✓

* n-1 passes are enough *



✓ i=0, j=~~0~~ ~~1~~ ~~2~~ ~~3~~ 4 o/p

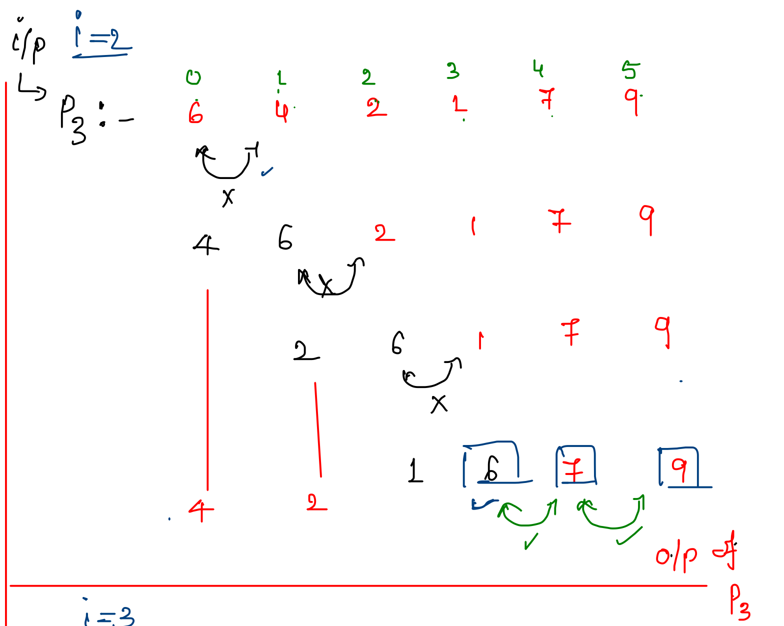
0 → 4

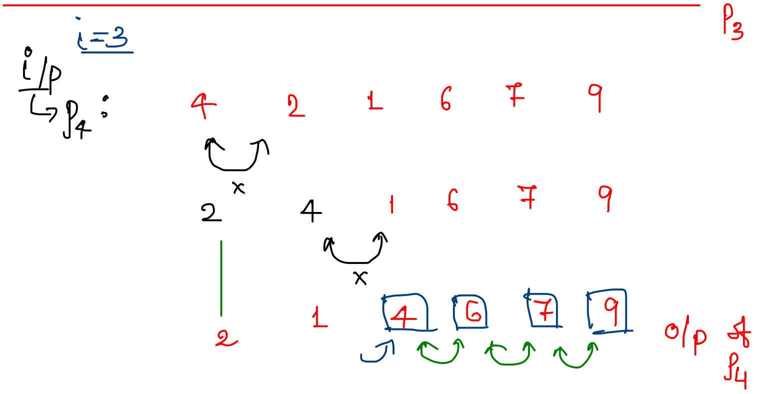


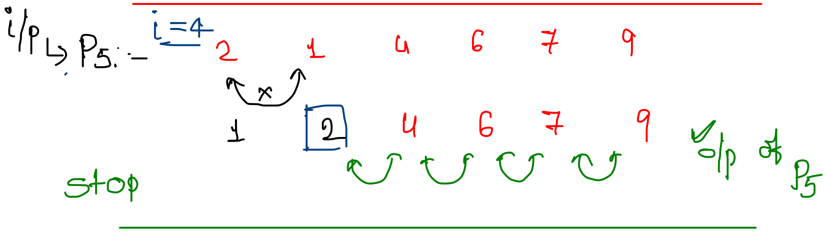
i=1, j=~~0~~ ~~1~~ ~~2~~ 3 o/p of P2

```
for(i=0; i<n-1; i++)
{
    for(j=0; j< ? ? ; j++)
    {
        if(a[j]>a[j+1])
        {
            swap(a[j],a[j+1]);
        }
    }
}
```

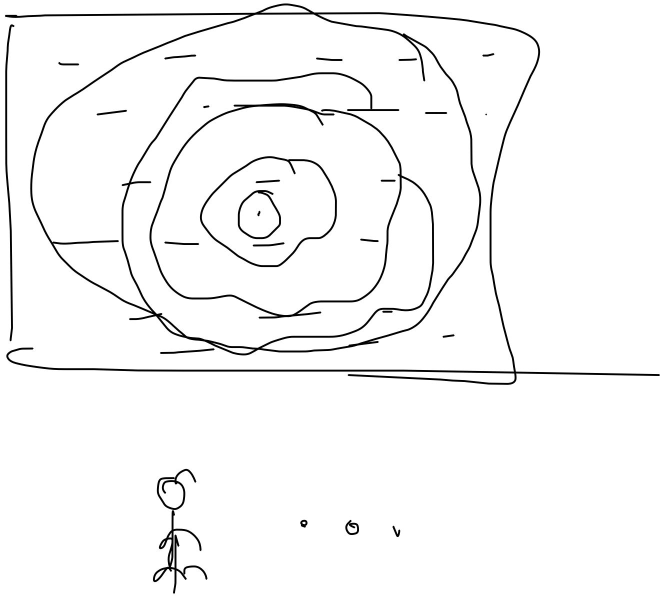
i=2, j=~~0~~ ~~1~~ ~~2~~ 3







stop



```
for(i=0; i<n-1; i++)
{
    for(j=0; j<n-i-1; j++)
    {
        if(a[j] > a[j+1])
        {   swap(a[j], a[j+1]);
        }
    }
}
```

$n = 6$ ✓

$\to (6-0-2)$

$i=0, \quad j=0 \quad$ to $\quad 4 \quad \}$

$\to (6-1-2)$

$i=1, \quad j=0 \quad$ to $\quad 3$

$i=2, \quad j=0 \quad$ to $\quad 2$

$\hookrightarrow (6-2-2)$

$\leq n-i-2$

$< n-i-1$

```
void bubbleSort(int arr[], int n)
{

        for(int i=0;i<n-1;i++)
        {
                for(int j=0;j<n-i-1;j++)
                {
                        if(arr[j]>arr[j+1])
                        {
                                int temp=arr[j];
                                arr[j]=arr[j+1];
                                arr[j+1]=temp;
                        }
                }
        }

}
```

Idea

F

T

swap (a[j], a[j+1])

worst case

TC:-

comp's + swap's

n = 7

a

i	0	1	2	3	4	5	6
a	7	6	5	4	3	2	1

i	comp's	swap's
0	6	6
1	5	5
2	4	4
3	3	3
4	2	2
5	1	1

total comp's: $1+2+\cdots+n-1 = \frac{n(n-1)}{2}$

swaps: ,, ,, $= \frac{n(n-1)}{2}$

$+ \quad \cancel{2}\frac{n(n-1)}{\cancel{2}}$

$n(n-1) = O(n^2)$

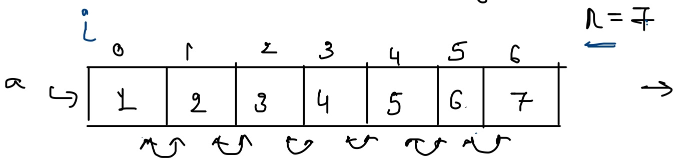
12:30

```
void bubbleSort(int arr[], int n)
{

          for(int i=0;i<n-1;i++)
          {

                    int swap_count=0;
                    for(int j=0;j<n-i-1;j++)
                    {
                              if(arr[j]>arr[j+1])
                              {
                                        int temp=arr[j];
                                        arr[j]=arr[j+1];
                                        arr[j+1]=temp;
                                        swap_count++;
                              }
                    }
                    if(swap_count==0)
                    {
                              break;
                    }
          }


}
```

TC:-

Comp's + Swap's

n = 7



	i	comp's	swap's	
first pass P1 →	0	6 ✓	0	No swap
P2 →	1	5 x	0	
P3	2	4 x	0	
	3	3 x	0	
	4	2 x	0	
	5	1 x	0	

✓ Best case : O(n) ✓

✓ WC : $O(n^2)$ ✓

H/W (GATE)

An array contains four occurrences of 0, five occurrences of 1, and three occurrences of 2 in any order. The array is to be sorted using swap operations (elements that are swapped need to be adjacent).

a. What is the minimum number of swaps needed to sort such an array in the worst case?

b. Give an ordering of elements in the above array so that the minimum number of swaps needed to sort the array is maximum.

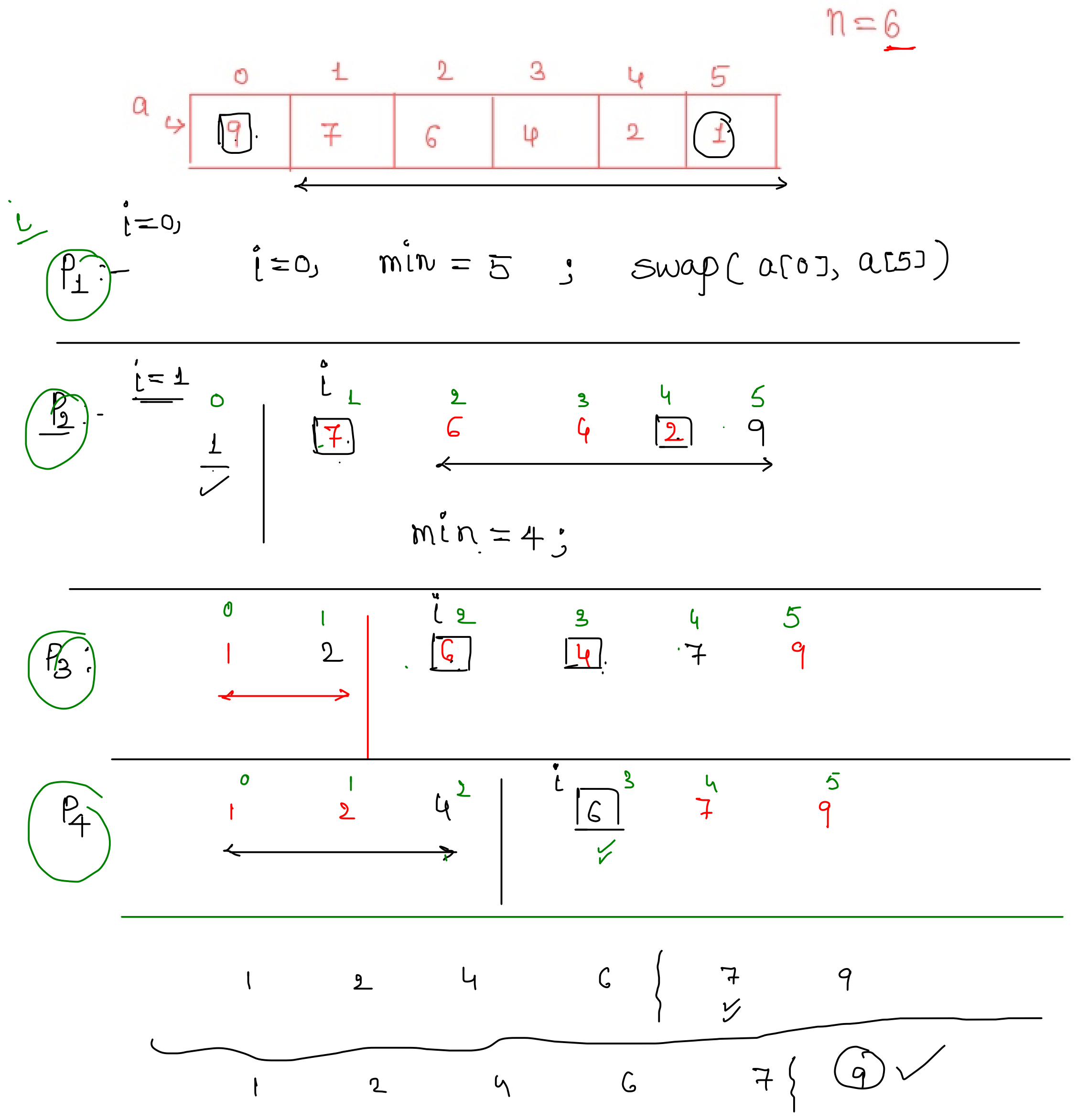
a

0	1	2	3	4	5	6	7	8	9	10	11

↑ order

selection sort

O(n)



```
int   findMin(int a[], int n)
{
    int min = a[0];
    for(i=1; i<n; i++)
    {
        if(a[i] < min)
        {
            min =a[i];
        }
    }
    return min;
}
```