# EXPERIMENT NO. 03

**Author:** Harsheet Chordiya

**Roll No.:** 37

**Sem and Section:** 3 CSEB

**Source file:** expt03.c

**Date:** 16/12/2021

================================================================================

**Aim** : To study and implement the divide-and-conquer sorting methods – Merge Sort, Quick Sort, and Heap Sort.

## PROBLEM STATEMENT :

Using the utility functions created in Experiment-2, write a menu driven program to order a list in ascending order using following "divide-and-conquer" approaches – the Quick Sort, the Heap Sort, and the Merge Sort.
You should compare the running time for ordering lists different input sizes in respect of traditional comparison sorts and divide-and-conquer sorts.

================================================================================

## THEORY

================================================================================

### Divide and conquer approach:-

Divide and Conquer is a recursive problem-solving approach which break a problem into smaller subproblems, recursively solve the subproblems, and finally combines the solutions to the subproblems to solve the original problem. This method usually allows us to reduce the time complexity to a large extent.

It consists of three phases:

Divide: Dividing the problem into two or more than two sub-problems that are similar to the original problem but smaller in size.

Conquer: Solve the sub-problems recursively.

Combine: Combine these solutions to subproblems to create a solution to the original problem.

## ALGORITHMS:

### 1.Quick Sort:

```
Step 1: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG =0
Step 2: Repeat Steps 3 to 6 while FLAG =0
Step 3: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
SET RIGHT = RIGHT-1
[END OF LOOP]
Step 4: IF LOC = RIGHT
SET FLAG=1
ELSE IF ARR[LOC] > ARR[RIGHT]
SWAP ARR[LOC] with ARR[RIGHT]
SET LOC = RIGHT
[END OF IF]
Step 5: IF FLAG =0
Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
SET LEFT = LEFT+1
[END OF LOOP]
Step 6: IF LOC = LEFT
SET FLAG=1
ELSE IF ARR[LOC] < ARR[LEFT]
SWAP ARR[LOC] with ARR[LEFT]
SET LOC = LEFT
[END OF IF]
[END OF IF]
Step 7: [END OF LOOP]
Step 8: END
```

**2.Merge Sort:**

```
Step 1: [INITIALIZE] SETI= BEG,J= MID+1, INDEX =0
Step 2: Repeat while (I <= MID) AND (J<=END)
IF ARR[I] < ARR[J]
SET TEMP[INDEX] = ARR[I]
SETI=I+1
ELSE
SET TEMP[INDEX] = ARR[J]
SETJ=J+1
[END OF IF]
SET INDEX = INDEX+1
[END OF LOOP]
Step 3: IFI> MID
Repeat whileJ<= END
SET TEMP[INDEX] = ARR[J]
SET INDEX = INDEX+1, SETJ=J+1
[END OF LOOP]
[Copy the remaining elements of left sub-array, if any]
ELSE
Repeat whileI<= MID
SET TEMP[INDEX] = ARR[I]
SET INDEX = INDEX+1, SETI=I+1
[END OF LOOP]
[END OF IF]
Step 4: [Copy the contents of TEMP back to ARR] SET K=
Step 5: Repeat whileK< INDEX
SET ARR[K] = TEMP[K]
SETK=K+1
[END OF LOOP]
Step 6: END
```

**3.Heap Sort:**

```
Step 1: [Build Heap H]

Repeat forI= to N-1

CALL Insert_Heap(ARR, N, ARR[I])

[END OF LOOP]

Step 2: (Repeatedly delete the root element)

Repeat while N>

CALL Delete_Heap(ARR, N, VAL)

SETN=N+1

[END OF LOOP]

Step 3: END
```

## Time Analysis:-

The average time complexity of quick sort is O(N log(N)).The derivation is based on the following notation: T(N) = Time Complexity of Quick Sort for input of size N.

Worst Case Time Complexity [ Big-O ]: O(N^2)

Best Case Time Complexity [Big-omega]: O(Nlog(N))

Average Time Complexity [Big-theta]: O(Nlog(N))

Merge Sort is quite fast, and has a time complexity of O(n*log n). It is also a stable sort, which means the "equal" elements are ordered in the same order in the sorted list. As we have already learned in Binary Search that whenever we divide a number into half in every step, it can be represented using a logarithmic function, which is log n and the number of steps can be represented by log n + 1(at most)

Also, we perform a single step operation to find out the middle of any subarray, i.e. O(1).

And to merge the subarrays, made by dividing the original array of n elements, a running time of O(n) will be required.

Hence the total time for mergeSort function will become n(log n + 1), which gives us a time complexity of O(n*log n).

Worst Case Time Complexity [ Big-O ]: O(n*log n)

Best Case Time Complexity [Big-omega]: O(n*log n)

Average Time Complexity [Big-theta]: O(n*log n)

Heapsort is a comparison-based sorting algorithm that uses a binary heap data structure. Like mergesort, heapsort has a running time of O(n\log n),O(nlogn), and like insertion sort, heapsort sorts in-place, so no extra space is needed during the sort.

The binary heap data structure allows the heapsort algorithm to take advantage of the heap's heap properties and the heapsort algorithm makes use of the efficient running time for inserting to and deleting from the heap.

Worst Case Time Complexity [ Big-O ]: O(n(log(n))

Best Case Time Complexity [Big-omega]: n * O(1)

Average Time Complexity [Big-theta]: (n∗log(n)−n+O(log(n)))

================================================================================

**PROGRAM**

================================================================================

```
// Header file delecrations

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <time.h>

// Pre-processor Directives (Macros)

#define SIZE 1000000

// Function Declaration/Signatures/Prototypes

void quickSort(int a[], int, int);

int partision(int a[], int, int);

void mergeSort(int a[], int, int);

void merge(int a[], int, int, int);

void copy(int *, int, int *);

void display(int a[], int);

void createRandomList(int *, int, int);
void createAscendingOrderList(int *, int, int);

void createDescendingOrderList(int *, int, int);

void createPartialAscendingOrderList(int *, int, int);

void heapify(int a[], int, int);

void heapSort(int a[], int);

int main()

{

    int responce;

    double elt;

    clock_t begin, end;

    int a[SIZE], n;

    int seed;

    printf("\thow many list elements you want ??\t");

    scanf("%d", &n);

    printf("\n");

    int saved[n];

    do

    {

 printf("\n\tSorting Algorithm in an array and there time complexity ....\n");

 printf("\t1.decreasing order list\t2.increasing order list \n");

 printf("\t3.increasing order with the Nth element out of order \n");

 printf("\t4.Randomly generated list\n");

 printf("\t0.exit\t\n");


 printf("\tOperation Code? ");

 scanf("%d", &responce);

 switch (responce)
```

```
    {
        case 0:
            printf("\tyou opted to Exit...\n");
            break;
        case 1:
            printf("\tenter the seed value : \t");
            scanf("%d", &seed);
            printf("\n");
            createAscendingOrderList(a, n, seed);
            display(a, n);
            copy(a, n, saved);
            begin = clock();
            quickSort(a, 0, n - 1);
            end = clock();
            printf("\tquick sort is completed\n");
            elt = ((double)(end - begin)) / CLOCKS_PER_SEC;
            printf("\ttime elapsed (quick sort is) = %.8lf seconds \n", elt);
            copy(saved, n, a);

            begin = clock();
            heapSort(a, n);
            end = clock();
            printf("\theap sort is completed\n");
            elt = ((double)(end - begin)) / CLOCKS_PER_SEC;
            printf("\ttime elapsed (heap sort is) = %.8lf seconds \n", elt);
            copy(saved, n, a);
            begin = clock();
```

```
mergeSort(a, 0, n - 1);

end = clock();

printf("\tmerge sort is completed\n");

elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

printf("\ttime elapsed (merge sorting is) = %.8lf seconds \n", elt);

display(a, n);

fflush(stdin);

break;

case 2 : printf("\tenter the seed value : \t");

scanf("%d", &seed);

printf("\n");

createDescendingOrderList(a, n, seed);

display(a, n);

copy(a, n, saved);

begin = clock();

quickSort(a, 0, n - 1);

end = clock();

printf("\tquick sort is completed\n");

elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

printf("\ttime elapsed (quick sort is) = %.8lf seconds \n", elt);

copy(saved, n, a);

begin = clock();

heapSort(a, n);

end = clock();

printf("\theap sort is completed\n");

elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

printf("\ttime elapsed (heap sort is) = %.8lf seconds \n", elt);
```

```
        copy(saved, n, a);

        begin = clock();

        mergeSort(a, 0, n - 1);

        end = clock();

        printf("\tmerge sort is completed\n");

        elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

        printf("\ttime elapsed (merge sorting is) = %.8lf seconds \n", elt);

        display(a, n);

        fflush(stdin);

        break;

    case 3:

        printf("\tenter the seed value : \t");

        scanf("%d", &seed);

        printf("\n");

        createPartialAscendingOrderList(a, n, seed);

        display(a, n);

        copy(a, n, saved);

        begin = clock();

        quickSort(a, 0, n - 1);

        end = clock();

        printf("\tquick sort is completed\n");

        elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

        printf("\ttime elapsed (quick sort is) = %.8lf seconds \n", elt);

        copy(saved, n, a);

        begin = clock();

        heapSort(a, n);

        end = clock();
        printf("\theap sort is completed\n");

        elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

        printf("\ttime elapsed (heap sort is) = %.8lf seconds \n", elt);

        copy(saved, n, a);

        begin = clock();

        mergeSort(a, 0, n - 1);

        end = clock();

        printf("\tmerge sort is completed\n");

        elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

        printf("\ttime elapsed (merge sorting is) = %.8lf seconds \n", elt);

        display(a, n);

        fflush(stdin);

        break;

    case 4:

        printf("\tenter the seed value : \t");

        scanf("%d", &seed);

        printf("\n");

        createRandomList(a, n, seed);

        display(a, n);

        copy(a, n, saved);

        begin = clock();

        quickSort(a, 0, n - 1);

        end = clock();

        printf("\tquick sort is completed\n");

        elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

        printf("\ttime elapsed (quick sort is) = %.8lf seconds \n", elt);

        copy(saved, n, a);
```

```
            begin = clock();

            heapSort(a, n);

            end = clock();

            printf("\theap sort is completed\n");

            elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

            printf("\ttime elapsed (heap sort is) = %.8lf seconds \n", elt);

            copy(saved, n, a);

            begin = clock();

            mergeSort(a, 0, n - 1);

            end = clock();

            printf("\tmerge sort is completed\n");

            elt = ((double)(end - begin)) / CLOCKS_PER_SEC;

            printf("\ttime elapsed (merge sorting is) = %.8lf seconds \n", elt);

            display(a, n);

            fflush(stdin);

            break;

        default:

            printf("\tInvalid operation code\n ");

            break;

 }

    } while (responce != 0);

    return 0;

}

void quickSort(int a[], int low, int high)

{

    int partisionIndex;

    if (low < high)
    {
        partisionIndex = partision(a, low, high);
        quickSort(a, low, partisionIndex - 1);
        quickSort(a, partisionIndex + 1, high);
    }
}
int partision(int a[], int low, int high)
{
    int povit = a[low];
    int i = low + 1;
    int j = high;
    int temp;
    do
    {
        while (a[i] <= povit)
            i++;
        while (a[j] > povit)
            j--;
        if (i < j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    } while (i < j);
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    return (j);
}
void merge(int a[], int low, int mid, int high)
{
    int i, j, k, c[high + 1];
    i = low;
    j = mid + 1;
    k = low;
    while (i <= mid && j <= high)
    {
        if (a[i] < a[j])
        {
            c[k] = a[i];
            i++;
        }
        else
        {
            c[k] = a[j];
            j++;
        }
        k++;
    }
while (i <= mid)
    {
        c[k] = a[i];
        i++;
        k++;

    }

    while (j <= high)

    {

        c[k] = a[j];

        j++;

        k++;

    }

    for (int i = low; i <= high; i++)

        a[i] = c[i];

}

void mergeSort(int a[], int low, int high)

{

    int mid;

    if (low < high)

    {

        mid = (low + high) / 2;

        mergeSort(a, low, mid);

        mergeSort(a, mid + 1, high);

        merge(a, low, mid, high);

    }

}

void heapify(int a[], int n, int i)

{

    int largest = i;

    int left = 2 * i + 1;

    int right = 2 * i + 2;
    if (left < n && a[left] > a[largest])
        largest = left;
    if (right < n && a[right] > a[largest])
        largest = right;
    if (largest != i)
    {
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;

        heapify(a, n, largest);
    }
}
void heapSort(int a[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
    for (int i = n - 1; i >= 0; i--)
    {
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;

        heapify(a, i, 0);
    }
}
void display(int a[], int n)
```

```
{
    for (int i = 0; i < n; i++)
    {
        if (i % 5 == 0 && i != 0)
        {
            printf("\n");
        }
        printf("%12d\t", a[i]);
    }
    printf("\n");
}
void copy(int *a, int n, int *b)
{
    int i;
    for (i = 0; i < n; i++)
        b[i] = a[i];
}
void createRandomList(int *arr, int noe, int seed)
{
    int i;
    int minimum = seed;
    int maximum = 10000;
    srand(6000);
    for (i = noe; i != 0; i--)
    {
        arr[i - 1] = minimum + (rand() % (maximum - minimum + 1));
    }
```

```
}

void createAscendingOrderList(int *a, int n, int seed)

{

    int i = 0;

    a[i] = seed;

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

    {

        a[i + 1] = a[i] + 5;

    }

}

void createDescendingOrderList(int *a, int n, int seed)

{

    int i = 0, j = seed + (n - 1) * 5;

    a[0] = j;

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

    {

        a[i + 1] = a[i] - 5;

    }

}

void createPartialAscendingOrderList(int *a, int n, int seed)

{

    int i = 0, j = seed + (n - 1) * 5, temp = seed + 10, flag = 2;

    a[i] = seed;

    a[i + 1] = seed + 5;

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

    {

        flag++;
```

```
            a[i + 1] = temp;

            if (flag % 5 == 0)

            {

                a[i + 1] = -1;

                temp = temp - 5;

            }

            temp = temp + 5;

        }

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

        {

            if (a[i] == -1)

            {

                a[i] = j;

                j = j - 5;

            }

        }

    }
```

================================================================================

## EXECUTION TRAIL

================================================================================

```
how many list elements you want ?? 10

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 1

 enter the seed value : 20

 20 25 30 35 40

 45 50 55 60 65

 quick sort is completed

 time elapsed (quick sort is) = 0.00000200 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00000200 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00000200 seconds

 20 25 30 35 40

 45 50 55 60 65

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 2
 enter the seed value : 50

 95 90 85 80 75

 70 65 60 55 50

 quick sort is completed

 time elapsed (quick sort is) = 0.00000300 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00000200 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00000200 seconds

 50 55 60 65 70

 75 80 85 90 95

CSP252: DSA Lab 2021-2022 Page 15 of

25

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 3

 enter the seed value : 100

 100 105 110 115 145

 120 125 130 135 140

 quick sort is completed

 time elapsed (quick sort is) = 0.00000200 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00000200 seconds

 merge sort is completed
```

```
time elapsed (merge sorting is) = 0.00000200 seconds

100 105 110 115 120

125 130 135 140 145

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit

Operation Code? 4

enter the seed value : 200

4168 2641 1966 672 8103

6965 312 2119 2718 1100

quick sort is completed

time elapsed (quick sort is) = 0.00000200 seconds

heap sort is completed

time elapsed (heap sort is) = 0.00000200 seconds

merge sort is completed

time elapsed (merge sorting is) = 0.00000200 seconds

312 672 1100 1966 2119

2641 2718 4168 6965 8103

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit
```

CSP252: DSA Lab 2021-2022 Page 16 of

25

```
 Operation Code? 0

 you opted to Exit...

-------------------------------------------------------------------------------

 how many list elements you want ?? 100

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 1

 enter the seed value : 10

 quick sort is completed

 time elapsed (quick sort is) = 0.00001400 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00001500 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00000900 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 2

 enter the seed value : 1000

 quick sort is completed

 time elapsed (quick sort is) = 0.00001600 seconds

 heap sort is completed
 time elapsed (heap sort is) = 0.00001400 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00000800 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 3
```

CSP252: DSA Lab 2021-2022 Page 17 of

25

```
 enter the seed value : 10000

 quick sort is completed

 time elapsed (quick sort is) = 0.00000800 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00001300 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00001500 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 4

 enter the seed value : 100000

 quick sort is completed

 time elapsed (quick sort is) = 0.00001400 seconds
 heap sort is completed

 time elapsed (heap sort is) = 0.00001300 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00001400 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 0

 you opted to Exit...
```

--------------------------------------------------------------------------------

```
 how many list elements you want ?? 1000

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 1

CSP252: DSA Lab 2021-2022 Page 18 of

25

 enter the seed value : 10

 quick sort is completed

 time elapsed (quick sort is) = 0.00103100 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00013500 seconds

 merge sort is completed
```

```
time elapsed (merge sorting is) = 0.00008800 seconds

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit

Operation Code? 2

enter the seed value : 10000

quick sort is completed

time elapsed (quick sort is) = 0.00108600 seconds

heap sort is completed

time elapsed (heap sort is) = 0.00017300 seconds

merge sort is completed

time elapsed (merge sorting is) = 0.00010000 seconds

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit

Operation Code? 3

enter the seed value : 40000

quick sort is completed

time elapsed (quick sort is) = 0.00008800 seconds

heap sort is completed

time elapsed (heap sort is) = 0.00015600 seconds

merge sort is completed

time elapsed (merge sorting is) = 0.00007500 seconds
```

```
 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

CSP252: DSA Lab 2021-2022 Page 19 of

25

 Operation Code? 4

 enter the seed value : 50000

 quick sort is completed

 time elapsed (quick sort is) = 0.00011000 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00018300 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00016200 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 0

 you opted to Exit...

--------------------------------------------------------------------------------

 how many list elements you want ?? 10000

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order
 4.Randomly generated list

 0.exit

 Operation Code? 1

 enter the seed value : 10

 quick sort is completed

 time elapsed (quick sort is) = 0.09805900 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00193200 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00100100 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

CSP252: DSA Lab 2021-2022 Page 20 of

25

 Operation Code? 2

 enter the seed value : 100000

 quick sort is completed

 time elapsed (quick sort is) = 0.09676200 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.00158200 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.00090900 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list
```

```
3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit

Operation Code? 3

enter the seed value : 200000

quick sort is completed

time elapsed (quick sort is) = 0.00072300 seconds

heap sort is completed

time elapsed (heap sort is) = 0.00169100 seconds

merge sort is completed

time elapsed (merge sorting is) = 0.00097000 seconds

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit

Operation Code? 4

enter the seed value : 300000

quick sort is completed

time elapsed (quick sort is) = 0.00112800 seconds

heap sort is completed

time elapsed (heap sort is) = 0.00202500 seconds

merge sort is completed

time elapsed (merge sorting is) = 0.00167800 seconds

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order
```

```
 4.Randomly generated list

CSP252: DSA Lab 2021-2022 Page 21 of

25

 0.exit

 Operation Code? 0

 you opted to Exit...

------------------------------------------------------------------------------

 how many list elements you want ?? 100000

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 1

 enter the seed value : 10

 quick sort is completed

 time elapsed (quick sort is) = 9.65150300 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.01881600 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.02768400 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

 Operation Code? 2
 enter the seed value : 100000

 quick sort is completed

 time elapsed (quick sort is) = 9.68571600 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.01859300 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.02772900 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit

CSP252: DSA Lab 2021-2022 Page 22 of

25

 Operation Code? 3

 enter the seed value : 200000

 quick sort is completed

 time elapsed (quick sort is) = 0.00596400 seconds

 heap sort is completed

 time elapsed (heap sort is) = 0.01968300 seconds

 merge sort is completed

 time elapsed (merge sorting is) = 0.02820000 seconds

 Sorting Algorithm in an array and there time complexity ....

 1.decreasing order list 2.increasing order list

 3.increasing order with the Nth element out of order

 4.Randomly generated list

 0.exit
```

```
Operation Code? 4

enter the seed value : 300000

quick sort is completed

time elapsed (quick sort is) = 0.01369800 seconds

heap sort is completed

time elapsed (heap sort is) = 0.02514300 seconds

merge sort is completed

time elapsed (merge sorting is) = 0.03562000 seconds

Sorting Algorithm in an array and there time complexity ....

1.decreasing order list 2.increasing order list

3.increasing order with the Nth element out of order

4.Randomly generated list

0.exit

Operation Code? 0

you opted to Exit...
```

================================================================================

## VIVA-VOCE

================================================================================

**1.Compare and contrast the Quick sort and Merge sort.**

**Ans:** Quick sort is an internal algorithm which is based on divide and conquer strategy. The array of elements is divided into parts repeatedly until it is not possible to divide it further. It is also known as “partition exchange sort”. It uses a key element (pivot) for partitioning the elements. One left partition contains all those elements that are smaller than the pivot and one right partition contains all those elements which are greater than the key element.

Merge sort is an external algorithm and based on divide and conquer strategy. The elements are split into two sub-arrays (n/2) again and again until only one element is left.Merge sort uses additional storage for sorting the auxiliary array.Merge sort uses three arrays where two are used for storing each half, and the third external one is used to store the final sorted list by merging other two and each array is then sorted recursively.At last, the all sub arrays are merged to make it ‘n’ element size of the array.

**2. What is binary heap? Why binary heap is implemented using arrays?**

**Ans:** A binary heap is a heap, i.e, a tree which obeys the property that the root of any tree is greater than or equal to (or smaller than or equal to) all its children (heap property). The primary use of such a data structure is to implement a priority queue. A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as array. The representation is done as The root element will be at Arr[0].

**3. On what account the Quick sort outperforms the merge sort that makes it preferred internal sort of all time.**

**Ans:** The array of elements is divided into parts repeatedly until it is not possible to divide it further. It is also known as “partition exchange sort”. It uses a key element (pivot) for partitioning the elements. One left partition contains all those elements that are smaller than the pivot and one right partition contains all those elements which are greater than the key element. Comparing average complexity we find that both type of sorts have O(NlogN) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space. Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of O(nLogn). The worst case is possible in randomized version also, but worst case doesn’t occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice. Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.

Quick Sort is also tail recursive, therefore tail call optimizations is done.

**4. List chronologically the invention details of above sorting technique, mentioning the inventors and the year of invention.**

**Ans:** 1. Merge sort is a divide and conquer algorithm that was invented by John von Neumann in 1945.

2. Quicksort developed by British computer scientist Tony Hoare in 1959 and published in 1961, it is still a commonly used algorithm for sorting.

3. Heapsort was invented by J. W. J. Williams in 1964. This was also the birth of the heap, presented already by Williams as a useful data structure in its own right. In the same year, R. W. Floyd published an improved version that could sort an array in-place, continuing his earlier research into the treesort algorithm.

================================================================================

## CONCLUSION

================================================================================

In this experiment we have studied what is the sorting and its characteristics. We have also learnt how to write the algorithms for sorting functions. We also learnt how to write functions of different sorts such as quick sort, merge sort, and heap sort. Sir explained us how the sortings work and the code for this sorting methods in detail.

================================================================================

**THANK-YOU!!**

================================================================================