Mergesort

Basic Idea of Merge sort

- Divide the array into two halves
- Sort the two sub-arrays   (How?)
- Merge the two sorted sub-arrays into a single sorted array
- Step 2 (sorting the sub-arrays) is done recursively (divide in two, sort, merge) until the array has a single element (base condition of recursion)

Mergesort Example

- Let us take an example of an array with 8 elements
- [16 9 12 48 30 11 65 22]
- We shall split it recursively in two parts till the array contains only a single element.
- Then we shall merge those two subarrays and move to higher sub-arrays
- In this case we know that after 3 splits , the sub array will contain single elements.

- [16 9 12 48 30 11 65 22]

$8/2 = 4$

[16 9 12 48] [

Divide [16 9] [12 48]

[16] [9]

merge

[9 16]

[12] [48]

merge

[12 48]

merge [9 12 16 48]

[30 11] [65 22]

merge [30] [11]

[11 30]

[6 5] [22]

[22 65]

merge

merge [11 30] with [22 65]

[11 22 30 65]

merge [9 12 16 48] with [11 22 30 65]

[9 11 12 16 22 30 48 65]

9 elements : $n/2$ $9/2 = 4$

[16 9 12 23 6 48 90 33 40]

[16 9 12 23] [6 48 90 33 40] $5/2 = 2$

[6 48] [90 33 40] $3/2 = 1$

[90] [33 40]

[16 9 12 48 30 11 65 22]

[16 9 12 48 | 30 11 65 22]

Split 1: left :[16 9 12 48] right: [30 11 65 22]

Split 2:[16 9] [12 48]

Split 3:[16] [9] , no more splits possible

merge1 [16] [9] : [9 16]

right: [12 48]

[12] [48]

merge2 [12] [48]: [12 48]

Now both subarrays at split 2 have been sorted so we can merge these

merge3 [9 16][12 48] : [9 12 16 48]

Now take up the right half of Split 1

Take right half [30 11 65 22]

[30 11] [65 22]

merge4 [30] [11] : [11 30]

merge5 [65] [22] : [22 65]

merge6 [11 30] and [22 65]

[11 22 30 65]

merge7 [9 12 16 48] and [11 22 30 65]

[9 11 12 16 22 30 48 65]

Sorting 8 elements required 7 merge steps

Sorting n elements requires n – 1 merge steps

Use mergesort on

[23 4 89 12 45 9 78 10]

[23 4 89 12] [45 9 78 10]

[23 4] [89 12]

[23] [4]

[4 23]……. Proceed in same manner as before

Main function

```
int main( )
        {
                int a[30],n,i;
                printf("Enter no of elements:");
                scanf("%d",&n);
                printf("Enter array elements:");
                for(i=0;i<n;i++)
                        scanf("%d",&a[i]);
                mergesort(a ,0, n-1);           //sort array a from index 0 to n-1
                printf("\nSorted array is :");
                for(i=0;i<n;i++)
                        printf("%d ",a[i]);
                return 0;
}
```

Merge sort function

- Mergesort needs a helper array to merge two sorted arrays.

```
void mergesort( int a[], int i, int j )
    {
        int mid;
        if(i<j)
        {
            mid = (i+j)/2;
            mergesort(a, i, mid); //left recursion
            mergesort(a, mid+1, j); //right recursion
            merge(a, i, mid, mid+1, j); //merging of two sorted sub-arrays
        }
    }
```

Merge

- Array [a] is copied into a temporary array.

- Pointer i is set to beginning of first subarray
- Pointer j is set to beginning of second subarray
- Pointer k is set for the array which collects elements after merging.

Merge

- Array [a] is copied into a temporary array.

- Pointer i is set to beginning of first subarray
- Pointer j is set to beginning of second subarray
- Pointer k is set for the array which collects elements after merging.

- Smaller of the two elements from the sub arrays are transferred to the merged array
- If all the elements of one subarray have been put in temp array, remaining elements of other array are simply copied into temp.

- After merging is over, elements are copied back into array [a]

Merge function

```
void merge(int a[],int i1,int j1,int i2,int j2)  {
      int temp[50]; //array used for merging
      int i,j,k;
      i=i1;                                                    //beginning of the first list
      j=i2;                                                    //beginning of the second list
      k=0;
      while(i<=j1 && j<=j2) {                                  //while elements in both lists
            if(a[i]<a[j])     temp[k++] = [i++];
            else        temp[k++] = a[j++];  }
      while(i<=j1)   temp[k++] = a[i++];                       //copy remaining elements of the first list
      while(j<=j2)   temp[k++] = a[j++];                        //copy remaining elements of the second list
      for(i=i1,j=0;i<=j2;i++,j++) a[i] = temp[j];                //Transfer elements from temp[] back to a[]
}
```

Mergesort complexity

- Like binary search, array divided in two parts at every stage, so work done would be of order log n .
- At every stage each element of the array is copied and may be processed. This is bound to be of order n.
- Thus intuitively mergesort needs O(n logn) operations.