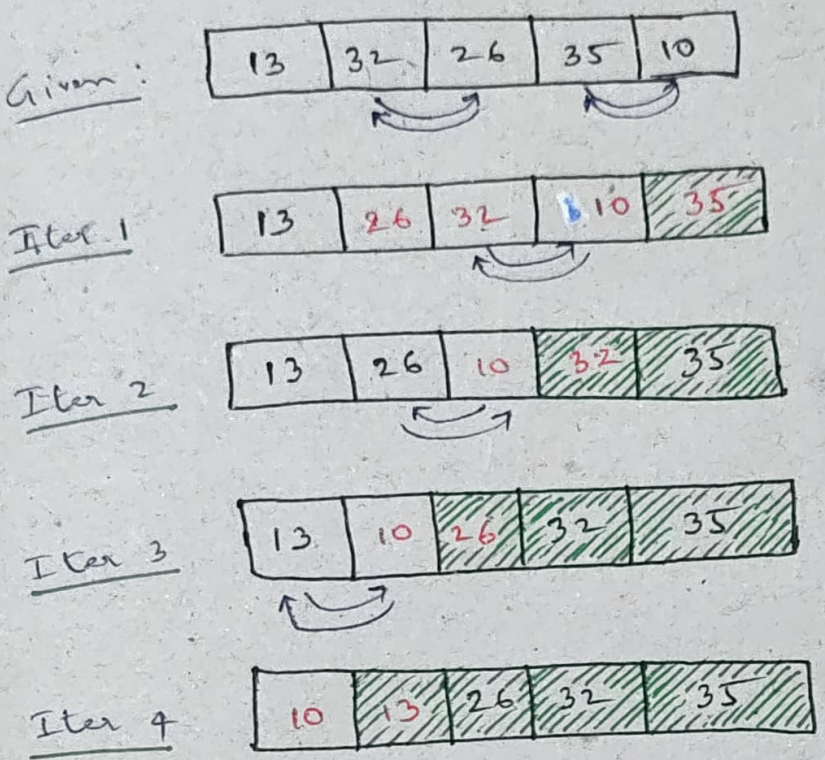
SORTING

- Arranging elements of an array in Ascending / Descending order

Bubble Sort (Ascending)



* Compare Adjacent elements
* n elements
  ↓
  n-1 Iterations

Space Complexity : O(1)

Time Complexity : O(n²)

(i) (n-1) × (n-1) ≈ n²

Best Case :

If Sorted array is given as input, in the first traversal, there wont be any swapping.
And a flag variable, will not be incremented, so, can be returned in the 1st iteration itself.

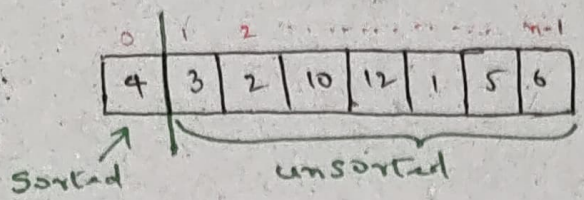
Time Complexity : O(n)

▨ — SORTED ALREADY

```
class Solution {
public:
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;
            }
        }
    }
};
```

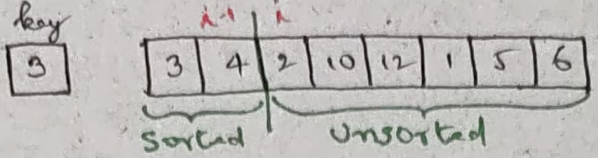
Insertion Sort (Ascending) n = 8 (i) 0 to 7 indices

Given:

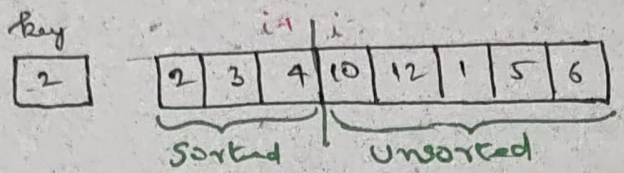


* n elements → (n-1) iterations (ASSUMPTION)

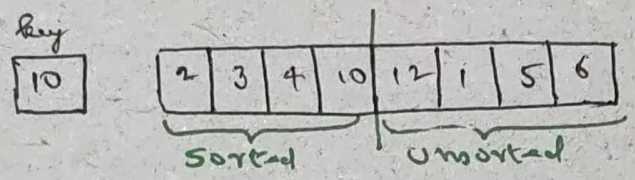
Iter 1



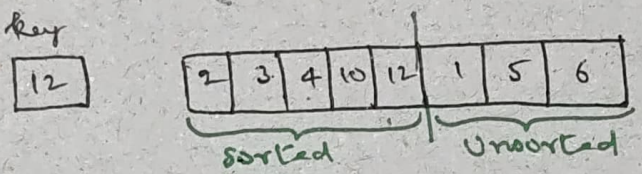
Iter 2



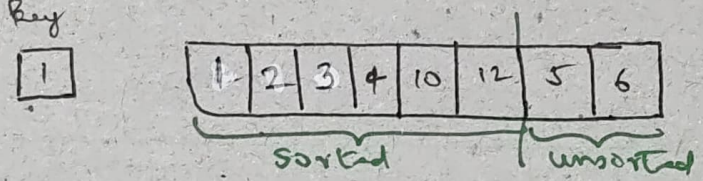
Iter 3



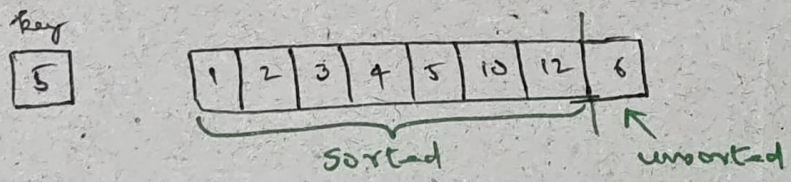
Iter 4



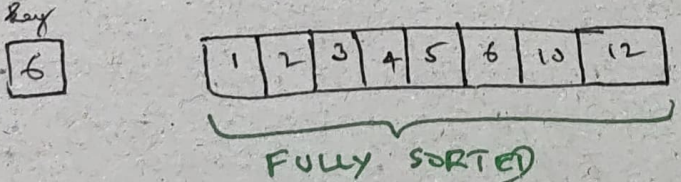
Iter 5



Iter 6



Iter 7



Analogy:

* Similar to how we sort a hand of playing cards.
* We pick up a card, and insert it into the correct position within the already sorted portion.

Best case Time Complexity O(n)

When sorted Array is given as input

Worst case Time Complexity O(n²)

Input → Descending order

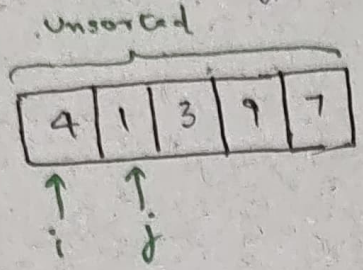
Sorting → To be done in Ascending order

Space Complexity: O(1)

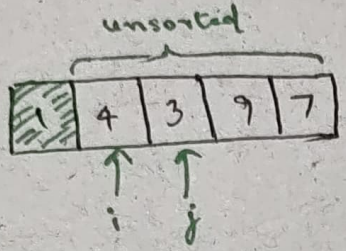
```
class Solution {
public:
    void insertionSort(int arr[], int n) {
        for(int i=1; i<n; i++) {
            int key = arr[i];
            int j = i-1;
            while (j>=0 && arr[j] > key) {
                arr[j+1] = arr[j];
                j--;
            }
            arr[j+1] = key;
        }
    }
};

int main() {
    Solution sol;
    int arr[] = {4, 3, 2, 10, 12, 1, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    sol.insertionSort(arr, n);
    for(int i=0; i<n; i++)
        printf("%d ", arr[i]);
}
```

Selection Sort (Ascending) n = 5

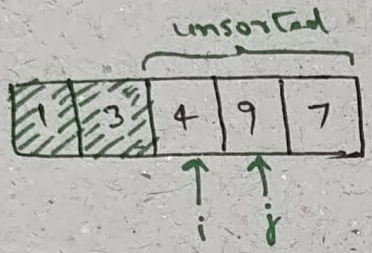
Given:



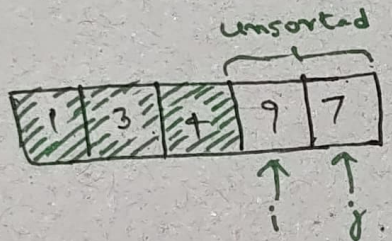
Iter 1



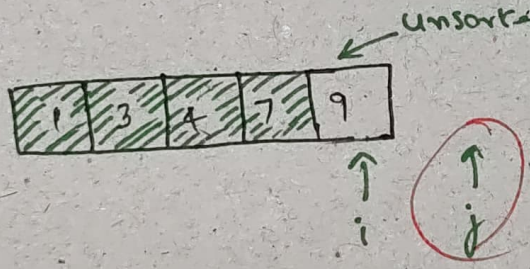
Iter 2



Iter 3



Iter 4



* It sorts an array by repeatedly selecting the smallest element from the unsorted portion and swapping it with the first unsorted element.
* This process continues until the entire array is sorted.

```
class Solution {
    public:
    void selectionSort (int arr[], int n) {
        for (int i=0; i<n-1; i++) {
            int temp = i;
            for (int j=i+1; j<n; j++) {
                if (arr[j] < arr[temp])
                    temp = j;
            }
            swap (arr[i], arr[temp]);
        }
    }
};
```