Algorithm to depict Merge Sort:

Step 1: Take the elements of an unsorted list (first, last).

//arr = [6, 8, 9, 3, 1, 4, 2, 7]

Step 2: If the first element is less than last element then find the mid element of the list.

// mid = (first + last) / 2

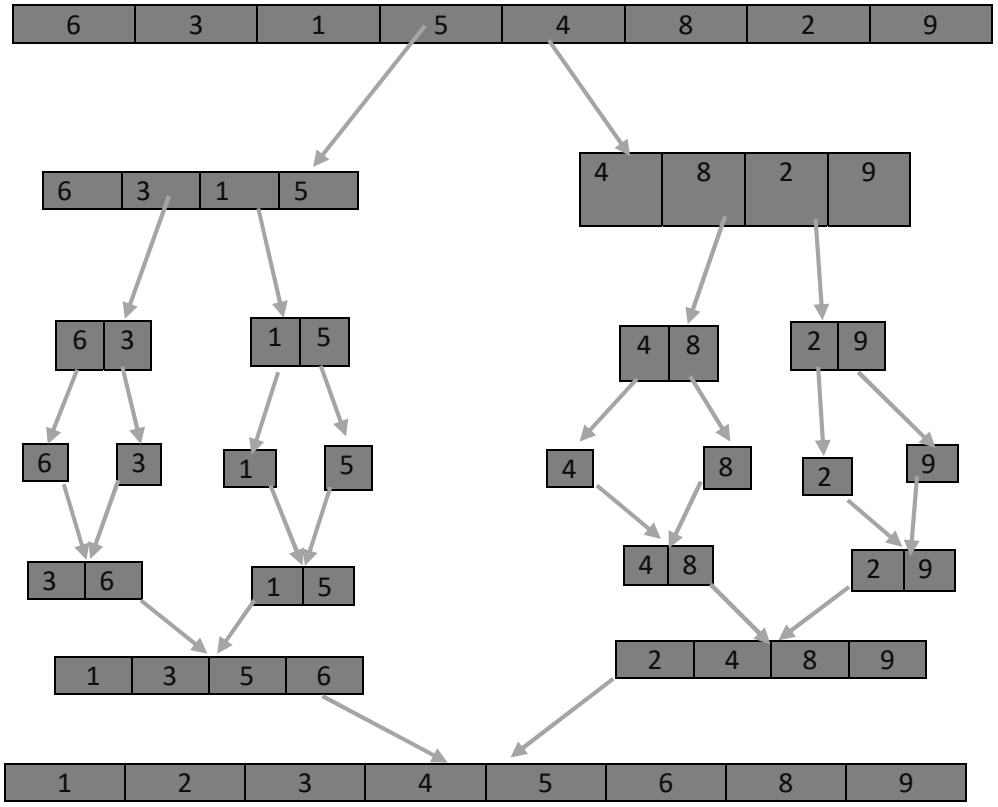
Step 3: Perform the merge sort from first to mid (first, mid).

Step 4: Perform the merge sort from mid+1 to last (mid+1, last).

Step 4: Repeat the steps 3 and 4 untill the elements gets separated.

Step 5: Perform the merge sort (first, mid, mid+1, last)

```
mergeSort(list, first, last)
{
If (first<last)
{
Mid = (first+last)/2;
mergeSort(first, mid);
mergeSort(mid+1, last);
mergeSort(first, mid, mid+1, last);
}
}
```



Algorithm to depict Binary Search:

Step 1: Take the elements of an sorted list (first, last).

//list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

Step 2: Take a Key to search elements in the list.

//Key = 7

Step 3: Find the mid element of the sorted list.

// mid = (first + last) / 2

Step 4: Compare the mid element with the key.

Step 5: If both the elements are matched display the element.

Step 6: If the both the elements are not matched check weather the element is smaller or larger than the middle element.

Step 7: If the element is smaller than the middle element then repeat the steps 3, 4, 5 and 6 for the left sub list of the element.

Step 7: If the element is greater than the middle element then repeat the steps 3, 4, 5 and 6 for the right sub list of the element.

Step 8: Repeat the process untill we find the search element in the list.

```
int binarySearch(list, n, key)
{
l=1, h=n;
while(l<=h)
{
Mid=(l+h)/2;
If(key==list(mid))
{
return mid;
if(key<mid)
return mid+1;
}
Else
{
l=mid+1;
}
return 0;
}
```