1. Description:

Merge sort follows the divide and rule policy. It divides the array of n elements into n separate arrays of size 1 (each array contains one element) and then rejoins the divided arrays one by one with each other to form the array of size n. Two variables start and end store the start and end limit of the array. The middle **(m)** position of the array is obtained by adding start and end and dividing it by 2.

Example:

|  |
| --- |
| 34 45 89 23 38 76 12 28 |

The length of the array is 8, so n=8. Thus end=0 (since position of elements in array starts from 0) and start=n-1=7.

array will be split from the middle, i.e. (0+7)/2=3 (element at 4th position). The array becomes:

[34 45 89 23] ….. [38 76 12 28]

Now each sub array will be considered an individual array and the procedure will be applied to each sub array again.

The left sub array has four elements and so, end=0 and start=3. Thus middle point occurs at 1 (position of second element).

The right sub array also has four elements and so, end=4 and start=7. Thus the middle point occurs at 5 (position of 6th element)

[34 45]… [89 23] … [38 76] …. [12 28]

This process is repeated till all sub arrays contain 1 element each.

[34] [45] [89] [23] [38] [76] [12] [28]

Then The Merging Starts:

1. [**34] [45] [89] [23] [38] [76] [12] [28]**
2. **[34 45] [23 89] [38 76] [ 12 28]**
3. **[23 34 45 89][12 28 38 76]**
4. **[12 23 28 34 38 45 76 89]**

The single array obtained at the end is the sorted array.

1. Pseudo Code

* **Merge Sort**

// split in half

m = n / 2

// recursive sorts

sort a[1..m]

sort a[m+1..n]

// merge sorted sub-arrays using temp array

b = copy of a[1..m]

i = 1, j = m+1, k = 1

while i <= m and j <= n,

a[k++] = (a[j] < b[i]) ? a[j++] : b[i++]

invariant: a[1..k] in final position

while i <= m,

a[k++] = b[i++]

invariant: a[1..k] in final position

* **Counting Inversions:**

CountInversions()

Input array

inv\_count = 0

for i = 0; i < n - 1; i++

for j = i + 1; j < n; j++

if (array[i] > array[j])

inv\_count+

print inv\_count

1. Time Complexity

T(n) = 2T(n/2) + n

T(n/2)=2T(n/4)+n/2

T(n)=2{2T(n/4)+n/2}+n

T(n)=4T(n/4)+2n

T(n/4)=2T(n/8)+n/4

T(n)=4{2T(n/8)+n/4}+2n

T(n)=8T(n/8)+3n

T(n)= 2^k T(n/2^k)+(k)n

as

n/2^k=1

n=2^K

since, k=logn

T(n)=n(1)+nlogn

*Complexity = nlog(n)*

1. Simulation Result

Note:

* Simulation Code and Output file for one RUN for each data size is saved in the Question’s Folder.
* Output For 10000 Elements had a very large size (500 MB) so the file is not attached but the results are mentioned below.
* Each time the code is run the output is different so the inversions and average execution time can differ.
* Code Running Note: Kindly **Change File Path Of Output** before running the code.

|  |  |  |
| --- | --- | --- |
| **Size** | **Inversions** | **Average Execution Time (ns)** |
| 10 | 30 | 229192 |
| 100 | 2292 | 8778366 |
| 1000 | 254164 | 181158892 |
| 10000 | 24730652 | 8675546727 |