Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Bubble sort

1. Compare all pairs of consecutive elements of the array one by one
2. If left element is greater than right element , then swap both
3. Repeat above steps until array is sorted

n – no. of elements

passes	comps
1	n-1
2	n-2
3	n-3
⋮	⋮
	2
n-1	1

Total comps = $1+2+3+\dots\dots+(n-2)+\underbrace{(n-1)}_{n}$

Total comps = $\frac{n(n+1)}{2}$

$= \frac{n^2+n}{2}$

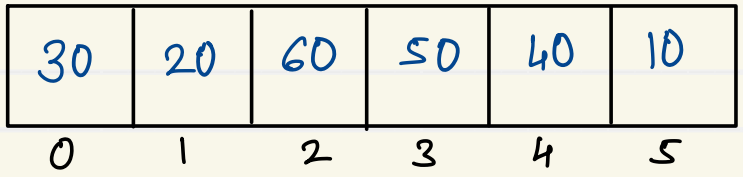
Time $\propto$ comps

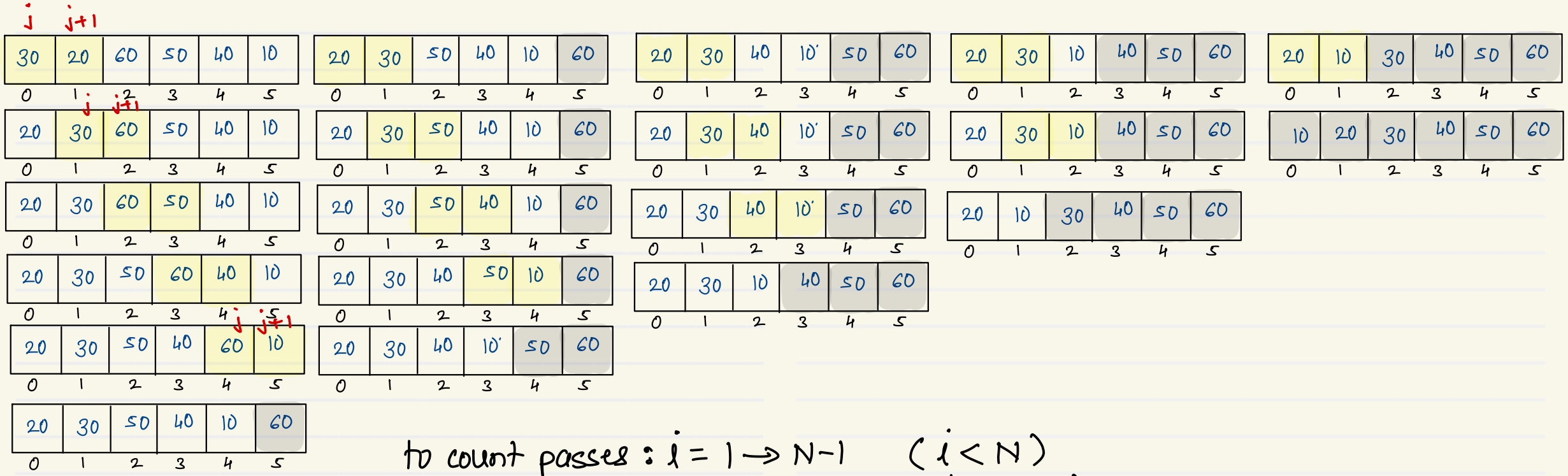
Time $\propto \frac{1}{2}(n^2+n)$

$\boxed{T(n) = O(n^2)}$ Avg Worst

$\boxed{S(n) = O(1)}$ ← in place sorting algo.

Bubble sort

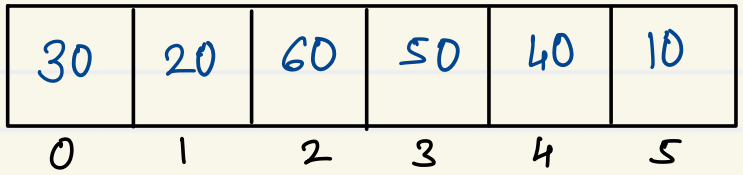


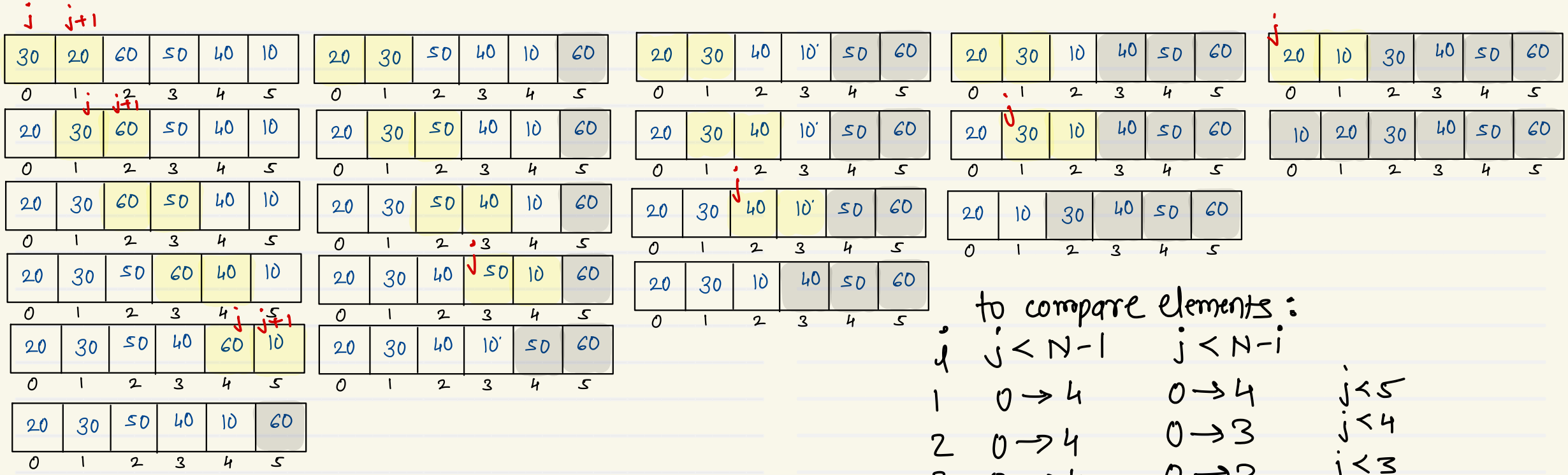


to count passes : $i = 1 \rightarrow N-1$ $(i < N)$

to compare elements : $j = 0 \rightarrow N-2$ $(j < N-1)$

Bubble sort

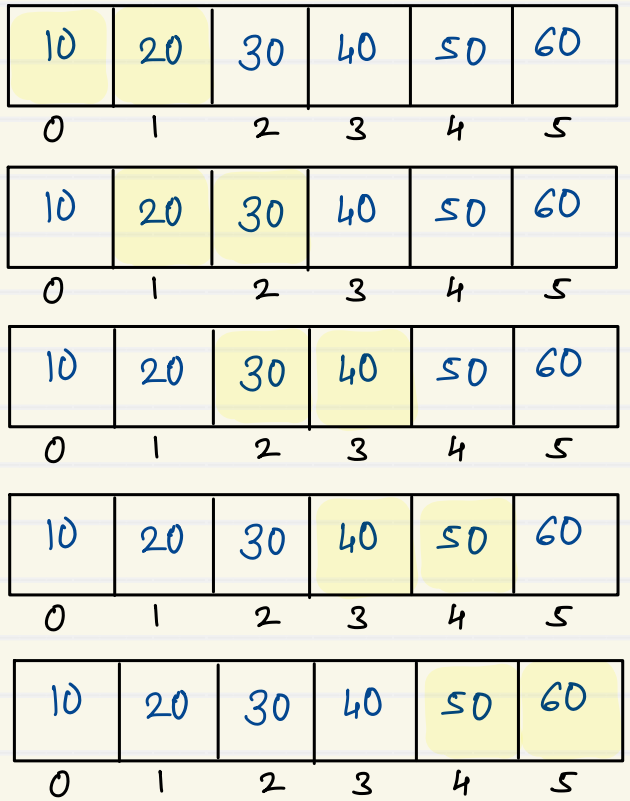




to compare elements :

i	j < N-1	j < N-i	
1	0 → 4	0 → 4	j < 5
2	0 → 4	0 → 3	j < 4
3	0 → 4	0 → 2	j < 3
4	0 → 4	0 → 1	j < 2
5	0 → 4	0 → 0	j < 1

Bubble sort




```
for(i=1; i<N; i++) {
    flag=0
    for(j=0; j<N-i; j++) {
        if(arr[j] > arr[j+1]) {
            SWAP(arr[j], arr[j+1];
            flag=1
        }
    }
    if(flag==0)
        break;
}
```

n - no. of elements

No. of comps = n-1

Time $\propto$ n-1

$T(n) = O(n)$ Best

Insertion sort

1. Pick one element of the array (start from 2nd element)
2. Compare picked element with all its left neighbours one by one
3. If left neighbour is greater, move it one position ahead
4. Insert picked element at its appropriate position
5. Repeat above steps until array is sorted (N-1)

n - no. of elements

passes	comps
1	1
2	2
3	3
:	:
n-1	n-1

Total comps = $1+2+\dots \underbrace{n-1}_{n}$

$$= \frac{n(n+1)}{2}$$

$$\text{Time} \propto \frac{1}{2}(n^2+n)$$

Avg, worst $\boxed{T(n) = O(n^2)}$

$\boxed{S(n) = O(1)}$ ← inplace sorting place

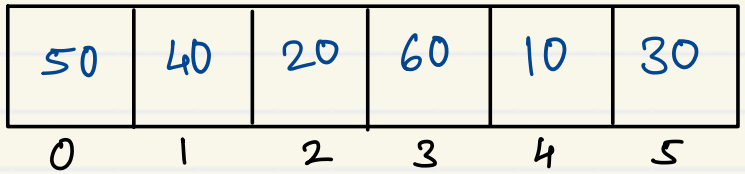
Array :	10	20	30	40	50	60
pass1 :	10	20	30	40	50	60
pass2 :	10	20	30	40	50	60
pass3 :	10	20	30	40	50	60
pass4 :	10	20	30	40	50	60
pass5 :	10	20	30	40	50	60

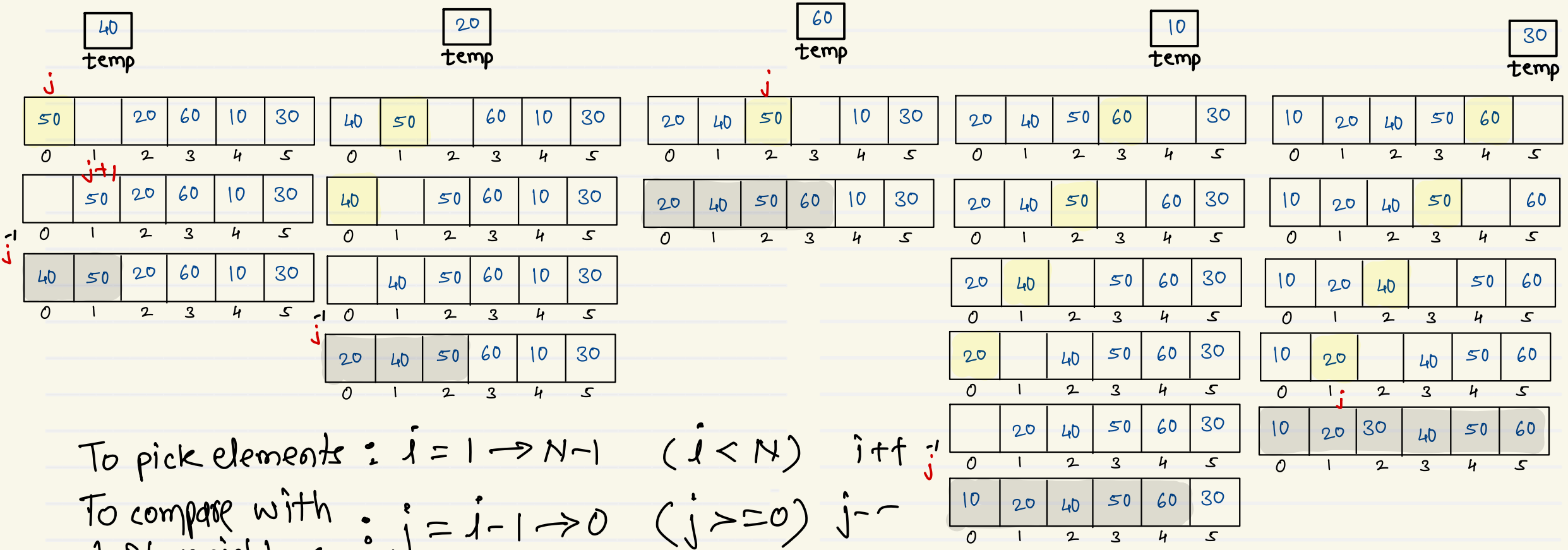
Total comps = n-1

Time $\propto$ n-1

$\boxed{T(n) = O(n)}$ Best

Insertion sort

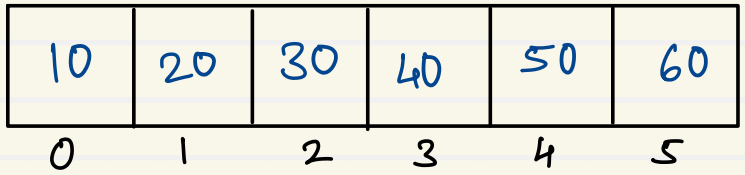




To pick elements : $i = 1 \rightarrow N-1$ $(i < N)$ $i++$

To compare with left neighbors : $j = i-1 \rightarrow 0$ $(j >= 0)$ $j--$

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



i	i<6	temp	j
1	T	40	0, -1
2	T	20	1, 0, -1
3	T	60	2
4	T	10	3, 2, 1, 0, -1
5	T	30	4, 3, 2, 1

Stable vs Unstable

Array : 22 11_a 44 11_b 33

After sorting

Array : 11_a 11_b 22 33 44 → stable sort (bubble, insertion, merge)

Array : 11_b 11_a 22 33 44 → unstable sort (selection, quick, heap)

Missing Number

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Example 1:

Input: nums = [3,0,1]

Output: 2

Example 2:

Input: nums = [0,1]

Output: 2

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]

Output: 8

1. find sum of n numbers
2. find sum of array element
3. find diff of both ← missing number

```
int missingNumber(int nums[]) {
    int n = nums.length;
    int nSum = n * (n+1) / 2;
    int numsSum = 0;
    for(int i=0; i<n; i++)
        numsSum += nums[i];

    return nSum - numsSum;
}
```

Find smallest letter greater than target

You are given an array of characters letters that is sorted in non-decreasing order, and a character target. There are at least two different characters in letters.

Return the smallest character in letters that is lexicographically greater than target. If such a character does not exist, return the first character in letters.

Example 1:

Input: letters = ["c","f","j"], target = "a"

Output: "c"

Example 2:

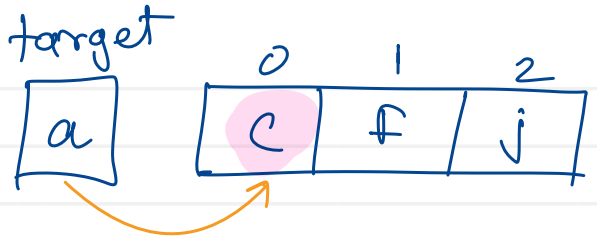
Input: letters = ["c","f","j"], target = "c"

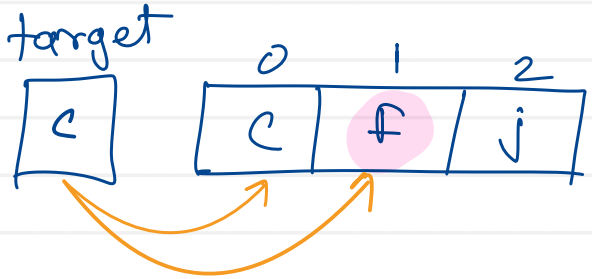
Output: "f"

Example 3:

Input: letters = ["x","x","y","y"], target = "z"

Output: "x"






```
char nextGreatestLetter(char [] letters, char target){
    int n = letters.length;
    for(i=0; i<n; i++){
        if(target < letters[i])
            return letters[i];
    }
    return letters[0];
}
```

Time Complexity

$T(n) = O(n)$

Remove Duplicates from sorted array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Example 1:

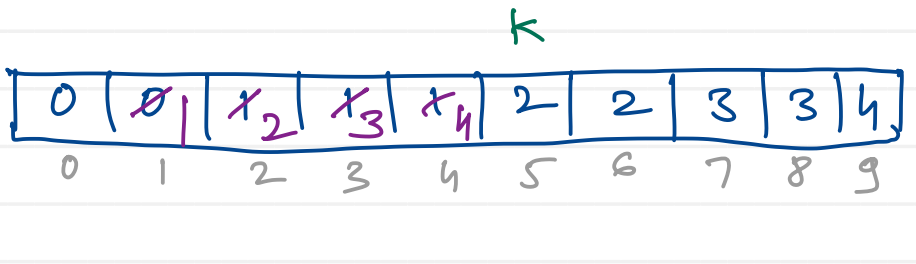
Input: nums = [1,1,2]

Output: 2, nums = [1,2,_]

Example 2:

Input: nums = [0,0,1,1,1,2,2,3,3,4]

Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]




```
int removeDuplicates(int[] nums) {
    int n = nums.length;
    int k = 1;
    for(int i = 1; i < n; i++) {
        if(nums[i] != nums[k-1])
            nums[k++] = nums[i];
    }
    return k;
}
```