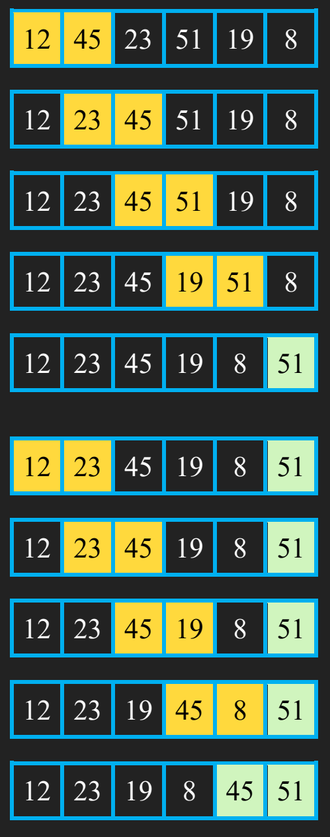
Bubble, Merge, Selection & Insertion Sort

Properties		Bubble Sort	Merge Sort	Selection Sort	Insertion Sort
Definition		Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.	Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.	Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.	Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.
Time Complexity	**Best**	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
	Average	$O(n^2)$	$O(n \log n)$	$O(n^2)$	$O(n^2)$
	Worst	$O(n^2)$	$O(n \log n)$	$O(n^2)$	$O(n^2)$
Space Complexity		Normal Bubble Sort: $O(1)$ Optimized Bubble Sort: $O(2)$	$O(n)$	$O(1)$	$O(1)$
Stability A sorting algorithm is considered stable if the two or more items with the same value maintain the same relative positions even after sorting		Yes	Yes	No	Yes
Applications		Bubble sort is used if: • Complexity does not matter • Short and simple code is preferred	Merge Sort Applications: • Inversion count problem • External sorting • E-commerce applications	The selection sort is used when: • A small list is to be sorted • Cost of swapping does not matter • Checking of all the elements is compulsory • Cost of writing to memory matters like in flash memory	The insertion sort is used when: • The array is having a small number of elements • There are only a few elements left to be sorted

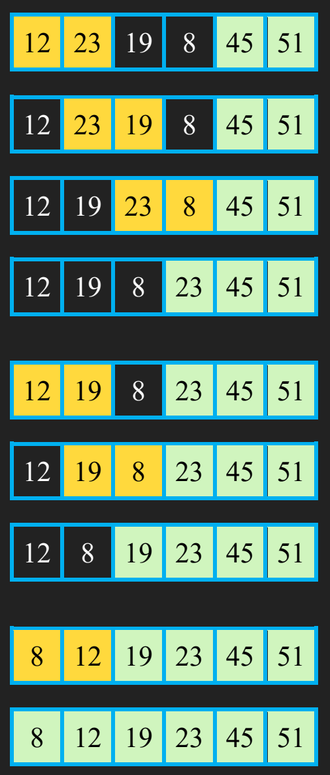
Visualization

1.



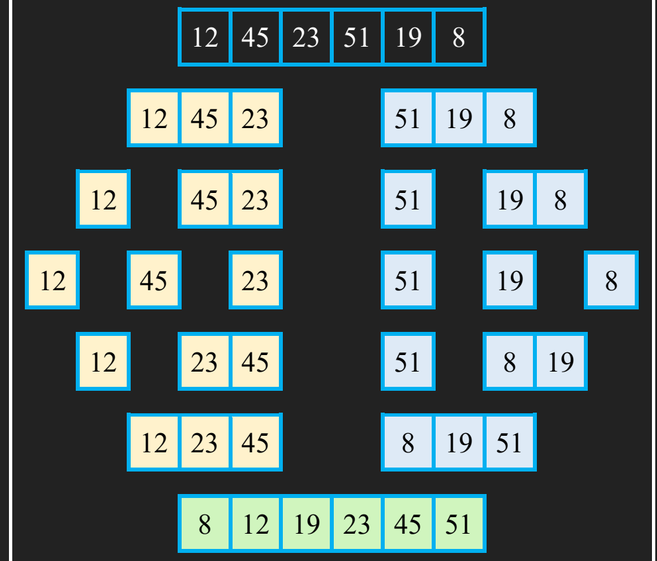
2.

3.

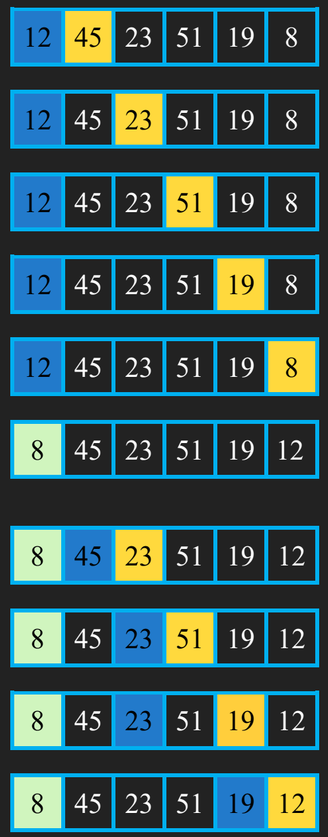


4.

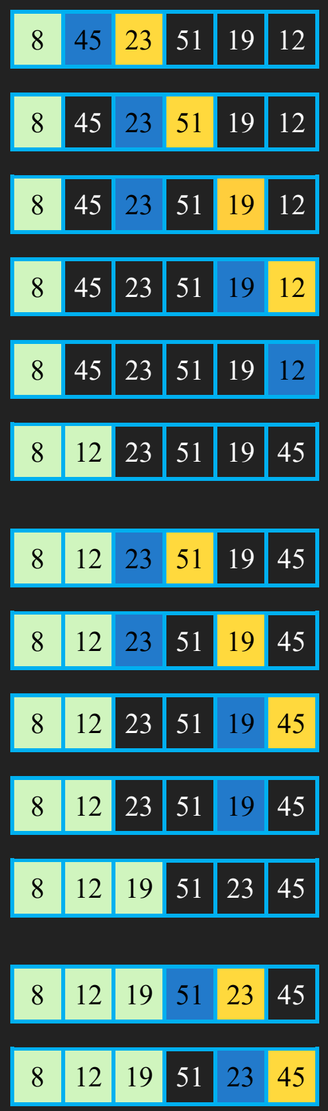
5.



1.

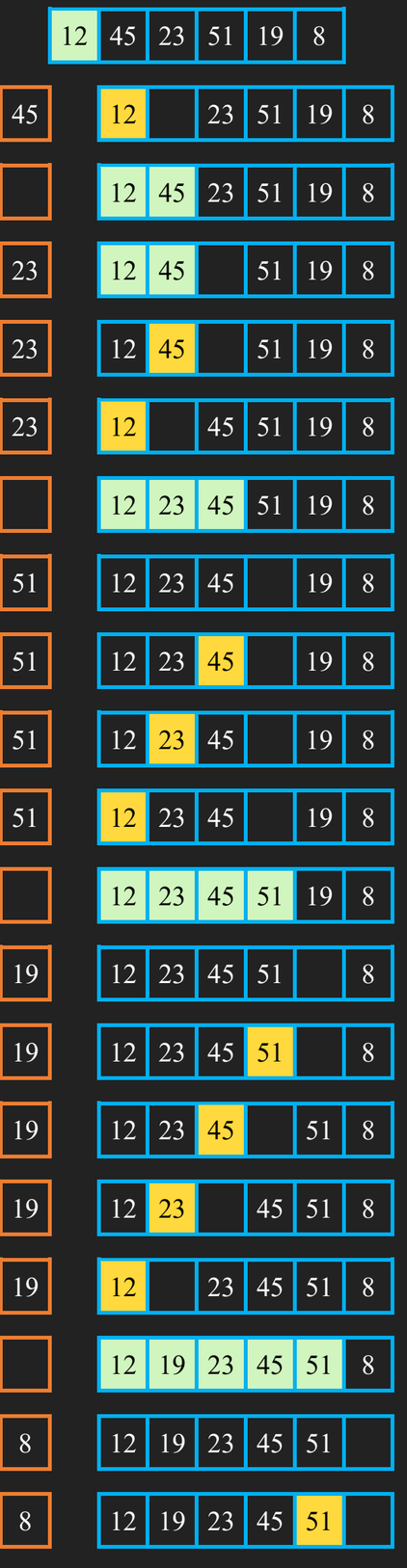


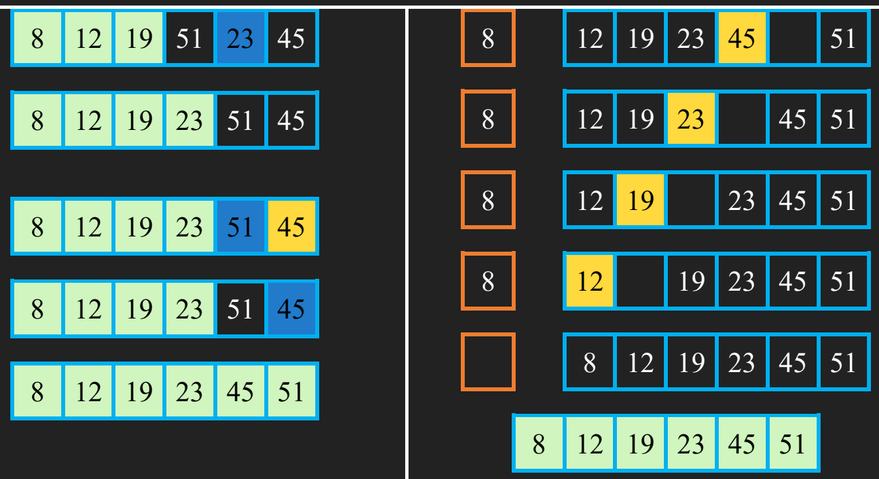
2.



3.

4.





Code Implementation

```c
#include <stdio.h>
#include <stdbool.h>

void bubble_sort(int array[], int array_length);
void swap(int *a, int *b);

int main()
{
  int array_length, index;
  printf("Enter the length of the array: ");
  scanf("%d", &array_length);
  int array[array_length];
  printf("Enter the elements of the array: ");
  for (index = 0; index < array_length; index++)
  {
    scanf("%d", &array[index]);
  }
```

```c
#include <stdio.h>

void merge(int array[], int left, int middle, int right);
void merge_sort(int array[], int left, int right);

int main()
{
  int array_length, index;
  printf("Enter the length of the array: ");
  scanf("%d", &array_length);
  int array[array_length];
  printf("Enter the elements of the array: ");
  for (index = 0; index < array_length; index++)
  {
    scanf("%d", &array[index]);
  }

  merge_sort(array, 0, array_length - 1);

  printf("The sorted array is: ");
```

```c
#include <stdio.h>

void selection_sort(int array[], int array_length);
void swap(int *a, int *b);

int main()
{
  int array_length, index;
  printf("Enter the length of the array: ");
  scanf("%d", &array_length);
  int array[array_length];
  printf("Enter the elements of the array: ");
  for (index = 0; index < array_length; index++)
  {
    scanf("%d", &array[index]);
  }
  selection_sort(array, array_length);
```

```c
#include <stdio.h>

void insertion_sort(int array[], int array_length);

int main()
{
  int array_length, index;
  printf("Enter the length of the array: ");
  scanf("%d", &array_length);
  int array[array_length];
  printf("Enter the elements of the array: ");
  for (index = 0; index < array_length; index++)
  {
    scanf("%d", &array[index]);
  }
  insertion_sort(array, array_length);
  return 0;
}
```

```
  bubble_sort(array, array_length);
  return 0;
}

void bubble_sort(int array[], int array_length)
{
  int pass, index;
  bool flag;
  for (pass = 0; pass < array_length - 1; pass++)
  {
    flag = false;
    for (index = 0; index < array_length - pass - 1; index++)
    {
      if (array[index] > array[index + 1])
      {
        swap(&array[index], &array[index + 1]);
        flag = true;
      }
    }
    if (flag == false)
    {
      break;
    }
  }
  printf("The sorted array is: ");
  for (index = 0; index < array_length; index++)
  {
    printf("%d ", array[index]);
  }
  printf("\n");
}

void swap(int *a, int *b)
{
  *a = *a ^ *b;
  *b = *a ^ *b;
  *a = *a ^ *b;
}
```

```
  for (index = 0; index < array_length; index++)
  {
    printf("%d ", array[index]);
  }
  printf("\n");
  return 0;
}

void merge(int array[], int left, int middle, int right)
{
  int left_size = middle - left + 1;
  int right_size = right - middle;

  int Left[left_size], Right[right_size];
  int i, j, k;

  // Copy data to temp arrays Left[] and Right[]
  for (i = 0; i < left_size; i++)
  {
    Left[i] = array[left + i];
  }
  for (j = 0; j < right_size; j++)
  {
    Right[j] = array[middle + 1 + j];
  }

  // Merge the temp arrays back into array[left...right]
  i = 0;
  j = 0;
  k = left;

  while (i < left_size && j < right_size)
  {
    if (Left[i] <= Right[j])
    {
      array[k] = Left[i];
      i++;
    }
    else
    {
      array[k] = Right[j];
      j++;
    }
    k++;
  }

  // Copy the remaining elements of Left[], if any
```

```
  return 0;
}

void selection_sort(int array[], int array_length)
{
  int index, pass, min_index;
  for (pass = 0; pass < array_length - 1; pass++)
  {
    min_index = pass;
    for (index = pass + 1; index < array_length; index++)
    {
      if (array[index] < array[min_index])
      {
        min_index = index;
      }
    }
    if (min_index != pass)
    {
      swap(&array[pass], &array[min_index]);
    }
  }
  printf("The sorted array is: ");
  for (index = 0; index < array_length; index++)
  {
    printf("%d ", array[index]);
  }
  printf("\n");
}

void swap(int *a, int *b)
{
  *a = *a ^ *b;
  *b = *a ^ *b;
  *a = *a ^ *b;
}
```

```
void insertion_sort(int array[], int array_length)
{
  int unsorted_array_index, sorted_array_index, temp;
  for (unsorted_array_index = 1; unsorted_array_index < array_length; unsorted_array_index++)
  {
    temp = array[unsorted_array_index];
    sorted_array_index = unsorted_array_index - 1;
    while (sorted_array_index >= 0 && array[sorted_array_index] > temp)
    {
      array[sorted_array_index + 1] = array[sorted_array_index];
      sorted_array_index--;
    }
    array[sorted_array_index + 1] = temp;
  }
  printf("The sorted array is: ");
  for (sorted_array_index = 0; sorted_array_index < array_length; sorted_array_index++)
  {
    printf("%d ", array[sorted_array_index]);
  }
  printf("\n");
}
```

```
  while (i < left_size)
  {
    array[k] = Left[i];
    i++;
    k++;
  }

  // Copy the remaining elements of Right[], if any
  while (j < right_size)
  {
    array[k] = Right[j];
    j++;
    k++;
  }
}

void merge_sort(int array[], int left, int right)
{
  if (left < right)
  {
    int middle = left + (right - left) / 2;

    // Sort first and second halves
    merge_sort(array, left, middle);
    merge_sort(array, middle + 1, right);

    // Merge the sorted halves
    merge(array, left, middle, right);
  }
}
```