Vashisth Zatakia
60003200163 B3 IT

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 O(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

Vashisth Zatakia
60003200163 B3 IT

Complexity:

Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$T(n) = 2T(n/2) + \theta(n)$

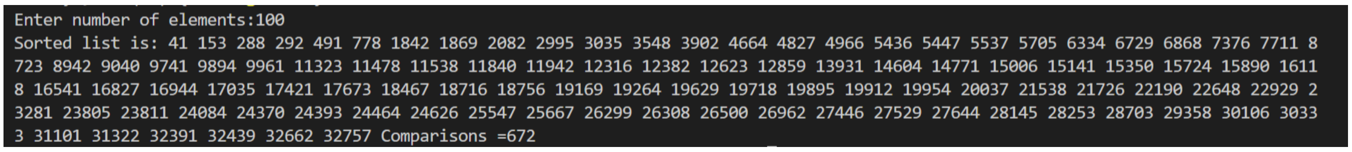
The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is θ(nLogn). Time complexity of Merge Sort is θ(nLogn) in all 3 cases (worst, average and best) as merge sort always divides the array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

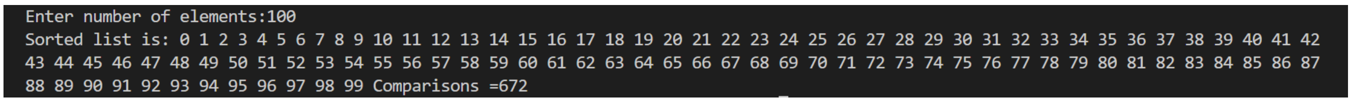
OUTPUT :

Random 100 nos.



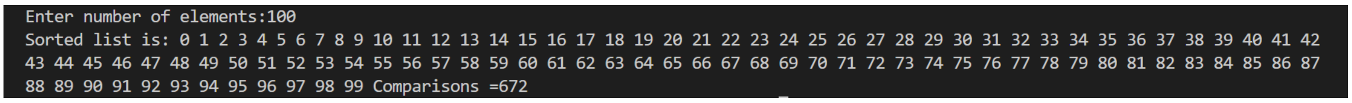
```
Enter number of elements:100
Sorted list is: 41 153 288 292 491 778 1842 1869 2082 2995 3035 3548 3902 4664 4827 4966 5436 5447 5537 5705 6334 6729 6868 7376 7711 8
723 8942 9040 9741 9894 9961 11323 11478 11538 11840 11942 12316 12382 12623 12859 13931 14604 14771 15006 15141 15350 15724 15890 1611
8 16541 16827 16944 17035 17421 17673 18467 18716 18756 19169 19264 19629 19718 19895 19912 19954 20037 21538 21726 22190 22648 22929 2
3281 23805 23811 24084 24370 24393 24464 24626 25547 25667 26299 26308 26500 26962 27446 27529 27644 28145 28253 28703 29358 30106 3033
3 31101 31322 32391 32439 32662 32757 Comparisons =672
```

100 nos. previously in ascending order:



```
Enter number of elements:100
Sorted list is: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 Comparisons =672
```

100 nos. previously in descending order



```
Enter number of elements:100
Sorted list is: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 Comparisons =672
```

Vashisth Zatakia
60003200163 B3 IT

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.

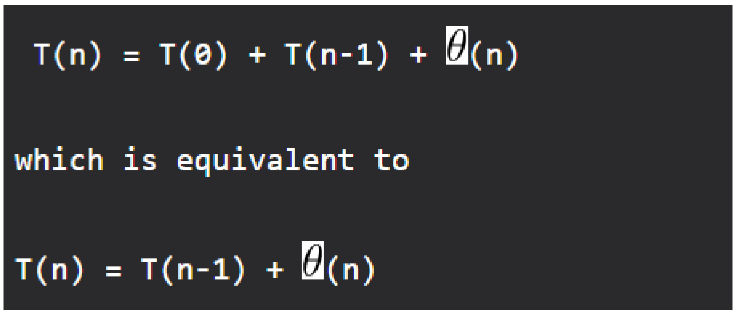
Analysis of QuickSort

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.



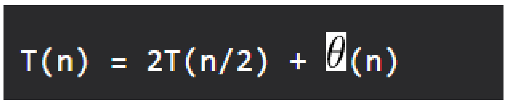
```
 T(n) = T(0) + T(n-1) + θ(n)

which is equivalent to

T(n) = T(n-1) + θ(n)
```

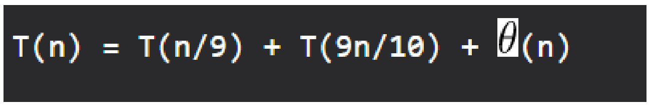
The solution of above recurrence is $O(n^2)$.

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



```
T(n) = 2T(n/2) + θ(n)
```

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.



```
T(n) = T(n/9) + T(9n/10) + θ(n)
```

Solution of above recurrence is also O(nLogn)

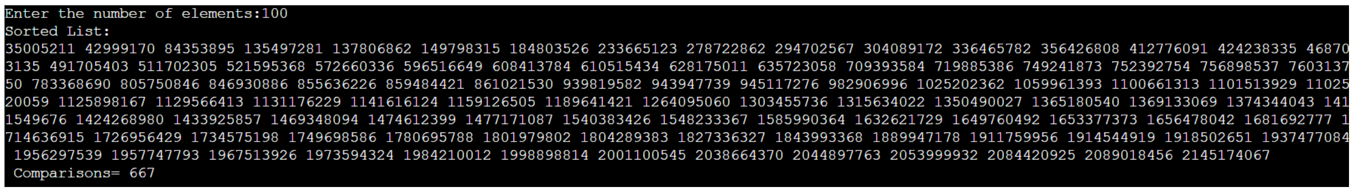
TABLE :

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

Vashisth Zatakia
60003200163 B3 IT

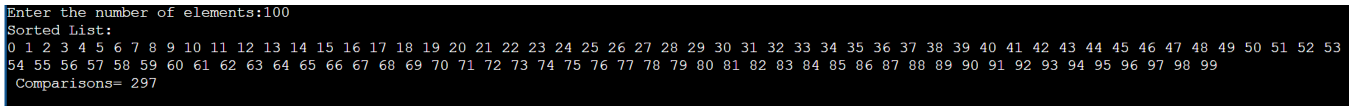
OUTPUT :

Random 100 nos.



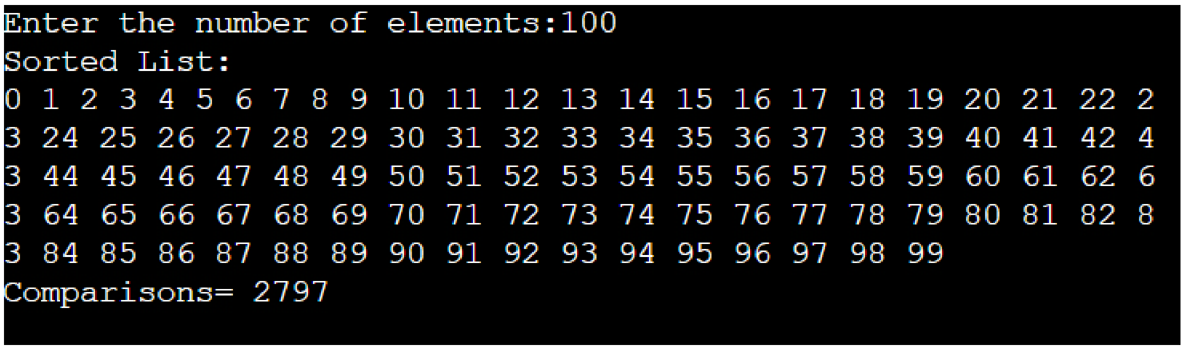
```
Enter the number of elements:100
Sorted List:
35005211 42999170 84353895 135497281 137806862 149798315 184803526 233665123 278722862 294702567 304089172 336465782 356426808 412776091 424238335 46870
3135 491705403 511702305 521595368 572660336 596516649 608413784 610515434 628175011 635723058 709393584 719885386 749241873 752392754 756898537 7603137
50 783368690 805750846 846930886 855636226 859484421 861021530 939819582 943947739 945117276 982906996 1025202362 1059961393 1100661313 1101513929 11025
20059 1125898167 1129566413 1131176229 1141616124 1159126505 1189641421 1264095060 1303455736 1315634022 1350490027 1365180540 1369133069 1374344043 141
1549676 1424268980 1433925857 1469348094 1474612399 1477171087 1540383426 1548233367 1585990364 1632621729 1649760492 1653377373 1656478042 1681692777 1
714636915 1726956429 1734575198 1749698586 1780695788 1801979802 1804289383 1827336327 1843993368 1889947178 1911759956 1914544919 1918502651 1937477084
 1956297539 1957747793 1967513926 1973594324 1984210012 1998898814 2001100545 2038664370 2044897763 2053999932 2084420925 2089018456 2145174067
 Comparisons= 667
```

100 nos. previously in ascending order



```
Enter the number of elements:100
Sorted List:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
 Comparisons= 297
```

100 nos. in previously descending order



```
Enter the number of elements:100
Sorted List:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 2
3 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 4
3 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 6
3 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 8
3 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
Comparisons= 2797
```