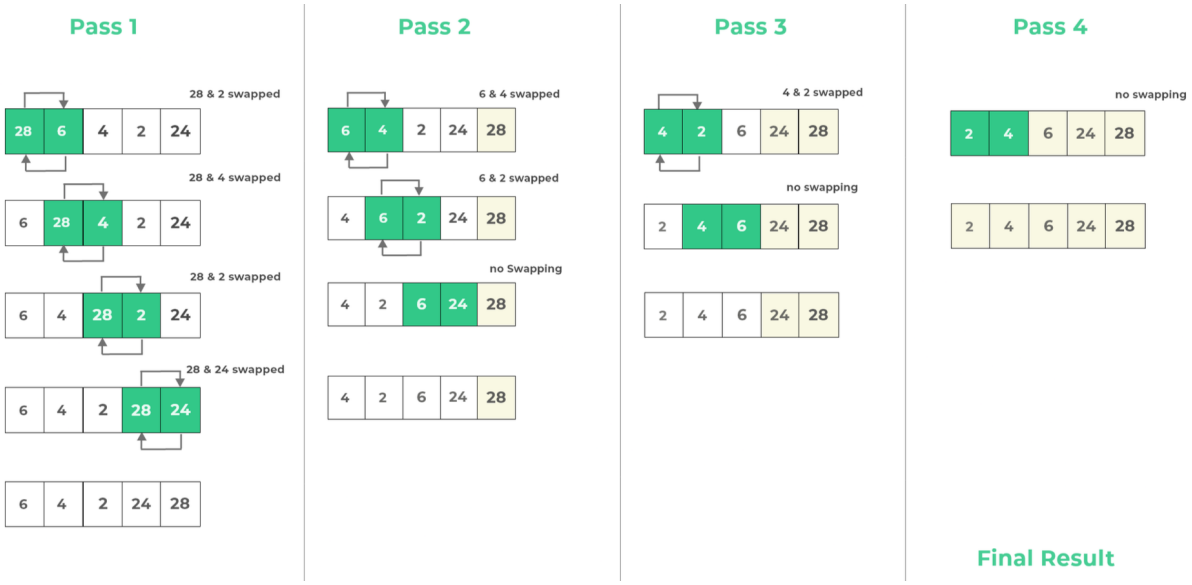
SORTING

▼ 1. Bubble Sort

A sorting algorithm in which we iterate through the array and swap the element with its adjacent element if they are not in ascending or descending order.




```
#include <stdio.h>


void bubbleSort(int arr[],int n){
    printf("\n Bubble Sorting Starting \n");


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

```
    }
}


void printArr(int arr[], int n){
    for(int i = 0; i<n; i++){
        printf(" %d ",arr[i]);
    }
    printf("\n");
}


int main() {
    int arr[] = {-2, 45, 0, 11, -9};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Initially array is: \n");
    printArr(arr,n);

    bubbleSort(arr,n);

    printf("Array after sorting:\n");
    printArr(arr,n);

    return 0;
}
```

Time Complexity

- Best - O(n)
- Worst - O(n^2)
- Average - O(n^2)

- T(n) = O(n) (print) + O(n^2) (Bubble Sort) + c

  T(n) = O(n^2)

Space Complexity

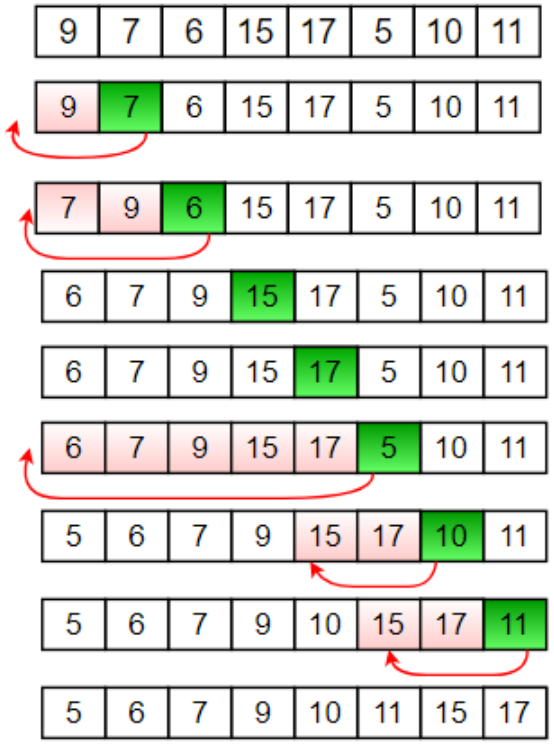
- O(1)

▼ 2. Insertion Sort

In this sorting algorithm, we assume the 1st element is a part of the sorted sub-array

Then we move to the next element, compare it with the sorted array, and insert it in the correct position.

We keep on repeating this step until the entire array is sorted.




```
#include <stdio.h>

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

```
      arr[j + 1] = arr[j];
      j--;
    }
    arr[j + 1] = key;
  }
}


void printArr(int arr[], int n){
    for(int i = 0; i<n; i++){
        printf(" %d ",arr[i]);
    }
    printf("\n");
}


int main() {
  int arr[] = {-2, 45, 0, 11, -9};
  int n = sizeof(arr) / sizeof(arr[0]);

  printf("Initially array is: \n");
  printArr(arr,n);

  insertionSort(arr,n);

  printf("Array after sorting:\n");
  printArr(arr,n);

  return 0;
}
```

Time Complexity

- Best - O(n)
- Worst - O(n^2)
- Average - O(n^2)

- T(n) = O(n) (print) + O(n^2) (Insertion Sort) + c

  T(n) = O(n^2)

Space Complexity

- O(1)

▼ 3. Selection Sort
▼ 4. Merge Sort
▼ 5. Quick Sort
▼ 6. Counting Sort
▼ 7. Radix Sort
▼ 8. Bucket Sort
▼ 9. Heap Sort
▼ 10. Shell Sort