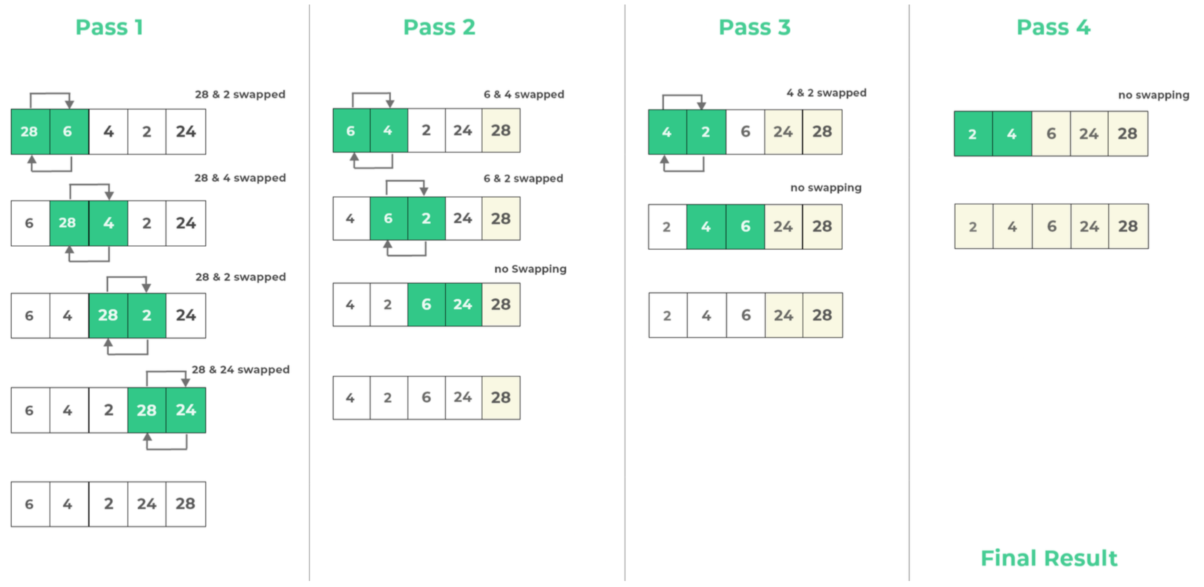
SORTING

▼ 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

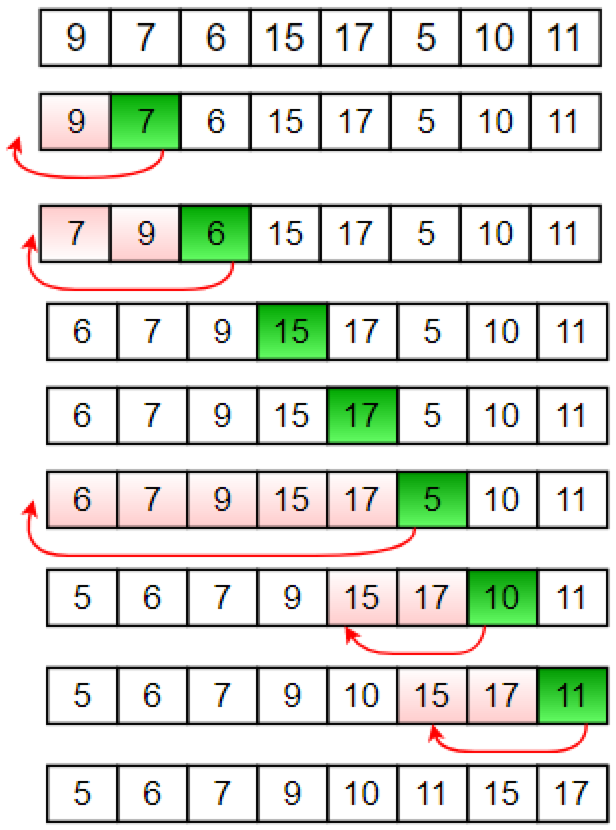
- O(1)

▼ 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

Divide and conquer approach.

Dividing the entire problem into small problems and then start solving the small problem

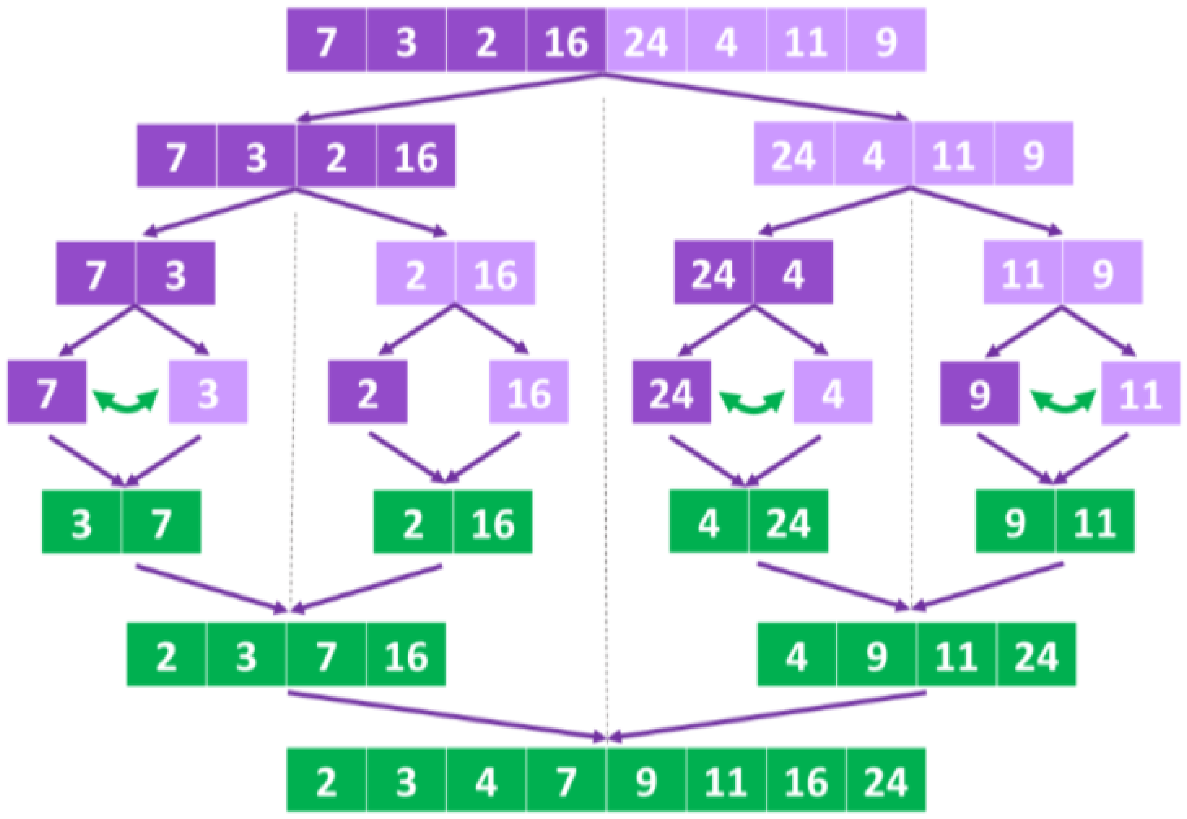
Once your done splitting the arrays into individual sub-arrays, we have to merge them back together. How do we do that?

In the below example we 1st compare 7, 3 and 2,16, which is simple.

Now how do we compare the 2?

1. We keep a counter(i) at 3 and another counter(j) at 2
2. Now we compare i(3) with j(2), as j<i, 2 is added into the bigger array and then increment j by 1
3. So now we again compare i(3) and j(16), as j>i, 3 is added into the bigger array and i is incremented by 1
4. Now we again compare i(7) and j(16), as j>i, 7 is added into the bigger array and i is incremented by 1
5. Finally as j(16) is the only element left, it is added to the bigger array

We repeat the above steps multiple times until we have merged the entire array.



```
#include <stdio.h>

void merge(int array[], int l, int m, int r) {
  int i, j, k;
  int n1 = m - l + 1; // size of left array
  int n2 = r - m; // size of right array

    // creating temp arrays
  int L[n1], R[n2];

  // Copy data to temp arrays L[] and R[]
  for (i = 0; i < n1; i++)
    L[i] = array[l + i];
  for (j = 0; j < n2; j++)
    R[j] = array[m + 1 + j];
  i = 0;
  j = 0;
  k = l;
  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      array[k] = L[i];
      i++;
    } else {
      array[k] = R[j];
      j++;
    }
    k++;
  }

  // Copy the remaining elements of L[], if there are any
  while (i < n1) {
    array[k] = L[i];
    i++;
    k++;
  }

  // Copy the remaining elements of R[], if there are any
  while (j < n2) {
    array[k] = R[j];
    j++;
    k++;
  }
}


void mergeSort(int array[], int l, int r) {

  if(l==r){
      return;
     }
  if (l < r) {
    int m = l + (r - l) / 2;

    mergeSort(array, l, m);
    mergeSort(array, m + 1, r);

        // merging 2 sorted sub-arrays
    merge(array, l, m, r);
  }
}


void printArray(int *array, int size) {
  for (int i = 0; i < size; i++)
    printf("%d ", array[i]);
  printf("\n");
}


int main() {
  int array[] = {7,3, 2, 16, 24, 4, 11, 9};
  int arr_size = sizeof(array) / sizeof(array[0]);

  printf("Given array is \n");
  printArray(array, arr_size);

  mergeSort(array, 0, arr_size - 1);

  printf("\nSorted array is \n");
  printArray(array, arr_size);
  return 0;
}
```

Time Complexity

- O(nlogn)

Space Complexity

- O(n)

▼ 5. Quick Sort

▼ 6. Counting Sort

▼ 7. Radix Sort

▼ 8. Bucket Sort

▼ 9. Heap Sort

▼ 10. Shell Sort