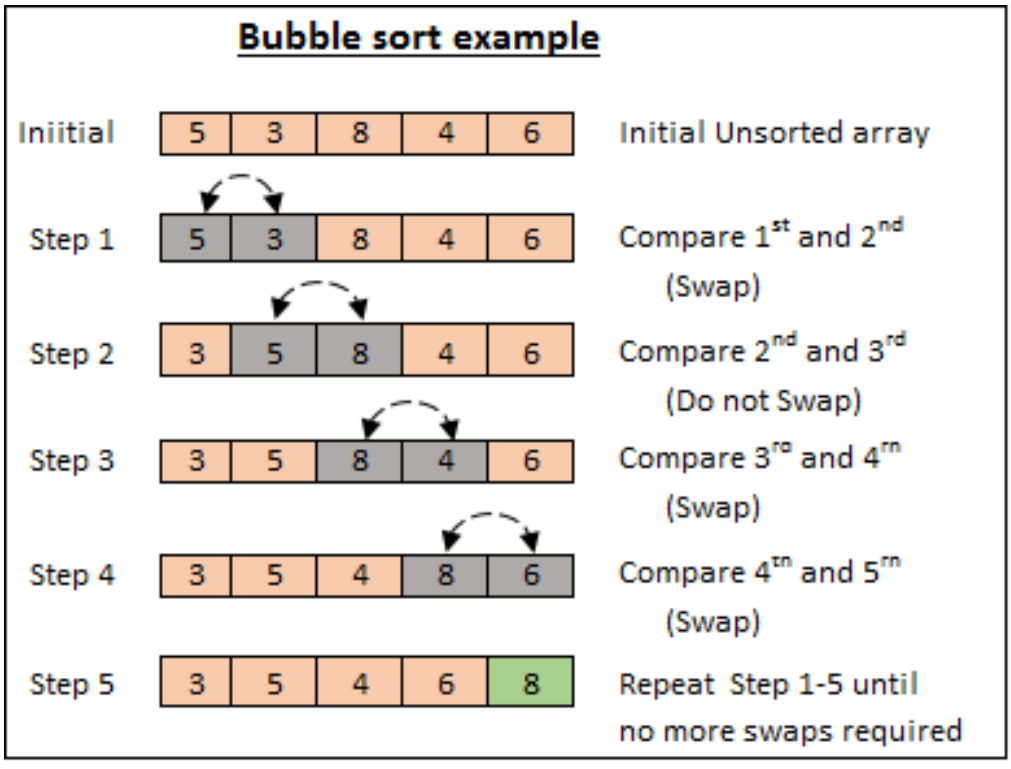
Bubble Sort:

خب این الگوریتم چیز ساده ای بنظر میاد که منطقش اینه تک تک اعضا با بغلیشون مقایسه میکنه اگر مقدار بیشتری نسبت به اون داشت جاش عوض میکنه

بر فرض ما مثل عکس پایین دارا ۵ عضو باشیم که باید مرتب بشوند

درواقق شما چند مرحله داری که در هر مرحله ما میایم تک تک از اول اعضا بررسی میکنیم و این مراحل تا جایی ادامه دارن که دیگه مرتب شده باشه البته بعضی مواقع ما چند تا مرحله میریم مثلا یکی مونده به اخرین مرحله سورت شده لیستمون ولی طبق الگوریتم باید یه مرحله دیگه هم بره

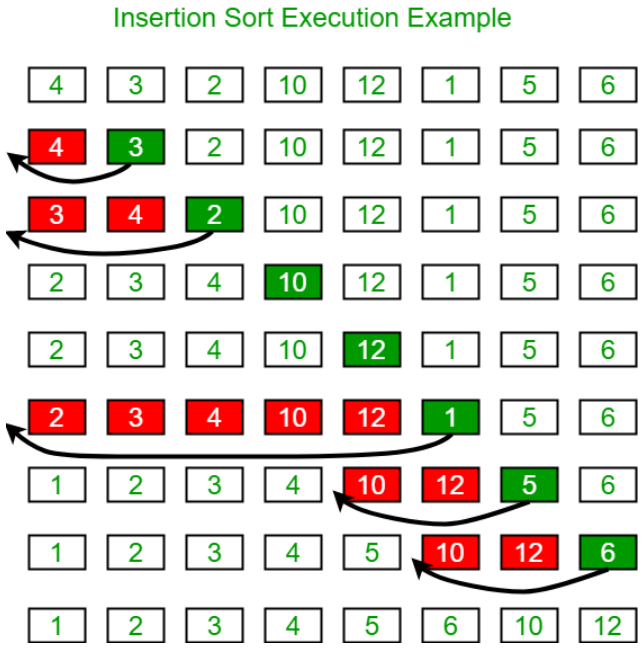
```
void Sort::Bubble(int arr[], int size) {
    for (int k = 0; k < size; k++)
        for (int i = 0; i < size -k -1; i++)
            if (arr[i] > arr[i +1]){
                temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
      }
     display();
}
```



Insertion sort:

در این روش شما از ایندکس شماره ۱ شماره میکنی دقت کنید صفر نگفتم تا اخرین ایندکس لیست
سپس به ترتیب ایندکس یک برمیداری با اعضا قبلش تک تک بررسی میکنی اگر کوچک تر بود از اونا میره جای اونا اما اگر بزرگ بود میریم سراغ ایندکس بعدی و همین کار باهاش میکنیم

```
void Sort::Insertion(int arr[],int size){
    int temp;
    for (int i = 1; i < size; i++)
    {
        temp = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > temp){
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = temp;
    }
    display();

}
```

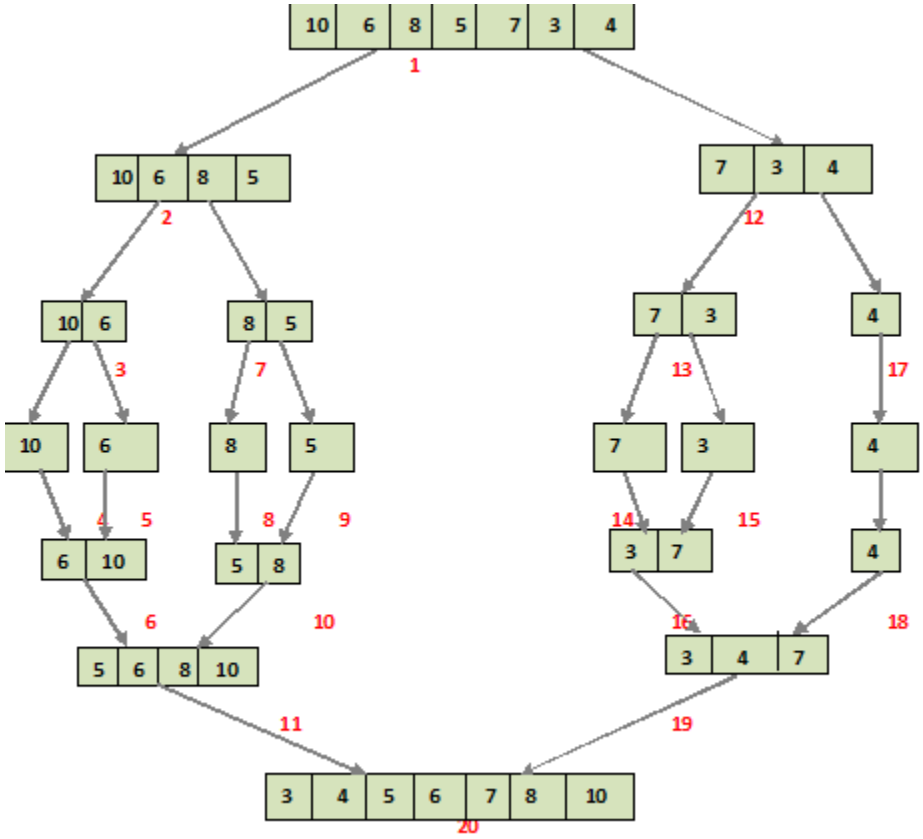


Merge Sort:

ببنید این الگوریتم بخوام توضیح بدیم از سه قسمت تشکیل شده و اون چیزی نیست جز
تقسیم
محاسبه
ادغام

خب اینا چی هستن؟ مرحله تقسیم یعنی اینکه میاد ارایه کلی که داریم به دو قسمت تقسیم میکنه و هر کدوم از اون قسمت باز تقسیم میشوند تا جایی که عنصر میانی وجود نداشته باشه برای نصف کردن (تک عضوی باشه)

سپس میاد با هم دیگر مقایسه میکنه و در اخر میاد همه اینارو باهم ادغام میکنه

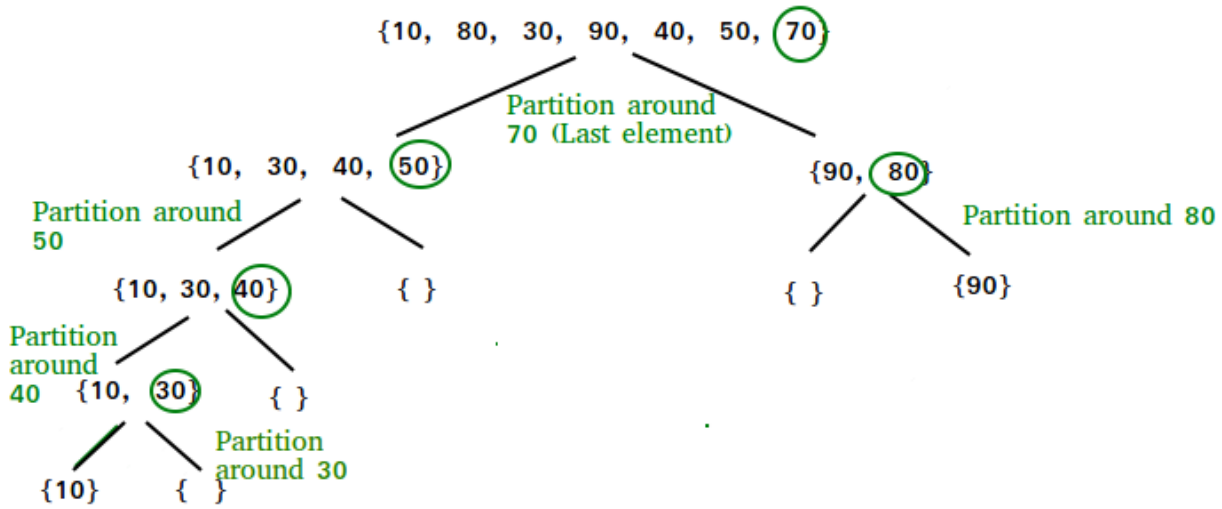


```cpp
void Sort::merge(int *array, int  left, int  mid, int  right)
{
    int  subArrayOne = mid - left + 1;
    int  subArrayTwo = right - mid;

    int *leftArray = new int[subArrayOne],
         *rightArray = new int[subArrayTwo];


    for (int i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (int j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    int indexOfSubArrayOne = 0,
        indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;


    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}
void Sort::mergeSort(int *array, int  begin, int  end)
{
    if (begin >= end)
        return; // Returns recursively

    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);

}
```

در این روش یک عنصر محوری انتخاب میشه که میتونه اول لیست باشه یا اخرش یا وسطش

تو مرج سورت ما وسطی انتخاب میکردیم

حالا تو این کدی که زدم من عنصر محوری اخرین عنصر در نظر گرفتم




```
{arr[] = {10, 80, 30, 90, 40, 50, 70
Indexes:  0   1   2   3   4   5   6
low = 0, high =  6, pivot = arr[h] = 70
Initialize index of smaller element, i = -1
Traverse elements from j = low to high-1
([j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j
i = 0
arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j
are same //
j = 1 : Since arr[j] > pivot, do nothing
[]No change in i and arr //
([j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j
i = 1
arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30
j = 3 : Since arr[j] > pivot, do nothing
[]No change in i and arr //
([j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j
i = 2
arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped
[j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j
i = 3
arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped
.We come out of loop because j is now equal to high-1
Finally we place pivot at correct position by swapping
(arr[i+1] and arr[high] (or pivot
```

```
arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped
Now 70 is at its correct place. All elements smaller than
are before it and all elements greater than 70 are after 70
.it
```

```
143 void Sort::swap(int* a, int* b)
144 {
145     int t = *a;
146     *a = *b;
147     *b = t;
148 }
149
150 int Sort::partition (int *arr, int low, int high)
151 {
152     int pivot = arr[high];
153     int i = (low - 1);
154
155     for (int j = low; j <= high - 1; j++)
156     {
157
158         if (arr[j] < pivot)
159         {
160             i++;
161             swap(&arr[i], &arr[j]);
162
163     }
164     swap(&arr[i + 1], &arr[high]);
165     return (i + 1);
166 }
167
168 void Sort::quickSort(int *arr, int low, int high)
169 {
170     if (low < high)
171     {
172
173         int pi = partition(arr, low, high);
174
175         quickSort(arr, low, pi - 1);
176         quickSort(arr, pi + 1, high);
177     }
178
```