**AIM :** Implement Quick sort and Merge sort

**MERGE SORT**

**Code**  :

#include <stdio.h>

#include <stdlib.h>

int \*arr;

int c = 0;

void merge(int l, int mid, int u)

{

    int \*temp, i = l, j = mid + 1, k = 0, m = u - l + 1;

    temp = (int \*)malloc(sizeof(int) \* m);

    while (i <= mid && j <= u)

    {

        if (arr[i] < arr[j])

        {

            temp[k++] = arr[i++];

            c++;

        }

        else

        {

            temp[k++] = arr[j++];

            c++;

        }

    }

    while (i <= mid)

    {

        temp[k++] = arr[i++];

        c++;

    }

    while (j <= u)

    {

        temp[k++] = arr[j++];

        c++;

    }

    for (i = l, j = 0; i <= u; i++, j++)

    {

        arr[i] = temp[j];

    }

}

void merge\_sort(int l, int u)

{

    if (l < u)

    {

        int mid = (l + u) / 2;

        merge\_sort(l, mid);

        merge\_sort(mid + 1, u);

        merge(l, mid, u);

    }

}

int main()

{

    int i, n;

    printf("Enter number of elements:");

    scanf("%d", &n);

    arr = (int \*)malloc(sizeof(int) \* n);

    for (i = 0; i < n; i++)

    {

        arr[i] = rand();

    }

    /\*for(i=0;i<n;i++)

    {

        arr[n-1-i]=i;

    }\*/

    /\*for(i=n-1;i>=0;i--)

    {

        arr[n-1-i]=i;

    }\*/

    merge\_sort(0, n - 1);

    printf("Sorted list is: ");

    for (i = 0; i < n; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("Comparisons =%d\n", c);

    return 0;

}

**Theory :**

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms. Merge sort first divides the array into equal halves and then combines them in a sorted manner. Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too. Merge sort works with recursion and we shall see our implementation in the same way

**Complexity:**

Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.   
T(n) = 2T(n/2) + θ(n)

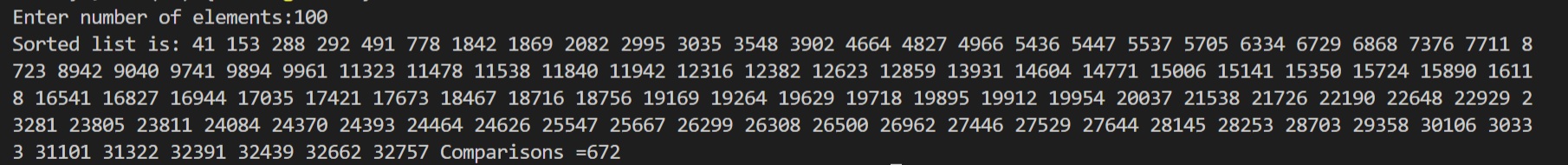
The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is θ(nLogn). Time complexity of Merge Sort is  θ(nLogn) in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.

**TABLE :**

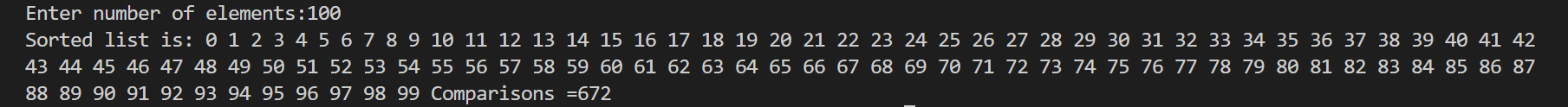
|  |  |
| --- | --- |
| **Cases** | **No. of Comparisons** |
| Random 100 nos. (Average) | 672 |
| 100 nos. in ascending order (Best) | 672 |
| 100 nos. in descending order (Worst) | 672 |

**OUTPUT :**

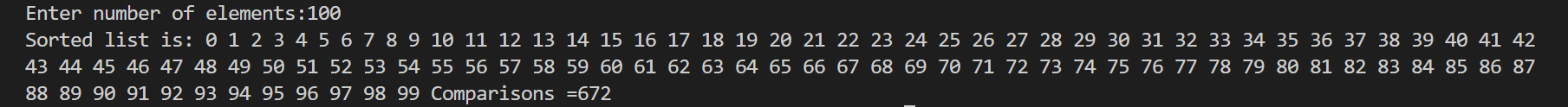
Random 100 nos.



100 nos. previously in ascending order:



100 nos. previously in descending order



**QUICK SORT :**

**CODE :**

#include<stdio.h>

#include<stdlib.h>

int c=0;

int partition(int \*a,int l,int u)

{

int temp,start,end,pivot;

pivot=a[l];

start=l;

end=u+1;

do

{

do

{

start++;

c++;

}while(a[start]<pivot && start<=u);

do

{

end--;

}while(pivot<a[end]);

if(start<end)

{

temp=a[start];

a[start]=a[end];

a[end]=temp;

c++;

}

}while(start<end);

a[l]=a[end];

a[end]=pivot;

c++;

return (end);

}

void quicksort(int \*a,int l,int u)

{

int pivot;

if(l<u)

{

pivot=partition(a,l,u);

quicksort(a,l,pivot-1);

quicksort(a,pivot+1,u);

c++;

}

}

int main()

{

int \*a,i,n;

printf("Enter the number of elements:");

scanf("%d",&n);

a=(int \*)malloc(sizeof(int)\*n);

for (i = 0; i < n; i++)

{

a[i] = rand();

}

/\*for(i=0;i<n;i++)

{

a[n-1-i]=i;

}\*/

/\*for(i=n-1;i>=0;i--)

{

a[n-1-i]=i;

}\*/

quicksort(a,0,n-1);

printf("Sorted List:\n");

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

{

printf("%d\n",a[i]);

}

printf("Comparisons= %d",c);

}

**Theory :**

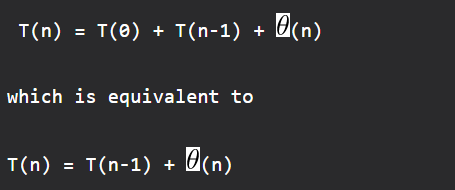
QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of QuickSort that pick pivot in different ways. 1. Always pick first element as pivot. 2. Always pick last element as pivot (implemented below) 3. Pick a random element as pivot. 4. Pick median as pivot. The key process in QuickSort is partition. Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

**Analysis of QuickSort**   
Time taken by QuickSort, in general, can be written as following.

T(n) = T(k) + T(n-k-1) + O(n)

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.   
The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

***Worst Case:*** The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.



The solution of above recurrence is  O(n2).

***Best Case:*** The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.



***Average Case:* .**We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.



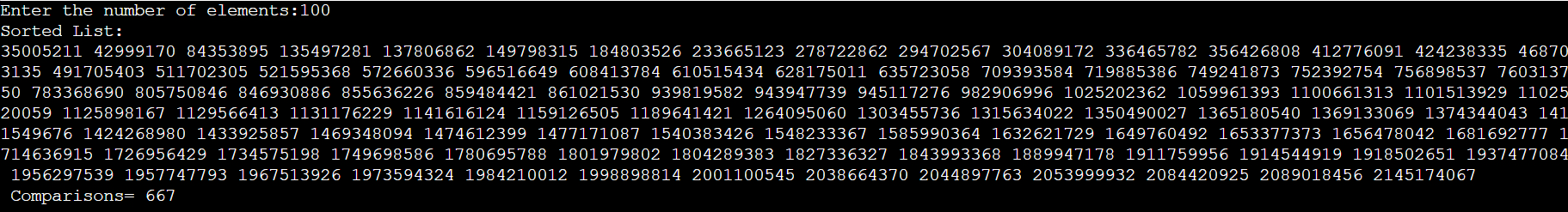
Solution of above recurrence is also O(nLogn)

**TABLE :**

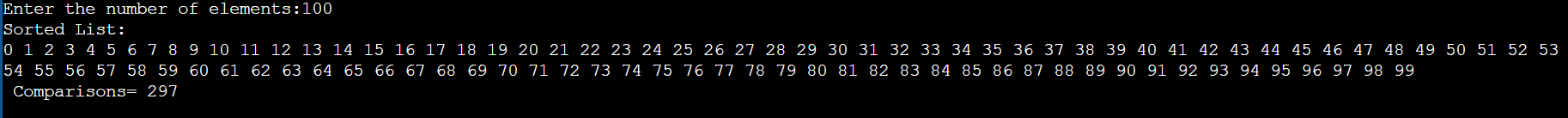
|  |  |
| --- | --- |
| **Cases** | **No. of comparisons** |
| Random 100 nos . (Average) | 667 |
| 100 nos. in ascending order | 297 |
| 100 nos. in descending order | 2797 |

**OUTPUT :**

Random 100 nos.

****

100 nos. previously in ascending order



100 nos. in previously descending order 