

Lab. Report  
On

# ALGORITHM LAB.

***Submitted in partial fulfillment of the requirements  
for the degree of***

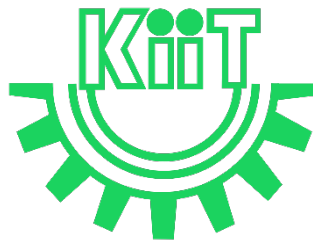
**Bachelor of Technology  
in  
Information Technology**

***Submitted by***

**Group - 1 (IT-G2)**  
**1806467 - Arunima, 1806473 – Hopansh Gahlot,**  
**1806477 – Ishwari Kumari, 1806505 – Rohan Kakar,**  
**1806506 – Ronit Kumar Nayak, 1806510 – Sahil Kumar (GR)**

***Under the Guidance of***

**Prof. Anil Kumar Swain**



**School of Computer Engineering  
Kalinga Institute of Industrial Technology  
Deemed to be University, Bhubaneswar**

**08 NOV 2020**



## Algorithm Laboratory

Sem: 6TH Section: IT-G2 Gr. Leader Roll 1806510 Name: Sahil Kumar

TA/ Instructor(s)

1. Prof. Rajeev Jena

Faculty: Prof. Anil Kumar Swain

2. Prof. Meena Moharana

## CONTENTS

Sl. No.	Title of Lab. Exercises	Page No.
1.	<b>Review of Fundamentals of Data Structures</b>	3
2.	<b>Fundamentals of Algorithmic Problem Solving-I:</b> Analysis of time complexity of small algorithms through step/frequency count method.	22
3.	<b>Fundamentals of Algorithmic Problem Solving-II:</b> Analysis of time complexity of algorithms through asymptotic notations.	
4.	<b>Divide and Conquer Method:</b> Binary Search, Merge Sort, Quick Sort, Randomized Quick Sort	
5.	<b>Heap &amp; Priority Queues:</b> Building a heap, Heap sort algorithm, Min-Priority queue, Max-Priority queue	
6.	<b>Greedy Technique:</b> Fractional knapsack problem, Activity selection problem, Huffman's code	
7.	<b>Dynamic Programming:</b> Matrix Chain Multiplication, Longest Common Subsequence	

# LAB - 1

## Review of Fundamentals of Data Structure

### PROGRAM EXERCISE

### CONTENTS

Prog. No.	Program Title	Page No.
1.1	Write a program to store random numbers into an array of n integers and then find out the smallest and largest number stored in it. n is the user input.	4
1.2	Write a program to store random numbers into an array of n integers, where the array must contain some duplicates. Do the following: a) Find out the total number of duplicate elements. b) Find out the most repeating element in the array.	6
1.3	Write a program to rearrange the elements of an array of n integers such that all even numbers are followed by all odd numbers. How many ways you can solve this problem? Write your approaches & strategy for solving this problem.	8
1.4	Write a program that takes three variable (a, b, c) as separate parameters and rotates the values stored so that value a goes to b, b to c and c to a by using SWAP(x,y)function that swaps/exchanges the numbers x & y.	10
1.5	Let A be n*n square matrix array. WAP by using appropriate user defined functions for the following: a) Find the number of nonzero elements in A b) Find the sum of the elements above the leading diagonal. c) Display the elements below the minor diagonal. d) Find the product of the diagonal elements.	12
1.6	Write a program to find out the second smallest and second largest element stored in an array of n integers. n is the user input. The array takes input through random number generation within a given range. How many ways you can solve this problem? Write your approaches & strategy for solving this problem	15
1.7	Write a program to swap pair of elements of an array of n integers from starting. If n is odd, then last number will remain unchanged.	18
1.8	Write a program to display an array of n integers (n>1), where at every index of the array should contain the product of all elements in the array except the element at the given index. Solve this problem by taking single loop and without an additional array.	20

# PROGRAM SOLUTIONS

## **1.1 Program Title:**

Write a program to store random numbers into an array of n integers and then find out the smallest and largest number stored in it. n is the user input.

### **Input/Output Screenshots:**

**RUN-1:**

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter the length of the array: 10  
The Array :  
28 43 72 79 23 70 55 39 69 1  
Largest = 79  
Smallest = 1
```

**RUN-2:**

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter the length of the array: 15  
The Array :  
28 43 72 79 23 70 55 39 69 1 41 40 5 25 95  
Largest = 95  
Smallest = 1
```

### **Source code:**

```
#include <stdio.h>  
#include <stdlib.h>  
  
int largest(int arr[], int n)  
{  
    int i;  
    int max = arr[0];  
    for (i = 1; i < n; i++)  
        if (arr[i] > max)  
            max = arr[i];  
    return max;  
}  
  
int smallest(int arr[], int n)  
{  
    int i;  
    int min = arr[0];  
    for (i = 1; i < n; i++)  
        if (arr[i] < min)  
            min = arr[i];  
    return min;  
}
```

```

int main()
{
    int length = 0;
    printf("Enter the length of the array: ");
    scanf("%d",&length);
    int arr[length];
    for (int i = 0; i < length; i++)
    {
        arr[i] = rand()%99;
    }
    printf("The Array :\n");
    for (int i = 0; i < length; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\nLargest = %d", largest(arr, sizeof(arr) / sizeof(arr[0])));
    printf("\nSmallest = %d", smallest(arr, sizeof(arr) / sizeof(arr[0])));
    printf("\n");
    return 0;
}

```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

## 1.2 Program Title:

Write a program to store random numbers into an array of n integers, where the array must contain some duplicates. Do the following:

- Find out the total number of duplicate elements.
- Find out the most repeating element in the array.

### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter the length of the array: 10  
The Array :  
3 1 2 0 3 0 1 2 4 1  
No of duplicate elements: 5  
Most repeating elements: 1
```

RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter the length of the array: 15  
The Array :  
3 1 2 0 3 0 1 2 4 1 2 2 0 4 3  
No of duplicate elements: 10  
Most repeating elements: 2
```

### Source code:

```
#include <stdio.h>  
#include <stdlib.h>  
  
int duplicates(int arr[], int length){  
    int count = 0, i, j;  
    for (i = 0; i < length; i++)  
    {  
        for(j = i + 1; j < length; j++)  
        {  
            if(arr[i] == arr[j])  
            {  
                count++;  
                break;  
            }  
        }  
    }  
    return count;  
}  
  
int mostOccurance(int arr[], int n)  
{  
    int max_freq = 0;  
    int ans = -1;
```

```

    for (int i = 0; i <= n-1; i++)
    {
        int curr_freq = 1;
        for (int j = i+1; j <= n-1; j++)
            if (arr[j] == arr[i])
                curr_freq = curr_freq + 1;

        if (max_freq < curr_freq)
        {
            max_freq = curr_freq;
            ans = arr[i];
        }
        else if (max_freq == curr_freq)
            ans = ans < arr[i] ? ans : arr[i];
    }
    return ans;
}

int main()
{
    int length = 0;
    printf("Enter the length of the array: ");
    scanf("%d", &length);
    int arr[length];
    for (int i = 0; i < length; i++)
    {
        arr[i] = rand() % 5;
    }
    printf("The Array :\n");
    for (int i = 0; i < length; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    printf("No of duplicate elements: %d \n", duplicates(arr, length));
    printf("Most repeating elements: %d \n", mostOccurance(arr, length));
    return 0;
}

```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

### 1.3 Program Title:

Write a program to rearrange the elements of an array of n integers such that all even numbers are followed by all odd numbers. How many ways you can solve this problem? Write your approaches & strategy for solving this problem.

#### Input/Output Screenshots:

##### RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter the length of the array: 10  
The Array :  
28 43 72 79 23 70 55 39 69 1  
Array after sorting:  
28 70 72 79 23 43 55 39 69 1
```

##### RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter the length of the array: 15  
The Array :  
28 43 72 79 23 70 55 39 69 1 41 40 5 25 95  
Array after sorting:  
28 40 72 70 23 79 55 39 69 1 41 43 5 25 95
```

#### Source code:

```
#include <stdio.h>  
#include <stdlib.h>  
  
void swap(int *a, int *b){  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
void EvenOdd(int arr[], int size){  
    int left = 0, right = size - 1;  
    while (left < right){  
        while (arr[left] % 2 == 0 && left < right)  
            left++;  
        while (arr[right] % 2 == 1 && left < right)  
            right--;  
  
        if (left < right){  
            swap(&arr[left], &arr[right]);  
            left++;  
            right--;  
        }  
    }  
}
```



```

int main()
{
    int size = 0;
    printf("Enter the length of the array: ");
    scanf("%d", &size);

    int arr[size];
    for (int i = 0; i < size; i++)
        arr[i] = rand() % 99;

    printf("The Array :\n");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");

    EvenOdd(arr, size);

    printf("Array after sorting: \n");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}

```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

#### 1.4 Program Title:

Write a program that takes three variable (a, b, c) as separate parameters and rotates the values stored so that value a goes to be, b, b to c and c to a by using SWAP(x,y)function that swaps/exchanges the numbers x & y.

#### Input/Output Screenshots:

##### RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter 3 integers: 10 11 12  
Before Rotation:  
a = 10  
b = 11  
c = 12  
After Rotation:  
a = 12  
b = 10  
c = 11
```

##### RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter 3 integers: 69 420 101  
Before Rotation:  
a = 69  
b = 420  
c = 101  
After Rotation:  
a = 101  
b = 69  
c = 420
```

#### Source code:

```
#include<stdio.h>  
#include<stdlib.h>  
#include<math.h>  
  
void swap(int *x, int *y){  
    int temp = *x;  
    *x = *y;  
    *y = temp;  
}  
  
void rotate(int *a, int *b, int *c){  
    swap(a,c);  
    swap(b,c);  
}
```

```
void main(){
    int a,b,c;
    printf("Enter 3 integers: ");
    scanf("%d%d%d", &a, &b, &c);
    printf("Before Rotation:\n a = %d\n b = %d\n c = %d\n", a, b, c);
    rotate(&a, &b, &c);
    printf("After Rotation:\n a = %d\n b = %d\n c = %d\n", a, b, c);
}
```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

### 1.5 Program Title:

Let A be  $n \times n$  square matrix array. WAP by using appropriate user defined functions for the following:

- Find the number of nonzero elements in A
- Find the sum of the elements above the leading diagonal.
- Display the elements below the minor diagonal.
- Find the product of the diagonal elements.

### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter order of the matrix  
4  
Enter arr[0][1-4]  
11 3 1 9  
Enter arr[1][1-4]  
4 9 0 12  
Enter arr[2][1-4]  
7 5 8 0  
Enter arr[3][1-4]  
4 13 2 6  
11 3 1 9  
4 9 0 12  
7 5 8 0  
4 13 2 6  
Number of zeros = 2  
Sum of elements above leading diagonal = 25  
Elements below minor diagonal are  
4  
7 5  
4 13 2  
Product of diagonal elements = 4752
```

Source code:

```
#include<stdio.h>  
#include<stdlib.h>  
  
void countZeros(int n,int arr[][n]){  
    int zeros = 0;  
    for (int i = 0; i < n; i++)  
    {  
        for (int j = 0; j < n; j++)  
        {  
            if(arr[i][j] == 0){  
                ++zeros;  
            }  
        }  
    }  
}
```

```

    }

    printf("Number of zeros = %d\n",zeros);
}

void sumLeading(int n,int arr[][n]){
    int sum = 0;
    for(int i = 0;i<n;++i){
        for(int j=i+1;j<n;++j)
        {
            sum+=arr[i][j];
        }
    }

    printf("Sum of elements above leading diagonal = %d\n",sum);
}

void showBelowMinorDiagonal(int n,int arr[][n]){
    printf("Elements below minor diagonal are \n");
    for(int i=1;i<n;++i)
    {
        for(int j=0;j<i;++j)
        {
            printf("%d ",arr[i][j]);
        }
        printf("\n");
    }
}

void diagonalProduct(int n,int arr[][n]){
    int product = 1;
    for (int i = 0; i < n; i++)
    {
        product*=arr[i][i];
    }

    printf("Product of diagonal elements = %d\n",product);
}

int main(){
    printf("Enter order of the matrix\n");
    int input;
    scanf("%d",&input);
    const int n = input;
    int arr[n][n];
    for (int i = 0; i < n; i++)
    {

```

```

        printf("Enter arr[%d][1-%d]\n",i,n);
        for (int j = 0; j < n; j++)
        {
            scanf("%d",&arr[i][j]);
        }

    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ",arr[i][j]);
        }
        printf("\n");
    }

    countZeros(n,arr);
    sumLeading(n,arr);
    showBelowMinorDiagonal(n,arr);
    diagonalProduct(n,arr);
}

```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

### 1.6 Program Title:

Write a program to find out the second smallest and second largest element stored in an array of n integers. n is the user input. The array takes input through random number generation within a given range. How many ways you can solve this problem? Write your approaches & strategy for solving this problem.

#### Input/Output Screenshots:

##### RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter length of array  
10  
Populating array  
  
30 18 0 41 22 34 15 15 27 42  
  
Sorted  
0 15 15 18 22 27 30 34 41 42  
Second smallest element is 15  
Second largest element is 41
```

##### RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter length of array  
15  
Populating array  
  
47 48 20 12 46 23 43 26 19 10 29 48 5 49 45  
  
Sorted  
5 10 12 19 20 23 26 29 43 45 46 47 48 48 49  
Second smallest element is 10  
Second largest element is 48
```

#### Source code:

```
#include<stdio.h>  
#include<stdlib.h>  
  
int randInt(){  
    __time_t t;  
    srand((unsigned) t*rand());  
    return rand()%50;  
}
```

```

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    const int n1 = m - l + 1;
    const int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; ++i)
        L[i] = arr[l + i];
    for (j = 0; j < n2; ++j)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        }
        else {
            arr[k++] = R[j++];
        }
    }
    while (i < n1) {
        arr[k++] = L[i++];
    }

    while (j < n2) {
        arr[k++] = R[j++]; ;
    }
}

void mergeSort(int arr[],int l,int r){
    {
        if(l<r){
            int m = l + (r - l) / 2;
            mergeSort(arr,l,m);
            mergeSort(arr,m+1,r);
            merge(arr,l,m,r);
        }
    }
}

void main(){

    printf("Enter length of array\n");
    int input = 7;
    scanf("%d",&input);
    const int n = input;

```



```

int arr[n];
printf("Populating array\n");
for (int i = 0; i < n; i++)
{
    arr[i] = randInt();
}
printf("\n");
for(int i =0 ; i<n ; ++i){
    printf("%d ",arr[i]);
}
printf("\n");

mergeSort(arr,0,n-1);
printf("\nSorted\n");
for (int i = 0; i < n; i++)
{
    printf("%d ",arr[i]);
}
printf("\n");
printf("Second smallest element is %d\n",arr[1]);
printf("Second largest element is %d\n",arr[n-2]);
}

```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

### 1.7 Program Title:

Write a program to swap pair of elements of an array of n integers from starting. If n is odd, then last number will remain unchanged.

#### Input/Output Screenshots:

##### RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter length of array  
10  
Populating array  
Entered array is  
32 45 7 29 31 33 43 4 31 31  
After swapping  
31 31 4 43 33 31 29 7 45 32
```

##### RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter length of array  
15  
Populating array  
Entered array is  
39 45 21 2 38 21 11 1 41 10 18 19 43 33 7  
After swapping  
33 43 19 18 10 41 1 11 21 38 2 21 45 39 7
```

#### Source code:

```
#include <stdio.h>  
#include <stdlib.h>  
  
int randInt(){  
    __time_t t;  
    srand((unsigned) t*rand());  
    return rand()%50;  
}  
  
void printArray(int n,int arr[n]){  
    for (int i = 0; i < n; i++)  
    {  
        printf("%d ",arr[i]);  
    }  
}  
  
void swap(int *a,int *b){  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```

void swapNumbers(int arr[], int l, int r)
{
    if (l < r)
    {
        swap(&arr[l], &arr[r]);
        swapNumbers(arr, l + 1, r - 1);
    }
}

void main()
{
    printf("Enter length of array\n");
    int input = 7;
    scanf("%d", &input);
    const int n = input;
    int arr[n];
    printf("Populating array\n");
    for (int i = 0; i < n; i++)
    {
        arr[i] = randInt();
    }
    printf("Entered array is \n");
    printArray(n, arr);
    if (!(n & 1))
    {
        swapNumbers(arr, 0, n - 1);
    }
    else
    {
        swapNumbers(arr, 0, n - 2);
    }
    printf("\nAfter swapping\n");
    printArray(n, arr);
    printf("\n");
}

```

### Conclusion/Observation

write here your conclusion/inference/final comment.

### 1.8 Program Title:

Write a program to display an array of n integers ( $n > 1$ ), where at every index of the array should contain the product of all elements in the array except the element at the given index. Solve this problem by taking single loop and without an additional array.

#### Input/Output Screenshots:

##### RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter length of array  
5  
Entering array  
Entered array is  
2 22 29 21 15  
products  
200970 18270 13860 19140 26796
```

##### RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab1_0729$  
Enter length of array  
4  
Entering array  
Entered array is  
8 30 18 27  
products  
14580 3888 6480 4320
```

#### Source code:

```
#include<stdio.h>  
#include<stdlib.h>  
  
int randInt(){  
    __time_t t;  
    srand((unsigned) t*rand());  
    return rand()%50;  
}  
  
void printArray(int n,int arr[n]){  
    for (int i = 0; i < n; i++)  
        printf("%d ",arr[i]);  
}  
  
int calcProduct(int arr[],int l,int r){  
    if(l<r)  
        return arr[l] * arr[r] * calcProduct(arr,l+1,r-1);  
    else if (l == r)  
        return arr[l];  
    else  
        return 1;  
}
```

```

void findProduct(int n,int index,int arr[n]){
    if(index == n){
        return ;
    }
    int productl = calcProduct(arr,0,index-1);
    int productr = calcProduct(arr,index+1,n-1);
    int temp = productl * productr;
    findProduct(n,index+1,arr);
    arr[index] = temp;
}

int main(){
    printf("Enter length of array\n");
    int input;
    scanf("%d", &input);
    const int n = input;
    int arr[n];
    printf("Entering array\n");
    for (int i = 0; i < n; i++)
    {
        arr[i] = randInt();
    }
    printf("Entered array is \n");
    printArray(n, arr);
    findProduct(n,0,arr);
    printf("\nproducts\n");
    printArray(n,arr);
    printf("\n");
}

```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

# Fundamentals of Algorithmic Problem Solving - I

(Analysis of time complexity of small algorithms through step/frequency count method.)

## PROGRAM EXERCISE

### CONTENTS

Prog. No.	Program Title	Page No.
2.1	Write a program to test whether a number n, entered through keyboard is prime or not by using different algorithms you know for at least 10 inputs and note down the time complexity by step/frequency count method for each algorithm & for each input. Finally make a comparison of time complexities found for different inputs, plot an appropriate graph & decide which algorithm is faster.	23
2.2	Write a program to find out GCD (greatest common divisor) using the following three algorithms. a) Euclid's algorithm b) Consecutive integer checking algorithm. c) Middle school procedure which makes use of common prime factors. For finding list of primes implement sieve of Eratosthenes algorithm.	26
2.3	Write a menu driven program as given below, to sort an array of n integers in ascending order by insertion sort algorithm and determine the time required (in terms of step/frequency count) to sort the elements. Repeat the experiment for different values of n and different nature of data (i.e. apply insertion sort algorithm on the data of array that are already sorted, reversely sorted and random data). Finally plot a graph of the time taken versus n for each type of data. The elements can be read from a file or can be generated using the random number generator. INSERTION SORT MENU 1.n Random numbers=>Array 2.Display the Array 3.Sort the Array in Ascending Order by using Insertion Sort Algorithm 4.Sort the Array in Descending Order by using any sorting algorithm 5.Time Complexity to sort ascending of random data 6.TC to sort ascending of data already sorted in ascending order 7.TC to sort ascending of data already sorted in descending order 8.Time Complexity to sort ascending of data for all Cases in Tabular form for values n=5000 to 50000, step=5000	28

# PROGRAM SOLUTIONS

## 2.1 Program Title:

Write a program to store random numbers into an array of n integers and then find out the smallest and largest number stored in it. n is the user input.

### Input/Output Screenshots:

#### RUN-1:

```
Enter 10 numbers
5235 1531 3534 1409 1234 5273 2342 2788 1381 3748

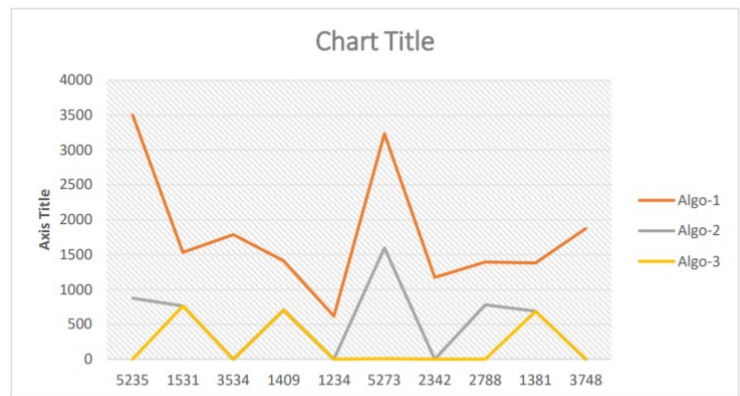
S.No    Input    Time    Taken    By
         Algo-1   Algo-2   Algo-3   Result    Fastest

1        5235    3491    874      2         not prime  algo-3
2        1531    1531    766      763       is prime   algo-3
3        3534    1768    2        1         not prime  algo-3
4        1409    1409    705      702       is prime   algo-3
5        1234    618     2        1         not prime  algo-3
6        5273    5273    2637     2634      is prime   algo-3
7        2342    1172    2        1         not prime  algo-3
8        2788    1395    2        1         not prime  algo-3
9        1381    1381    691      688       is prime   algo-3
10       3748    1875    2        1         not prime  algo-3

sahil@sahil-Probook:~/sem5_lab/DAA/lab2_0805$
```

### Graph:

Input	Time Taken By(Steps)			Result	Fastest
	Algo-1	Algo-2	Algo-3		
5235	3503	877	2	Not Prime	Algo-3
1531	1531	766	763	Prime	Algo-3
3534	1786	2	1	Not Prime	Algo-3
1409	1409	705	702	Prime	Algo-3
1234	618	2	1	Not Prime	Algo-3
5273	3235	1599	6	Not Prime	Algo-3
2342	1172	2	1	Not Prime	Algo-3
2788	1395	782	1	Not Prime	Algo-3
1381	1381	691	688	Prime	Algo-3
3748	1875	2	1	Not Prime	Algo-3



### Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
int approach1Counter = 0;
int approach2Counter = 0;
int approach3Counter = 0;
```

```

void resetCounter(){
    approach1Counter = 0;
    approach2Counter = 0;
    approach3Counter = 0;
}

bool approach1(int n) {
    int count = 0, num = n-1; approach1Counter++;
    while (n % num != 0 && num >= 1) {
        num--; approach1Counter++;
    }
    if (num==1) {
        approach1Counter++;
        return true;
    }
    else {
        approach1Counter++;
        return false;
    }
}

bool approach2(int n) {
    int count = 0, num = n/2; approach2Counter++;
    while (n % num != 0 && num >= 1) {
        num--; approach2Counter++;
    }
    if (num==1) {
        approach2Counter++;
        return true;
    }
    else {
        approach2Counter++;
        return false;
    }
}

bool approach3(int num) {
    if (num < 2) {
        approach3Counter++;
        return false;
    }
    if (num == 2) {
        approach3Counter++;
        return true;
    }
    int count = 0;
    for (int i=2; i<num/2; ++i) {
        if (num % i == 0) {
            count++;
            approach3Counter++;
            break;
        }
    }
}

```



```

        }
        else
            approach3Counter++;
    }
    if(count > 0)
        return false;
    else
        return true;
}

void main() {
    printf("Enter 10 numbers\n");
    int arr[10];
    for(int i = 0;i<10;i++)
        scanf("%d",&arr[i]);
    printf("\t\tTime\tTaken\tBy\t\t\n");
    printf("S.No\tInput\tAlgo-1\tAlgo-2\tAlgo-3\tResult\t\t\tFastest\n\n");
    char result[10];
    for(int i = 0;i<10;i++){
        bool res = approach1(arr[i]);
        approach2(arr[i]);
        approach3(arr[i]);
        if (approach1Counter < approach2Counter && approach1Counter < approach3-
Counter)
            strcpy(result,"algo-1");
        else if (approach2Counter < approach3Counter )
            strcpy(result,"algo-2");
        else
            strcpy(result,"algo-3");
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%s\t%s\n",i+1,arr[i],approach1Counter,ap-
proach2Counter,approach3Counter,res?"is prime":"not prime",result);
        resetCounter();
    }
}

```

### **Conclusion/Observation:**

The three algorithms used were

1. Brute Force from 1 to n
2. Brute force from 1 to n/2
3. Half range search

Findings: Approach 2 and 3 had same time complexity but took half the amount of time as Approach 1 because the range was half.

## 2.2 Program Title:

Write a program to find out GCD (greatest common divisor) using the following three algorithms.

- Euclid's algorithm
- Consecutive integer checking algorithm.
- Middle school procedure which makes use of common prime factors. For finding list of primes implement sieve of Eratosthenes algorithm.

### Input/Output Screenshots:

RUN-1:

```
sahil@sahil-Probook:~/sem5_lab/DAA/lab2_0805$ cd "/home/sahil/sem5_lab/"
```

31415	14142	56	32566	34218	56	12	15	31415	31415	12	12
S.No	Input	Euclid	CIC	Mid Sch	GCD	Fastest					
1	(31415, 14142)	12	14142	14142	1	Euclid					
2	(56, 32566)	6	55	56	2	Euclid					
3	(34218, 56)	4	55	56	2	Euclid					
4	(12, 15)	3	10	12	3	Euclid					
5	(31415, 31415)	2	1	31415	31415	CIC					
6	(12, 12)	2	1	12	12	CIC					

### Source code:

```
#include<stdio.h>
#include <string.h>

int count1 = 0;
int count2 = 0;
int count3 = 0;
void resetCounter(){
    count1 = 0;
    count2 = 0;
    count3 = 0;
}
int euclid(int a, int b){
    count1++;
    if (a == 0)
        return b;
    return euclid(b%a, a);
}
int cid(int a,int b,int t){
    count2++;
    if( a%t == 0 && b%t == 0)
        return t;
    else return cid(a,b,t-1);
}
int middleSchool(int n1, int n2){
    int gcd;
    for(int i=1; i <= n1 && i <= n2; ++i){
        count3++;
        if(n1%i==0 && n2%i==0)
```

```

        gcd = i;
    }
    return gcd;
}

void main(){
    int arr[12];
    for(int i=0;i<12;i++)
        scanf("%d",&arr[i]);
    char result[12];
    printf("S.No\t    Input\tEuclid\tCIC\tMid Sch\tGCD\tFastest\n");
    printf("-----\n");
    ;
    for(int i=0,j=0;i<12;i+=2){
        int gcd1 = euclid(arr[i],arr[i+1]);
        int gcd2 = cid(arr[i],arr[i+1], arr[i]>arr[i+1] ? arr[i+1] : arr[i]);
        int gcd3 = middleSchool(arr[i],arr[i+1]);
        if (count1 < count2 && count1 < count3)
            strcpy(result,"Euclid");
        else if (count2 < count3 )
            strcpy(result,"CIC");
        else
            strcpy(result,"Mid Sch");
        printf("%d\t(%d,\t%d)\t%d\t%d\t%d\t%d\t%s\n",+
+j,arr[i],arr[i+1],count1,count2,count3,gcd1,result);
        resetCounter();
    }
}

```

### Conclusion/Observation:

Three procedures were used

1. Euclid
2. CIC
3. Middle School

Findings: Middle School algorithm performed the worst with a time complexity of  $O(N^3)$ .  
Euclid was generally faster with a time complexity of  $\log(\min(a,b))$

### 2.3 Program Title:

Write a menu driven program as given below, to sort an array of n integers in ascending order by insertion sort algorithm and determine the time required (in terms of step/frequency count) to sort the elements. Repeat the experiment for different values of n and different nature of data. Finally plot a graph of the time taken versus n for each type of data. The elements can be read from a file or can be generated using the random number generator.

#### INSERTION SORT MENU

1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting algorithm
5. Time Complexity to sort ascending of random data
6. TC to sort ascending of data already sorted in ascending order
7. TC to sort ascending of data already sorted in descending order
8. Time Complexity to sort ascending of data for all Cases in Tabular form for values n=5000 to 50000, step=5000

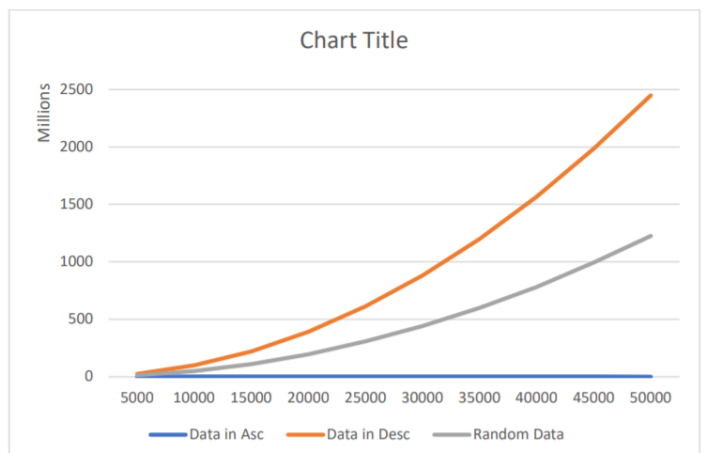
### Input/Output Screenshots:

#### RUN-1:

```
Select an option
0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting algorithm
5. Time Complexity to sort ascending of random data
6. Time Complexity to sort ascending of data already sorted in ascending order
7. Time Complexity to sort ascending of data already sorted in descending order
8. Time Complexity to sort ascending of data for all Cases (Data Ascending, Data in Descending & Random Data)
in Tabular form for values n=5000 to 50000, step=5000
8
S.No  Val N  Data in Asc  Data in Desc  Random Data
1      5000    24996      24519562     12323670
2     10000    49996      98042114     48923570
3     15000    74996      220559384    111690280
4     20000    99996      392082654    195528166
5     25000   124996     612599596    305313302
6     30000   149996     882110878    441825450
7     35000   174996    1200615514    600831952
8     40000   199996    1568139030    781956510
9     45000   224996    1984645892    993019280
10    50000   249996    2450140932   1224990256
```

### Graph:

S.No	Val N	Data in Asc	Data in Desc	Random Data
1	5000	24996	24519828	12158650
2	10000	49996	98041528	48915068
3	15000	74996	220564436	110104332
4	20000	99996	392088526	196944648
5	25000	124996	612607604	307674746
6	30000	149996	882115810	441821594
7	35000	174996	1200639446	599077570
8	40000	199996	1568152196	782485438
9	45000	224996	1984663508	994336780
10	50000	249996	2450178590	1225635696



### Source code:

```
#include <stdio.h>
#include <stdlib.h>

int randInt()
{
    __time_t t;
    srand((unsigned)t * rand());
    return rand() % 50;
}

long insertionSort(int arr[], int n)
{
    long count = 1;
    for (int j = 1; j < n; ++j, count++)
    {
        count++; //number of times temp is assigned
        int temp = arr[j];
        count++; //number of times i is assigned
        int i = j - 1;
        count++; //number of times while loop is initialized
        while (arr[i] > temp && i >= 0)
        {
            count++; //number of shift operations
            arr[i + 1] = arr[i];
            count++; //number of i decrements
            i--;
        }
        count++; // number of times temp is assigned to it's position
        arr[i + 1] = temp;
    }
    return count;
}

long insertionSortDescending(int arr[], int n)
{
    long count = 1;
    for (int j = 1; j < n; ++j, count++)
    {
        count++; //number of times temp is assigned
        int temp = arr[j];
        count++; //number of times i is assigned
        int i = j - 1;
        count++; //number of times while loop is initialized
        while (arr[i] < temp && i >= 0)
        {
            count++; //number of shift operations
            arr[i + 1] = arr[i];
            count++; //number of i decrements
```

```

        i--;
    }
    count++; // number of times temp is assigned to it's position
    arr[i + 1] = temp;
}
return count;
}

void analyze(int n){
    int *arr1 = malloc(n*sizeof(int)), *arr2 = malloc(n*sizeof(int)), *arr3 = malloc(n*sizeof(int)); //for ascending, descending and random respectively
    for (int i = 0; i < n; ++i)
    {
        int num = randInt();
        arr1[i] = num;
        arr2[i] = num;
        arr3[i] = num;
    }
    insertionSort(arr1,n); //sort data in ascending order
    insertionSortDescending(arr2,n); //sort data in descending order

    long count1 = insertionSort(arr1,n); //count steps for sorting already ascending sorted data;
    long count2 = insertionSort(arr2,n); //count steps for sorting descending sorted data;
    long count3 = insertionSort(arr3,n); //count steps for sorting random data

    printf("%5d\t%5d\t%10ld\t%10ld\t%10ld",n/5000,n,count1,count2,count3);
}

void main()
{
    int *arr;
    int n;
    int ch = 0;
    long count = 0;

    do
    {
        printf("Select an option\n");
        printf("\
            0. Quit\n\
            1. n Random numbers=>Array\n\
            2. Display the Array\n\
            3. Sort the Array in Ascending Order by using Insertion Sort Algorithm\n\
            4. Sort the Array in Descending Order by using any sorting algorithm\n\
            5. Time Complexity to sort ascending of random data\n\
            6. Time Complexity to sort ascending of data already sorted in ascending order\n\

```

```

7. Time Complexity to sort ascending of data already sorted in de-
scending order\n\
8. Time Complexity to sort ascending of data for all Cases (Data As-
cending, Data in Descending & Random Data) in Tabular form for values n=5000 to 5
0000, step=5000\n");
scanf("%d", &ch);
switch (ch)
{
case 0:
    break;
case 1:
{
    printf("Enter length of array\n");
    scanf("%d",&n);
    arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        arr[i] = randInt();
    break;
}
case 2:{
    printf("The array is \n");
    for(int i=0;i<n;++i)
        printf("%d ",arr[i]);
    printf("\n");
    break;
}
case 3:{
    count = insertionSort(arr,n);
    printf("Array of length %d sorted in %ld steps \n",n,count);
    printf("Array after sorting is \n");
    for(int i=0;i<n;++i)
        printf("%d ",arr[i]);
    printf("\n");
    break;
}
case 4:{
    insertionSortDescending(arr,n);
    printf("Array after sorting is \n");
    for(int i=0;i<n;++i)
        printf("%d ",arr[i]);
    printf("\n");
    break;
}
case 5:{
    count = insertionSort(arr,n);
    printf("Array of length %d sorted in %ld steps \n",n,count);
    break;
}
case 6:{
    insertionSort(arr,n);

```

```

        count = insertionSort(arr,n);
        printf("An already sorted array was sorted in %ld steps\n",count);
        break;
    }
    case 7:{
        insertionSortDescending(arr,n);
        count = insertionSort(arr,n);
        printf("An array sorted in descending order was sorted in %ld steps\n",count);
    }
    case 8:{
        printf("S.No\tVal N\tData in Asc\tData in Desc\tRandom Data\n");
        for(int i=5000;i<=50000;i+=5000){
            analyze(i);
            printf("\n");
        }
        break;
    }
}
} while (ch != 0);
}

```

### **Conclusion/Observation:**

For insertion sort, 3 cases were taken

1. Random Data
2. Sorted Data
3. Reverse sorted data

Findings: The algorithm shows best case time complexity of  $O(N)$  on sorted data. It shows worst case time complexity  $O(N^2)$  on reverse sorted data. It tends to perform moderately on random data.



## LAB - 3

# Fundamentals of Algorithmic Problem Solving-II:

(Analysis of time complexity of algorithms through the concept of asymptotic notations)

### PROGRAM EXERCISE

## CONTENTS

Prog. No.	Program Title	Page No.
3.1	Rewrite the program no- 2.1 (Insertion Sort) with the following details. i. Compare the best case, worst case, and average case time complexity with the same data except time complexity will count the CPU clock time. ii. Plot a graph showing the above comparison (n, the input data Vs. CPU times for best, worst & average case) iii. Compare manually program no- 2.1 graph vs program no- 3.1 graph and draw your inference	
3.2	Let A be a list of n (not necessarily distinct) integers. Write a program by using User Defined Function (UDF)s to test whether any item occurs more than $\lceil n/2 \rceil$ times in A. a) UDF should take $O(n^2)$ time and use no additional space. b) UDF should take $O(n)$ time and use $O(1)$ additional space.	
3.3	Write a program by using a user defined function for computing $\lfloor \sqrt{n} \rfloor$ for any positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.	
3.4	Let A be an array of n integers $a_0, a_1, \dots, a_{n-1}$ (negative integers are allowed), denoted, by $A[i \dots j]$ , the sub-array $a_i, a_{i+1}, \dots, a_j$ for $i \leq j$ . Also let $S_{i-j}$ denote the sum $a_i + a_{i+1} + \dots + a_j$ . Write a program by using an udf that must run in $O(n^2)$ time to find out the maximum value of $S_{i-j}$ for all the pair i, j with $0 \leq i \leq j \leq n-1$ . Call this maximum value S. Also obtains the maximum of these computed sums. Let $j < i$ in the notation $A[i \dots j]$ is also allowed. In this case, $A[i \dots j]$ denotes the empty sub-array (that is, a sub-array that ends before it starts) with sum $S_{i-j} = 0$ . Indeed, if all the elements of A are negative, then one returns 0 as the maximum sub-array sum. For example, for the array $A[] = \{1, 3, 7, -2, -1, -5, -1, -2, -4, 6, 2\}$ . This maximum sum is $S = S_{0-2} = 1+3+7=11$ .	
3.5	Design a data structure to maintain a set S of n distinct integers that supports the following two operations: a) INSERT (x, S): insert integer x into S	

	b) REMOVE-BOTTOM-HALF(S): remove the smallest $\lceil n/2 \rceil$ integers from S. Write a program by using UDFs that give the worse-case time complexity of the two operations INSERT(x, S) in $O(1)$ time and REMOVE-BOTTOM-HALF(S) in $O(n)$ time.	
<b>3.6</b>	Consider an $n \times n$ matrix $A = (a_{ij})$ , each of whose elements $a_{ij}$ is a nonnegative real number and suppose that each row and column of A sums to an integer value. We wish to replace each element $a_{ij}$ with either $\lfloor a_{ij} \rfloor$ or $\lceil a_{ij} \rceil$ without disturbing the row and column sums. Write a program by defining a user defined function that is used to produce the rounded matrix as described in the above example. Find out the time complexity of your algorithm/function.	

# PROGRAM SOLUTIONS

## 3.1 Program Title:

Rewrite the program no- 2.1 (Insertion Sort) with the following details.

- Compare the best case, worst case, and average case time complexity with the same data except time complexity will count the CPU clock time.
- Plot a graph showing the above comparison (n, the input data Vs. CPU times for best, worst & average case)

Compare manually program no- 2.1 graph vs program no- 3.1 graph and draw your inference

## Input/Output Screenshots:

### RUN-1:

```
MAIN MENU FOR INSERTION SORTSelect An Option:          0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting algorithm
5. Time Complexity to sort ascending of random data
6. Time Complexity to sort ascending of data already sorted in ascending order
7. Time Complexity to sort ascending of data already sorted in descending order
8. Time Complexity to sort ascending of data for all Cases
   (Data Ascending, Data Descending & Random Data) in a table for values n=5000 to 50000,step=5000
```

S.No	Val N	Best Case	Worst Case	Random Data
1	5000	0.000022	0.032762	0.016591
2	10000	0.000040	0.129634	0.065105
3	15000	0.000059	0.297738	0.147263
4	20000	0.000086	0.527723	0.260395
5	25000	0.000096	0.822368	0.407052
6	30000	0.000141	1.170978	0.585731
7	35000	0.000134	1.600844	0.820009
8	40000	0.000154	2.082813	1.041582
9	45000	0.000170	2.643699	1.312801
10	50000	0.000206	3.263022	1.636584

## Graph:

S.No	Val N	Data in Asc	Data in Desc	Random Data
1	5000	0.000015	0.032811	0.016012
2	10000	0.000031	0.129267	0.064571
3	15000	0.000046	0.289408	0.143626
4	20000	0.000061	0.513867	0.256122
5	25000	0.000075	0.804302	0.404248
6	30000	0.000091	1.158664	0.579988
7	35000	0.000105	1.578975	0.79128
8	40000	0.00012	2.064948	1.031703
9	45000	0.000132	2.598908	1.29938
10	50000	0.000149	3.221705	1.609535



### Source code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int randInt(){
    __time_t t;
    srand((unsigned)t * rand());
    return rand() % 1000;
}

void insertionSort(int arr[], int n,int p){
    for (int j = 1; j < n; ++j){
        int temp = arr[j];
        int i = j - 1;
        while (arr[i] > temp && i >= 0){
            arr[i + 1] = arr[i];
            i--;
        }
        arr[i + 1] = temp;
    }
    if(p==1){
        for(int i=0;i<n;++i)
            printf("%d ",arr[i]);
        printf("\n");
    }
}

void insertionSortDescending(int arr[], int n,int p){
    for (int j = 1; j < n; ++j){
        int temp = arr[j];
        int i = j - 1;
        while (arr[i] < temp && i >= 0){
            arr[i + 1] = arr[i];
            i--;
        }
        arr[i + 1] = temp;
    }
    if(p==1){
        for(int i=0;i<n;++i)
            printf("%d ",arr[i]);
        printf("\n");
    }
}

void analyze(int n){
    int *arr1 = malloc(n*sizeof(int)),*arr2 = malloc(n*sizeof(int)),*arr3 = malloc(n*sizeof(int)); //for ascending, descending and random respectively
    for (int i = 0; i < n; ++i){
```

```

        int num = randInt();
        arr1[i] = num;
        arr2[i] = num;
        arr3[i] = num;
    }
    clock_t t;
    double time_taken;
    insertionSort(arr1,n,0); //sort data in ascending order
    insertionSortDescending(arr2,n,0); //sort data in descending order

    t = clock();
    insertionSort(arr1,n,0); //count seconds for sorting already ascending sorted
data;
    t = clock() - t;
    time_taken = ((double)t)/CLOCKS_PER_SEC;
    double count1 = time_taken;

    t = clock();
    insertionSort(arr2,n,0); //count seconds for sorting descending sorted data;
    t = clock() - t;
    time_taken = ((double)t)/CLOCKS_PER_SEC;
    double count2 = time_taken;

    t = clock();
    insertionSort(arr3,n,0); //count seconds for sorting random data
    t = clock() - t;
    time_taken = ((double)t)/CLOCKS_PER_SEC;
    double count3 = time_taken;
    printf("%5d\t%5d\t%10f\t%10f\t%10f",n/5000,n,count1,count2,count3);
}
void main(){
    int *arr;
    int n;
    int ch = 0;
    double count = 0;
    clock_t t;
    double time_taken;
    do{
        printf("MAIN MENU FOR INSERTION SORT");
        printf("Select An Option:");
        printf("\n
            0. Quit\n\
            1. n Random numbers=>Array\n\
            2. Display the Array\n\
            3. Sort the Array in Ascending Order by using Insertion Sort Algo-
rithm\n\
            4. Sort the Array in Descending Order by using any sorting algorithm\
n\
            5. Time Complexity to sort ascending of random data\n\

```

```

        6. Time Complexity to sort ascending of data already sorted in as-
ascending order\n\
        7. Time Complexity to sort ascending of data already sorted in de-
descending order\n\
        8. Time Complexity to sort ascending of data for all Cases\n\t\t(Data
Ascending, Data Descending & Random Data) in a table for values n=5000 to 50000,s
tep=5000\n");
scanf("%d", &ch);
switch (ch)
{
case 0:
    break;
case 1:{
    printf("Enter length of array\n");
    scanf("%d",&n);
    arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        arr[i] = randInt();
    break;
}
case 2:{
    printf("The array is \n");
    for(int i=0;i<n;++i)
        printf("%d ",arr[i]);
    printf("\n");
    break;
}
case 3:{
    insertionSort(arr,n,1);
    break;
}
case 4:{
    insertionSortDescending(arr,n,1);
    break;
}
case 5:{
    t = clock();
    insertionSort(arr,n,0);
    t = clock() - t;
    time_taken = ((double)t)/CLOCKS_PER_SEC;
    double count = time_taken;
    printf("Array of length %d sorted in %f seconds \n",n,count);
    break;
}
case 6:{
    t = clock();
    insertionSort(arr,n,0);
    insertionSort(arr,n,0);
    t = clock() - t;
    time_taken = ((double)t)/CLOCKS_PER_SEC;

```

```

        double count = time_taken;
        printf("An already sorted array was sorted in %f seconds\n",count);
        break;
    }
    case 7:{
        t = clock();
        insertionSortDescending(arr,n,0);
        insertionSort(arr,n,0);
        t = clock() - t;
        time_taken = ((double)t)/CLOCKS_PER_SEC;
        double count = time_taken;
        printf("Array sorted in descending order was sorted in %f seconds\n",
count);
        break;
    }
    case 8:{
        printf("S.No\tVal N\tBest Case\tWorst Case\tRandom Data\n");
        for(int i=5000;i<=50000;i+=5000){
            analyze(i);
            printf("\n");
        }
        break;
    }
}
} while (ch != 0);
}

```

### **Conclusion/Observation**

It was observed that the data arranged in descending order took the highest time whereas data arranged in ascending order took least time, to arrange the data in ascending order, whereas data arrange in random order always took time in between ascended and descended sorted array. Hence it can be concluded that time complexity of insertion sort depends on data.

### 3.2 Program Title:

Let A be a list of n (not necessarily distinct) integers. Write a program by using User Defined Function (UDF)s to test whether any item occurs more than  $\lceil n/2 \rceil$  times in A.

- UDF should take  $O(n^2)$  time and use no additional space.
- UDF should take  $O(n)$  time and use  $O(1)$  additional space.

#### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab3_0812$  
arr[0] = 1  
arr[1] = 2  
arr[2] = 3  
arr[3] = 4  
arr[4] = 2  
arr[5] = 2  
arr[6] = 6  
arr[7] = 2  
arr[8] = 2  
O(n^2) approach  
2 occurs more than n/2 times  
O(n) approach  
2 was repeated more than n/2 times
```

#### Source code

```
#include<stdio.h>  
#include<stdlib.h>  
  
int randInt(){  
    __time_t t;  
    srand((unsigned)t * rand());  
    return rand() % 50;  
}  
  
int findMax(int arr[],int n){  
    int max = 0;  
    for(int i=0;i<n;++i){  
        if (max < arr[i])  
            max = arr[i];  
    }  
    return max;  
}  
  
void main(){  
    int arr[] = {1,2,3,4,2,2,6,2,2};  
    int n = 9;  
    for (int i = 0; i < n; i++)  
        printf("arr[%d] = %d\n",i,arr[i]);  
    printf("O(n^2) approach\n");
```



```

for(int i=0;i<n;++i){
    int num = arr[i];
    int count = 0;
    for(int j=i+1;j<n;++j)
        if(num == arr[j])
            count++;
    if(count>=n/2)
        printf("%d occurs more than n/2 times\n",num);
}

int max = findMax(arr,n);
int b[max];
for(int i=0;i<max;++i)
    b[i] = 0;
for(int i=0;i<n;++i){
    int num = arr[i];
    int count = 0;
    b[num]++;
}
printf("O(n) approach\n");
for(int i=0;i<max;++i)
    if(b[i] > n/2)
        printf("%d was repeated more than n/2 times\n",i);
}

```

### Conclusion/Observation

Elements repeating more than  $n/2$  time were found using two different approach. Two different UDF found same result but with different time complexity as  $O[n]$  and  $O[n^2]$ .

### 3.3 Program Title:

Write a program by using a user defined function for computing  $\lfloor \sqrt{n} \rfloor$  for any positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.

#### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab3_0812$  
Enter a number: 64  
Result: 8
```

RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab3_0812$  
Enter a number: 36  
Result: 6
```

#### Source code:

```
#include <stdio.h>  
  
void compute(int number)  
{  
    int sqrt = number / 2;  
    int temp = 0;  
    while (sqrt != temp)  
    {  
        temp = sqrt;  
        sqrt = (number / temp + temp) / 2;  
    }  
    printf("Result: %d\n", sqrt);  
}  
  
void main()  
{  
    int n;  
    printf("Enter a number: ");  
    scanf("%d", &n);  
    compute(n);  
}
```

#### Conclusion/Observation

Square root of number was found using an UDF with time complexity  $O[n]$ .

### 3.4 Program Title:

Let A be an array of n integers  $a_0, a_1, \dots, a_{n-1}$  (negative integers are allowed), denoted, by A [i... j], the sub-array  $a_i, a_{i+1}, \dots, a_j$  for  $i \leq j$ . Also let  $S_{i-j}$  denote the sum  $a_i + a_{i+1} + \dots + a_j$ . Write a program by that must run in  $O(n^2)$  time to find out the maximum value of  $S_{i-j}$  for all the pair i, j with  $0 \leq i \leq j \leq n-1$ . Call this maximum value S. Also obtains the maximum of these computed sums. Let  $j < i$  in the notation A [i... j] is also allowed. In this case, A[i... j] denotes the empty sub-array (that is, a sub-array that ends before it starts) with sum  $S_{i-j}=0$ . Indeed, if all the elements of A are negative, then one returns 0 as the maximum sub-array sum.

#### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab3_0812$  
1 3 7 -2 -1 -5 -1 -2 -4 6 2  
Max subarray is 11
```

#### Source code:

```
#include <stdio.h>  
#include <stdlib.h>  
int findMax(int arr[], int n)  
{  
    int max = 0;  
    for(int i=0;i<n;++i){  
        int sum = 0;  
        for(int j=i;j<n;++j){  
            sum+=arr[j];  
            if(sum > max)  
                max = sum;  
        }  
    }  
    return max;  
}  
  
int main(){  
    int A[] = {1, 3, 7, -2, -1, -5, -1, -2, -4, 6, 2};  
    int max = 0;  
    int n = sizeof(A) / sizeof(int);  
    max = findMax(A, n);  
    for(int i=0;i<n;i++)  
        printf("%d ",A[i]);  
    printf("\n");  
    printf("Max subarray is %d\n", max);  
}
```

#### Conclusion/Observation:

Range with maximum sum of elements was found using a brute force method with time complexity  $O[n^2]$ .

### 3.5 Program Title:

Design a data structure to maintain a set S of n distinct integers that supports the following two operations:

- INSERT (x, S): insert integer x into S
- REMOVE-BOTTOM-HALF(S): remove the smallest  $\lceil n/2 \rceil$  integers from S. Write a program by using UDFs that give the worse-case time complexity of the two operations INSERT(x, S) in O(1) time and REMOVE-BOTTOM-HALF(S) in O(n) time.

#### Input/Output Screenshots:

##### RUN-1:

```
0.To exit
1.To Insert Element
2.Remove Bottom Half(S)
1 0

Enter the value to insert

After Operation
H->0

0.To exit
1.To Insert Element
2.Remove Bottom Half(S)
1 2

Enter the value to insert

After Operation
H->2->0

0.To exit
1.To Insert Element
2.Remove Bottom Half(S)
1 4

Enter the value to insert

After Operation
H->4->2->0

0.To exit
1.To Insert Element
2.Remove Bottom Half(S)
1 9

Enter the value to insert

After Operation
H->9->4->2->0

0.To exit
1.To Insert Element
2.Remove Bottom Half(S)
1 6

Enter the value to insert

After Operation
H->6->9->4->2->0
```

### Source code:

```
#include <stdio.h>
#include <stdlib.h>
int A[100000];
int size=0;
struct node{
    int val;
    struct node* next;
};
void Insert(struct node **head, int value)
{
    if(A[value]==0){
        struct node * new_node = NULL;
        new_node = (struct node *)malloc(sizeof(struct node));
        if (new_node == NULL)
        {
            printf("Failed to insert element.Out of memory");
            return;
        }
        new_node->val = value;
        new_node->next = *head;
        *head = new_node;
        A[value]=1;
    }
}
void Display(struct node *head)
{
    struct node *temp;
    temp=head;
    printf("\033[0;32m");
    printf("H");
    while(temp)
    {
        printf("->%d", temp->val);
        temp = temp->next;
    }
    printf("\n\n");
    printf("\033[0m");
}
void Asc(struct node *head){
    int max;
    struct node *temp2=head,*temp3=head,*temp4=head,*temp5=head;
    for(int i=0;temp2->next!=NULL;temp2=temp2->next){
        if(max<=temp2->val) max=temp2->val;
        size=size+1;
    }
    int M[max+1];
    for(int i=0;i<=max;i++) M[i]=0;
    while (temp3!=NULL)
```

```

    {
        M[temp3->val]=1;
        temp3=temp3->next;
    }
    for(int i=0;i<=max;i++){
        if(M[i]==1){
            temp4->val=i;
            temp4=temp4->next;
        }
    }
    size=size/2;
}

void deleteFirstNode(struct node *head)
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\n Node deleted from the beginning ...");
    }
}

void delete(struct node *head)
{
    while (size!=0)
    {
        deleteFirstNode(head);
        size=size-1;
    }
}

void main(){
    int val;
    struct node * head = NULL;
    int n=1;
    while (n!=0)
    {
        printf("\n0.To exit\n 1.To Insert Element\n 2.Remove Bottom Half(S)\n");
        scanf("%d",&n);
        switch (n)
        {
            {
            case 0:
                n=0;
                break;
            case 1:
                printf("Enter the value to insert\n");

```

```

        scanf("%d",&val);
        Insert(&head,val);
        printf("\nAfter Operation\n");
        Display(head);
        break;
    case 2:
        Asc(head);
        Display(head);
        delete(head);
        printf("\nAfter Operation\n");
        Display(head);
        break;
    default:
        printf("Enter valid option");
        break;
    }
}
}

```

### **Conclusion/Observation:**

A Data Structure was formed to maintain a set S of n distinct integers, then two functions were written to facilitate Element Insertion and removal bottom half elements within time complexity of  $O[1]$  and  $O[n]$ , respectively.

### 3.6 Program Title:

Consider an  $n \times n$  matrix  $A = (a_{ij})$ , each of whose elements  $a_{ij}$  is a nonnegative real number and suppose that each row and column of  $A$  sums to an integer value. We wish to replace each element  $a_{ij}$  with either  $\lfloor a_{ij} \rfloor$  or  $\lceil a_{ij} \rceil$  without disturbing the row and column sums. Write a program by defining a user defined function that is used to produce the rounded matrix as described in the above example. Find out the time complexity of your algorithm/function.

#### Input/Output Screenshots:

RUN-1:

```
Enter the size of Matrix upto 5x5
4
Enter The values of array (rows followed by columns)
10.9 2.5 1.3 9.3
3.8 9.2 2.2 11.8
7.9 5.2 7.3 0.6
3.4 13.1 1.2 6.3

Entered Matrix
10.9    2.5    1.3    9.3
3.8     9.2    2.2   11.8
7.9     5.2    7.3    0.6
3.4    13.1    1.2    6.3

After Round off
11.0    3.0    1.0    9.0
4.0     9.0    3.0   11.0
8.0     5.0    7.0    1.0
3.0    13.0    1.0    7.0
```

#### Source code

```
#include <stdio.h>
#include <math.h>

float M[4][4];
float floatA[5][5];
float sum_row[5];
float sum_col[5];
float A[5][5];
int n;
void display(){
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
```



```

        printf("%.1f\t",A[i][j]);
    }
    printf("\n");
}
}
void FloatSum(){
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            floatA[i][j]=A[i][j]-(int)A[i][j];
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            sum_row[i]=(sum_row[i]+floatA[i][j]);
        }
        sum_row[i]=round(sum_row[i]);
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            sum_col[i]=(sum_col[i]+floatA[j][i]);
        }
        sum_col[i]=round(sum_col[i]);
    }
}
void algo(){
    printf("\n");
    for(int w=0;w<n;w++){
        for(int q=0;q<n;q++){
            if(sum_col[q]>0 && sum_row[w]>0){
                A[w][q]=ceil(A[w][q]);
                sum_col[q]=sum_col[q]-1;
                sum_row[w]=sum_row[w]-1;
            }
            else A[w][q]=floor(A[w][q]);
        }
    }
}
void main(){

    printf("\nEnter the size of Matrix upto 5x5\n");
    scanf("%d",&n);
    printf("Enter The values of array (rows followed by columns)\n");
    for(int i=0;i<n;i++){
        for (int j = 0; j < n; j++)

```

```

        {
            scanf("%f",&A[i][j]);
        }
    }
    printf("\nEntered Matrix\n");
    display();
    FloatSum();
    algo();
    printf("\nAfter Round off\n");
    display();
}

```

### **Conclusion/Observation:**

A matrix's elements were rounded off while maintaining the rows and columns sum. A UDF with time complexity  $O[n^2]$  was written to find the result. It was observed that multiple results are possible and the very first fit found was displayed by the written UDF.

## LAB - 4

# Divide and Conquer Method

### PROGRAM EXERCISE

## CONTENTS

Prog. No.	Program Title	Page No.
4.1	Write a program to search an element x in an array of n integers using binary search algorithm that uses divide and conquer technique. Find out the best case, worst case, and average case time complexities for different values of n and plot a graph of the time taken versus n. The n integers can be generated randomly, and x can be chosen randomly, or any element of the array or middle or last element of the array depending on type of time complexity analysis.	
4.2	Write a program to sort a list of n elements using the merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n and different nature of data (random data, sorted data, reversely sorted data) in the list. n is the user input and n integers can be generated randomly. Finally plot a graph of the time taken versus n.	
4.3	Write a program to use divide and conquer method to determine the time required to find the maximum and minimum element in a list of n elements. The data for the list can be generated randomly. Compare this time with the time taken by straight forward algorithm or brute force algorithm for finding the maximum and minimum element for the same list of n elements. Show the comparison by plotting a required graph for this problem.	
4.4	Write a program that uses a divide-and-conquer algorithm/user defined function for the exponentiation problem of computing $a^n$ where $a > 0$ and n is a positive integer. How does this algorithm compare with the brute-force algorithm in terms of number of multiplications made by both algorithms	

# PROGRAM SOLUTIONS

## 4.1 Program Title:

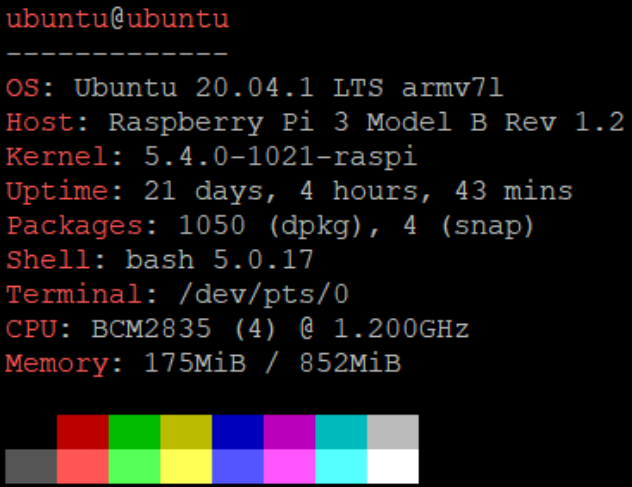
Write a program to search an element x in an array of n integers using binary search algorithm that uses divide and conquer technique. Find out the best case, worst case, and average case time complexities for different values of n and plot a graph of the time taken versus n. The n integers can be generated randomly, and x can be chosen randomly, or any element of the array or middle or last element of the array depending on type of time complexity analysis.

### Input/Output Screenshots:

#### RUN-1:


```
ubuntu@ubuntu:~/DAA$ neofetch
      .-/+oossssoo+/- .
    `:+ssssssssssssssssss+:`
      -+ssssssssssssssssssyyssss+-
    .ossssssssssssssssssdMMMMyssssso.
  /ssssssssssshdmNNmyNMMMMhssssss/
 +ssssssssshmydMMMMMMNddddyssssssst+
 /ssssssssshNMMMyhhyyyymNMMMNhssssss/
.ssssssssdMMMNhssssssssshNMMMdssssss.
+ssssshhhyNMMNysssssssssssyNMMMyssssst+
ossyNMMMNyMMhssssssssssshmmmhssssssso
ossyNMMMNyMMhssssssssssshmmmhssssssso
+ssssshhhyNMMNysssssssssssyNMMMyssssst+
.ssssssssdMMMNhssssssssshNMMMdssssss.
 /ssssssssshNMMMyhhyyyhdNMMMNhssssss/
 +ssssssssdmydMMMMMMNddddyssssssst+
 /ssssssssshdmNNNNmyNMMMMhssssss/
  .ossssssssssssssssssdMMMMyssssso.
    -+ssssssssssssssssssyyssss+-
      `:+ssssssssssssssssss+:`
        .-/+oossssoo+/- .

ubuntu@ubuntu:~/DAA$ ./a.out
avg      best      worst
0.000009 0.000007 0.000014
0.000005 0.000003 0.000009
0.000005 0.000003 0.000010
0.000005 0.000003 0.000012
0.000005 0.000004 0.000014
0.000006 0.000003 0.000013
0.000006 0.000003 0.000012
0.000005 0.000004 0.000012
0.000006 0.000003 0.000013
0.000006 0.000004 0.000013
ubuntu@ubuntu:~/DAA$
```



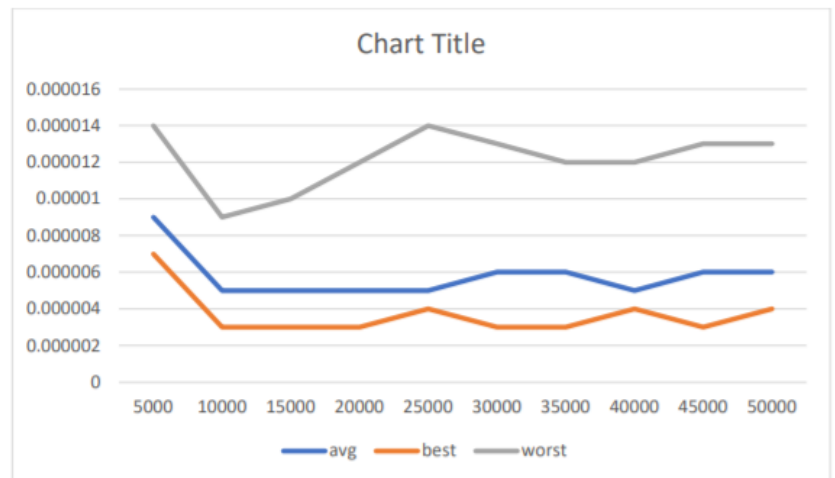
System information from neofetch:

- OS: Ubuntu 20.04.1 LTS armv7l
- Host: Raspberry Pi 3 Model B Rev 1.2
- Kernel: 5.4.0-1021-raspi
- Uptime: 21 days, 4 hours, 43 mins
- Packages: 1050 (dpkg), 4 (snap)
- Shell: bash 5.0.17
- Terminal: /dev/pts/0
- CPU: BCM2835 (4) @ 1.200GHz
- Memory: 175MiB / 852MiB



### Graph:

	avg	best	worst
5000	0.000009	0.000007	0.000014
10000	0.000005	0.000003	0.000009
15000	0.000005	0.000003	0.00001
20000	0.000005	0.000003	0.000012
25000	0.000005	0.000004	0.000014
30000	0.000006	0.000003	0.000013
35000	0.000006	0.000003	0.000012
40000	0.000005	0.000004	0.000012
45000	0.000006	0.000003	0.000013
50000	0.000006	0.000004	0.000013



### Source code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int randInt(int n)
{
    __time_t t;
    srand((unsigned)t * rand());
    return rand() % n;
}

void printArray(int arr[],int n){
    for(int i=0;i<n;++i){
        printf("%d ",arr[i]);
    }
    printf("\n");
}

void insertionSort(int arr[], int n, int p)
{
    for (int j = 1; j < n; ++j)
    {
        int temp = arr[j];
        int i = j - 1;
        while (arr[i] > temp && i >= 0)
        {
            arr[i + 1] = arr[i];
            i--;
        }
        arr[i + 1] = temp;
    }
    if (p == 1)
    {
```

```

    for (int i = 0; i < n; ++i)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
}

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

        if (arr[mid] == x)
        {
            return mid;
        }

        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}

void analyse(int n)
{
    int *arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
    {
        arr[i] = i+1;
    }

    // printArray(arr,n);

    clock_t t;
    double time_taken;
    int x, result;

    x = 0;
    t = clock();
    result = binarySearch(arr, 0, n - 1, x);
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    double count3 = time_taken;

    x = arr[randInt(n)];
    t = clock();

```

```

    result = binarySearch(arr, 0, n - 1, x);
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    double count1 = time_taken;

    x = arr[n / 2];
    t = clock();
    result = binarySearch(arr, 0, n - 1, x);
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    double count2 = time_taken;

    printf("%f %f %f \n", count1, count2, count3);
}

int main(void)
{
    printf("avg\t best\t worst\n");
    for (int i = 5001; i <= 50001; i += 5000)
    {
        analyse(i);
    }
    return 0;
}

```

### Conclusion/Observation

Using a lower CPU speed machine Raspberry Pi 3 we were able to get the accurate results.

We observe that for Binary Search, the best-case scenario, which is when the element is in middle of the array, the search takes 0.000003/0.000004s to complete hence almost equal to  $O(1)$  time for the range 5000 to 50000 elements. Progressively the avg and the worst-case scenario, when the element is at the extreme end of the array, the search takes a logarithmic increasing time curve hence satisfying the condition of  $O(\log n)$ .

## 4.2 Program Title:

Write a program to sort a list of n elements using the merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n and different nature of data (random data, sorted data, reversely sorted data) in the list. n is the user input and n integers can be generated randomly. Finally plot a graph of the time taken versus n.

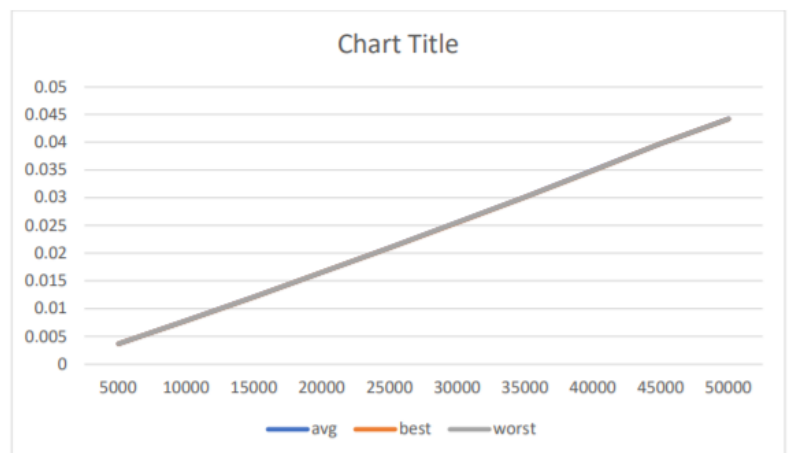
### Input/Output Screenshots:

#### RUN-1:

```
ubuntu@ubuntu:~/DAA$ ./a.out
n      avg      best      worst
5000   0.003693  0.003683  0.003687
10000  0.007838  0.007810  0.007817
15000  0.012121  0.012078  0.012077
20000  0.016557  0.016529  0.016520
25000  0.020980  0.020946  0.020948
30000  0.025552  0.025506  0.025530
35000  0.030111  0.030083  0.030068
40000  0.034931  0.034845  0.034857
45000  0.039770  0.039756  0.039744
50000  0.044232  0.044172  0.044191
ubuntu@ubuntu:~/DAA$
```

### Graph:

n	avg	best	worst
5000	0.003693	0.003683	0.003687
10000	0.007838	0.00781	0.007817
15000	0.012121	0.012078	0.012077
20000	0.016557	0.016529	0.01652
25000	0.02098	0.020946	0.020948
30000	0.025552	0.025506	0.02553
35000	0.030111	0.030083	0.030068
40000	0.034931	0.034845	0.034857
45000	0.03977	0.039756	0.039744
50000	0.044232	0.044172	0.044191





### Source code

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int randInt()
{
    __time_t t;
    srand((unsigned)t * rand());
    return rand() % 5000;
}

void rvereseArray(int arr[], int start, int end)
{
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
    }
}
```

```

        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void analyse(int n){
    int *arr1 = malloc(n*sizeof(int));
    for (int i = 0; i < n; ++i)
    {
        arr1[i] = randInt();
    }

    __clock_t t;
    double time_taken;
    int x,result;

    t = clock();
    mergeSort(arr1,0,n-1);
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    double count1 = time_taken;

    t = clock();
    mergeSort(arr1,0,n-1);
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    double count2 = time_taken;
}

```

```

    rvereseArray(arr1, 0, n-1);

    t = clock();
    mergeSort(arr1,0,n-1);
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    double count3 = time_taken;

    printf("%d\t%f %f %f \n", n,count1, count2, count3);
}

int main(void)
{
    printf("n\tavg\t best\t worst\n");
    for (int i = 5000; i <= 50000; i += 5000)
    {
        analyse(i);
    }
    return 0;
}

```

### Conclusion/Observation

For merge sort, different no of elements sorted in ascending order, descending order, and random order were taken for best case, worst case, and average case, respectively. After running the program multiple times for values ranging from 5000 to 50000, we observe that the time taken by merge sort for all three cases is almost equal hence concluding that merge sort takes  $O(n \log n)$  time for all cases. This can also be seen in the graph where all the 3 slopes overlap each-other.

#### 4.3 Program Title:

Write a program to use divide and conquer method to determine the time required to find the maximum and minimum element in a list of n elements. The data for the list can be generated randomly. Compare this time with the time taken by straight forward algorithm or brute force algorithm for finding the maximum and minimum element for the same list of n elements. Show the comparison by plotting a required graph for this problem.

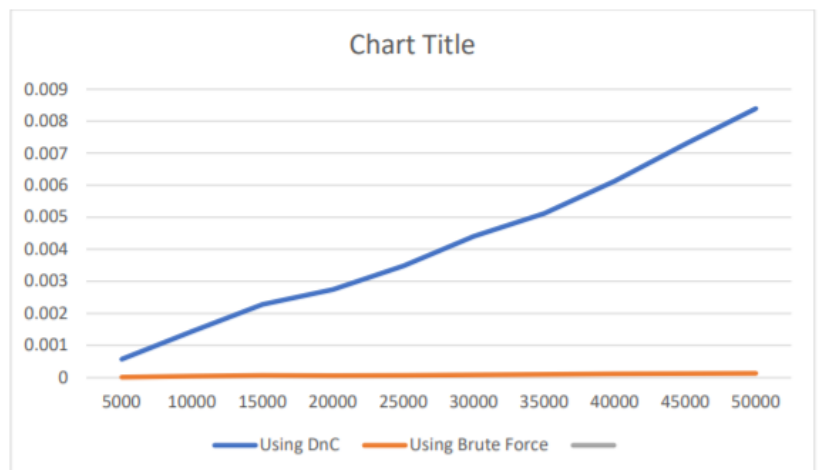
#### Input/Output Screenshots:

##### RUN-1:

S.No	Val N	Using DnC	Using Brute Force
1	5000	0.000577	0.000014
2	10000	0.001436	0.000039
3	15000	0.002275	0.000063
4	20000	0.002745	0.000056
5	25000	0.003480	0.000068
6	30000	0.004404	0.000083
7	35000	0.005117	0.000096
8	40000	0.006126	0.000109
9	45000	0.007286	0.000123
10	50000	0.008393	0.000135

#### Graph:

Val N	Using DnC	Using Brute Force
5000	0.000577	0.000014
10000	0.001436	0.000039
15000	0.002275	0.000063
20000	0.002745	0.000056
25000	0.00348	0.000068
30000	0.004404	0.000083
35000	0.005117	0.000096
40000	0.006126	0.000109
45000	0.007286	0.000123
50000	0.008393	0.000135



#### Source code:

```
/* C implementation QuickSort */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int randInt(){
    __time_t t;
    srand((unsigned)t * rand());
    return rand() % 1000;
}
```

```

}

void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high){
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++){
        if (arr[j] < pivot){
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high){
    if (low < high){
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size){
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void analyse(int n){
    int *arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
    {
        int num = randInt();
        arr[i] = num;
    }

    int mx,mn,i;
    clock_t t;
    double rt, bt;
    t = clock();
    mx = arr[0];
    mn = arr[0];

```

```

    for(i=1; i<n; i++){
        if(arr[i]>mx)
            mx = arr[i];
        if(arr[i]<mn)
            mn = arr[i];
    }
    t = clock() - t;
    bt = ((double)t) / CLOCKS_PER_SEC;

    t = clock();
    quickSort(arr, 0, n - 1);
    t = clock() - t;
    rt = ((double)t) / CLOCKS_PER_SEC;
    printf("%d\t%5d\t%10f\t%10f", n / 5000, n, rt, bt);
}

int main(){
    printf("S.No\tVal N\t Using DnC\t Using Brute Force\n");
    for (int i = 500; i <= 5000; i += 500){
        analyse(i);
        printf("\n");
    }
}

```

### Conclusion/Observation

We observe that to find the maximum and minimum element, the brute force method is way more effective than that of a divide and conquer algo as in brute force we only need to traverse the array one time thus giving the time complexity of  $O(n)$  where  $n$  is number of elements, whereas in divide and conquer method, the time taken is  $O(n \log n)$  which is exponentially higher as seen in the graph.

#### 4.4 Program Title:

Write a program that uses a divide-and-conquer algorithm/user defined function for the exponentiation problem of computing  $a^n$  where  $a > 0$  and  $n$  is a positive integer. How does this algorithm compare with the brute-force algorithm in terms of number of multiplications made by both algorithms?

#### Input/Output Screenshots:

##### RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab4_0826$  
Soln by Divide and conquer :  
Ans: 1073741824  
No. of Multiplications: 5  
Soln by Brute Force :  
Ans: 1073741824  
No. of Multiplications: 30
```

##### RUN-2:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab4_0826$  
Soln by Divide and conquer :  
Ans: 16777216  
No. of Multiplications: 4  
Soln by Brute Force :  
Ans: 16777216  
No. of Multiplications: 12
```

#### Source code

```
#include<stdio.h>  
int count_divide_and_conquer = 0;  
int count_naive = 0;  
unsigned long long int divide_and_conquer_Power(int a,int n){  
    ++count_divide_and_conquer;  
    if(n == 1) return a;  
    if(n == 0) return 1;  
    int number = divide_and_conquer_Power(a,n/2);  
    if(n&1)  
        return number * a * number;  
    else  
        return number * number;  
}  
unsigned long long int naive_Power(int a,int n){  
    if (n==0) return 1;  
    if (n==1) return a;  
    int num = a;  
    ++count_naive;  
    for(int i=1;i<n;i++){  
        ++count_naive;  
        num*=a;  
    }  
    return num;  
}
```

```

int main(){
    int a = 4;
    int b = 12;
    unsigned long long int pow1 = divide_and_conquer_Power(a,b);
    unsigned long long int pow2 = naive_Power(a,b);
    printf("Soln by Divide and conquer :\n");
    printf("Ans: %lli\nNo. of Multiplications: %d\n",pow1,count_divide_and_con-
quer);
    printf("Soln by Brute Force :\n");
    printf("Ans: %lli\nNo. of Multiplications: %d\n",pow2,count_naive);
}

```

### **Conclusion/Observation:**

The divide and conquer algorithm takes lesser number of multiplication compared to brute force method hence is more efficient.



# LAB - 5

(Building a heap, Heap Sort algorithm, Min Priority Queue, Max Priority queue)

## Heap & Priority Queues

### PROGRAM EXERCISE

### CONTENTS

Prog. No.	Program Title	Page No.
5.1	<p>Write a menu (given as follows) driven program to sort an array of n integers in ascending order by heap sort algorithm and perform the operations on max heap.</p> <p>Determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the array to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.</p> <p>MAX-HEAP &amp; PRIORITY QUEUE MENU</p> <ol style="list-style-type: none"><li>0. Quit</li><li>1. n Random numbers=&gt;Array</li><li>2. Display the Array</li><li>3. Sort the Array in Ascending Order by using Max-Heap Sort technique</li><li>4. Sort the Array in Descending Order by using any algorithm</li><li>5. Time Complexity to sort ascending of random data</li><li>6. Time Complexity to sort ascending of data already sorted in ascending order</li><li>7. Time Complexity to sort ascending of data already sorted in descending order</li><li>8. Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending &amp; Random Data) in Tabular form for values n=5000 to 50000, step=5000</li><li>9. Extract largest element</li><li>10. Replace value at a node with new value</li><li>11. Insert a new element</li><li>12. Delete an element</li></ol>	
5.2	<p>Similar to above program no.5.1, write a menu driven program to sort an array of n integers in descending order by heap sort algorithm.</p> <p>Hints: Use min heap and accordingly change the menu options.</p>	

### PROGRAM SOLUTIONS

#### **5.1 Program Title:**

Write a menu (given as follows) driven program to sort an array of n integers in ascending order by heap sort algorithm and perform the operations on max heap.

Determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the array to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

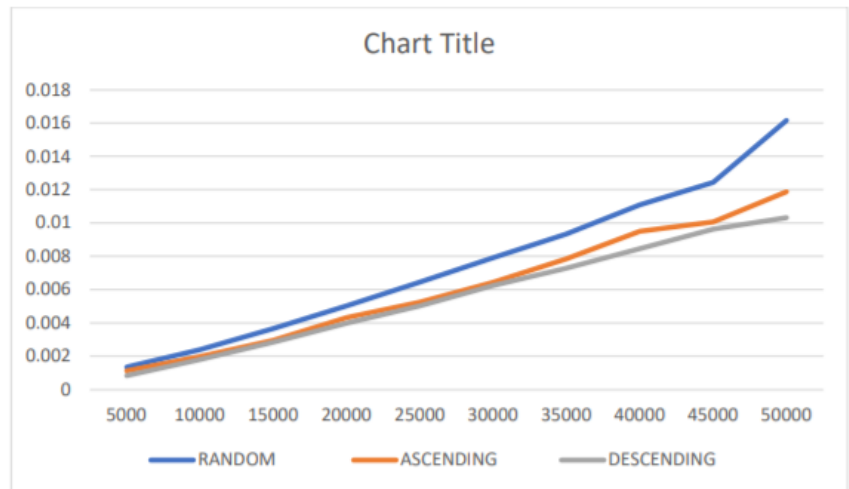
### Input/Output Screenshots:

#### RUN-1:

```
0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Max-Heap Sort
   technique
4. Sort the Array in Descending Order by using any algorithm
5. Time Complexity to sort ascending of random data
6. Time Complexity to sort ascending of data already sorted in
   ascending order
7. Time Complexity to sort ascending of data already sorted in
   descending order
8. Time Complexity to sort ascending all Cases (Data Ascending,
   Data in Descending & Random Data) in Tabular form for
   values n=5000 to 50000, step=5000
9. Extract largest element
10. Replace value at a node with new value
11. Insert a new element
12. Delete an element
Input Choice
8
RANDOM          ASCENDING          DESCENDING
0.001353        0.001133          0.000839
0.002391        0.001968          0.001802
0.003663        0.002957          0.002847
0.005023        0.004313          0.003977
0.006450        0.005251          0.005024
0.007909        0.006441          0.006237
0.009344        0.007839          0.007289
0.011090        0.009494          0.008454
0.012434        0.010057          0.009627
0.016167        0.011875          0.010327
0. Quit
```

## Graph:

N	RANDOM	ASCENDING	DESCENDING
5000	0.001353	0.001133	0.000839
10000	0.002391	0.001968	0.001802
15000	0.003663	0.002957	0.002847
20000	0.005023	0.004313	0.003977
25000	0.00645	0.005251	0.005024
30000	0.007909	0.006441	0.006237
35000	0.009344	0.007839	0.007289
40000	0.01109	0.009494	0.008454
45000	0.012434	0.010057	0.009627
50000	0.016167	0.011875	0.010327



## Source code:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

int partition(int arr[], int low, int high){
    int pivot = arr[high]; // pivot
    int i = (low - 1);      // Index of smaller element
    for (int j = low; j <= high - 1; j++){
        // If current element is smaller than the pivot
        if (arr[j] > pivot){
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void sortDescending(int arr[], int low, int high){
    if (low < high){
        /* pi is partitioning index, arr[p] is now
        at right place */
    }
}
```

```

        int pi = partition(arr, low, high);
        // Separately sort elements before
        // partition and after partition
        sortDescending(arr, low, pi - 1);
        sortDescending(arr, pi + 1, high);
    }
}

void printArr(int arr[], int n){
    printf("[ ");
    for (int i = 0; i < n; ++i)
        printf("%d, ", arr[i]);
    printf("]\n");
}

void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

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

    for (int i = n - 1; i > 0; i--)
    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

void buildHeap(int arr[], int n)
{
    // Index of last non-leaf node

```

```

    int startIdx = (n / 2) - 1;

    // Perform reverse level order traversal
    // from last non-leaf node and heapify
    // each node
    for (int i = startIdx; i >= 0; i--)
    {
        heapify(arr, n, i);
    }
}

void fillRandom(int arr[], int n)
{
    srand(0);
    for (int i = 0; i < n; i++)
        arr[i] = rand() % n;
}

void analyze()
{
    int arr[50000];
    printf("RANDOM\t");
    printf("\tASCENDING\t");
    printf("DESCENDING\n");
    for (int n = 5000; n <= 50000; n += 5000)
    {
        fillRandom(arr, n);
        //Start timer for random
        clock_t t_random = clock();
        heapSort(arr, n);
        //Stop timer for random
        t_random = clock() - t_random;

        //Start timer for data in ascending
        clock_t t_ascending = clock();
        heapSort(arr, n);
        //Stop timer for sorted data
        t_ascending = clock() - t_ascending;

        //Sort in descending
        sortDescending(arr, 0, n);
        //Start timer for descending sort
        clock_t t_descending = clock();
        heapSort(arr, n);
        //Stop timer for descending sort
        t_descending = clock() - t_descending;

        printf("%f\t%f\t%f\n", (double)t_random / CLOCKS_PER_SEC, (double)t_ascending / CLOCKS_PER_SEC, (double)t_descending / CLOCKS_PER_SEC);
    }
}

```

```

}

int heap_extract_max(int arr[], int n)
{
    if (n < 1)
        return -1;
    int max = arr[0];
    arr[0] = arr[n - 1];
    heapify(arr, n, 0);
    printArr(arr, n - 1);
    return max;
}

int main()
{
    clock_t t_asc;
    double tt_asc;
    int i, j, temp;
    int arr[100000];
    int n;
    int ch = 0;
    do
    {
        printf("\n
0. Quit\n\
1. n Random numbers=>Array\n\
2. Display the Array\n\
3. Sort the Array in Ascending Order by using Max-Heap Sort\n\
technique\n\
4. Sort the Array in Descending Order by using any algorithm\n\
5. Time Complexity to sort ascending of random data\n\
6. Time Complexity to sort ascending of data already sorted in\n\
ascending order\n\
7. Time Complexity to sort ascending of data already sorted in\n\
descending order\n\
8. Time Complexity to sort ascending all Cases (Data Ascending,\n\
Data in Descending & Random Data) in Tabular form for\n\
values n=5000 to 50000, step=5000\n\
9. Extract largest element\n\
10. Replace value at a node with new value\n\
11. Insert a new element\n\
12. Delete an element\nInput Choice\n");

        scanf("%d", &ch);

        switch (ch)
        {
            case 1:
            {
                printf("Number of elements?\n");

```

```

        scanf("%d", &n);
        fillRandom(arr, n);
        buildHeap(arr,n);
    }
    break;

    case 2:
        printArr(arr, n);
        break;

    case 3:
        heapSort(arr, n);
        printArr(arr, n);
        break;

    case 4:
        sortDescending(arr, 0, n);
        break;

    case 5:
    {
        fillRandom(arr, n);

        clock_t start = clock();
        heapSort(arr, n);
        printf("Time taken to sort random data = %f", (double)(clock() - start) / CLOCKS_PER_SEC);
    }
    break;

    case 6:
    {
        fillRandom(arr, n);
        heapSort(arr, n);

        //Start the clock
        clock_t start = clock();
        heapSort(arr, n);
        //Stop the clock and print time
        printf("Time taken to sort already sorted data is = %f", (double)
(clock() - start) / CLOCKS_PER_SEC);
    }
    break;

    case 7:
    {
        sortDescending(arr, 0, n);
        //Start the clock
        clock_t start = clock();
        heapSort(arr, n);

```

```

        //Stop the clock and print time
        printf("Time taken to sort already descending data is = %f", (double)
(clock() - start) / CLOCKS_PER_SEC);
    }
    break;
    case 8:
        analyze();
        break;

    case 9:
    {
        buildHeap(arr, n);
        int max = heap_extract_max(arr, n);
        printf("Max element is %d\n", max);
    }
    break;

    case 10:
    {
        int index;
        printf("Enter index of value to be changed\n");
        scanf("%d", &index);
        printf("Enter new value\n");
        int num;
        scanf("%d", &num);
        arr[index] = num;
        buildHeap(arr, n);
        printArr(arr, n);
    }
    break;

    case 11:
    {
        int num;
        printf("Enter number to be inserted\n");
        scanf("%d", &num);
        printf(";;\n");
        arr[n + 1] = num;
        buildHeap(arr, n + 1);
        printArr(arr, n + 1);
    }
    break;

    case 12:
    {
        int index;
        printf("Enter index of element to delete\n");
        scanf("%d", &index);
        arr[index] = __INT32_MAX__ + 1;
        n--;
    }

```



```
        buildHeap(arr, n);
        printArr(arr, n);
    }
    break;

    default:
        break;
}

} while (ch);
}
```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.

## 5.2 Program Title:

Similar to above program no.5.1, write a menu driven program to sort an array of n integers in descending order by heap sort algorithm.

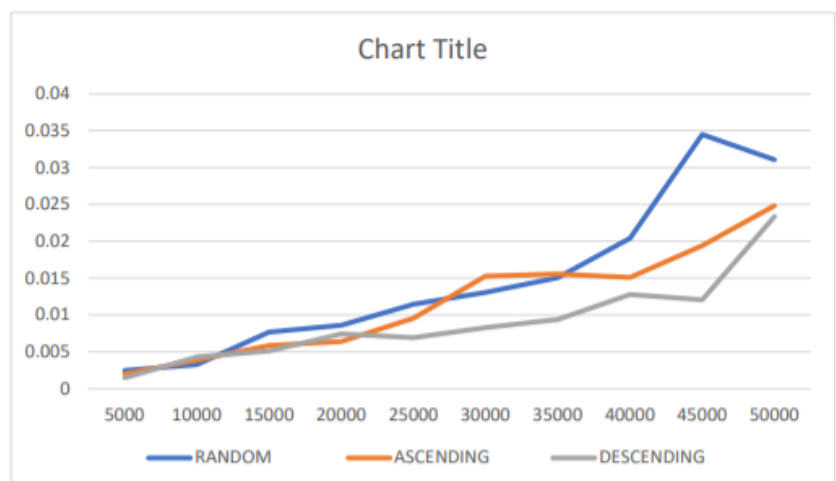
### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab5_0930$ cd "/home/sahil/sem5_lab/DAA/lab5_0930"
0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order by using Min-Heap Sort technique
4. Sort the Array in Descending Order by using any algorithm
5. Time Complexity to sort ascending of random data
6. Time Complexity to sort ascending of data already sorted in ascending order
7. Time Complexity to sort ascending of data already sorted in descending order
8. Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
9. Extract largest element
10. Replace value at a node with new value
11. Insert a new element
12. Delete an element
Input Choice
8
RANDOM          ASCENDING      DESCENDING
0.002466        0.002018        0.001492
0.003264        0.003867        0.004285
0.007658        0.005854        0.005095
0.008609        0.006366        0.007427
0.011445        0.009540        0.006925
0.013056        0.015250        0.008288
0.015036        0.015546        0.009392
0.020410        0.015085        0.012770
0.034510        0.019412        0.012043
0.031072        0.024817        0.023386
```

### GRAPH:

N	RANDOM	ASCENDING	DESCENDING
5000	0.002466	0.002018	0.001492
10000	0.003264	0.003867	0.004285
15000	0.007658	0.005854	0.005095
20000	0.008609	0.006366	0.007427
25000	0.011445	0.009540	0.006925
30000	0.013056	0.015250	0.008288
35000	0.015036	0.015546	0.009392
40000	0.020410	0.015085	0.012770
45000	0.034510	0.019412	0.012043
50000	0.031072	0.024817	0.023386



### Source code:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

int partition(int arr[], int low, int high){
    int pivot = arr[high]; // pivot
    int i = (low - 1);      // Index of smaller element
    for (int j = low; j <= high - 1; j++){
        // If current element is smaller than the pivot
        if (arr[j] > pivot){
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void sortDescending(int arr[], int low, int high){
    if (low < high){
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);
        // Separately sort elements before
        // partition and after partition
        sortDescending(arr, low, pi - 1);
        sortDescending(arr, pi + 1, high);
    }
}

void printArr(int arr[], int n){
    printf("[ ");
    for (int i = 0; i < n; ++i)
        printf("%d, ", arr[i]);
    printf("]\n");
}

void heapify(int arr[], int n, int i)
{
    int smallest = i;
```

```

    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] < arr[smallest])
        smallest = l;

    if (r < n && arr[r] < arr[smallest])
        smallest = r;

    if (smallest != i) {
        swap(&arr[i], &arr[smallest]);

        heapify(arr, n, smallest);
    }
}

void heapSort(int arr[], int n){
    for (int i = n / 2 - 1; i >= 0; i--){
        heapify(arr, n, i);
    }
    for (int i = n - 1; i > 0; i--){
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

void buildHeap(int arr[], int n){
    // Index of last non-leaf node
    int startIdx = (n / 2) - 1;
    // Perform reverse level order traversal
    // from last non-leaf node and heapify
    // each node
    for (int i = startIdx; i >= 0; i--){
        heapify(arr, n, i);
    }
}

void fillRandom(int arr[], int n){
    srand(0);
    for (int i = 0; i < n; i++){
        arr[i] = rand() % n;
    }
}

void analyze(){
    int arr[50000];
    printf("RANDOM\t");
    printf("\tASCENDING\t");
    printf("DESCENDING\n");
    for (int n = 5000; n <= 50000; n += 5000){
        fillRandom(arr, n);
        //Start timer for random
        clock_t t_random = clock();
    }
}

```

```

        heapSort(arr, n);
        //Stop timer for random
        t_random = clock() - t_random;

        //Start timer for data in ascending
        clock_t t_ascending = clock();
        heapSort(arr, n);
        //Stop timer for sorted data
        t_ascending = clock() - t_ascending;

        //Sort in descending
        sortDescending(arr, 0, n);
        //Start timer for descending sort
        clock_t t_descending = clock();
        heapSort(arr, n);
        //Stop timer for descending sort
        t_descending = clock() - t_descending;

        printf("%f\t%f\t%f\n", (double)t_random / CLOCKS_PER_SEC, (double)t_ascending / CLOCKS_PER_SEC, (double)t_descending / CLOCKS_PER_SEC);
    }
}

int heap_extract_min(int arr[], int n){
    if (n < 1)
        return -1;
    int max = arr[0];
    arr[0] = arr[n - 1];
    heapify(arr, n, 0);
    printArr(arr, n - 1);
}

int main(){
    clock_t t_asc;
    double tt_asc;
    int i, j, temp;
    int arr[100000];
    int n;
    int ch = 0;
    do{
        printf("\n
0. Quit\n\
1. n Random numbers=>Array\n\
2. Display the Array\n\
3. Sort the Array in Ascending Order by using Min-Heap Sort\n\
technique\n\
4. Sort the Array in Descending Order by using any algorithm\n\
5. Time Complexity to sort ascending of random data\n\
6. Time Complexity to sort ascending of data already sorted in\n\
ascending order\n\

```

7. Time Complexity to sort ascending of data already sorted in\n\ndescending order\n\  
 8. Time Complexity to sort ascending all Cases (Data Ascending,\n\  
 Data in Descending & Random Data) in Tabular form for\n\  
 values n=5000 to 50000, step=5000\n\  
 9. Extract largest element\n\  
 10. Replace value at a node with new value\n\  
 11. Insert a new element\n\  
 12. Delete an element\nInput Choice\n");

```
scanf("%d", &ch);
switch (ch){
case 1:{
    printf("Number of elements?\n");
    scanf("%d", &n);
    fillRandom(arr, n);
}
break;

case 2:
    printArr(arr, n);
    break;

case 3:
    heapSort(arr, n);
    printArr(arr, n);
    break;

case 4:
    sortDescending(arr, 0, n);
    break;

case 5:{
    fillRandom(arr, n);
    clock_t start = clock();
    heapSort(arr, n);
    printf("Time taken to sort random data = %f", (double)(clock() - start) / CLOCKS_PER_SEC);
}
break;

case 6:{
    fillRandom(arr, n);
    heapSort(arr, n);

    //Start the clock
    clock_t start = clock();
    heapSort(arr, n);
    //Stop the clock and print time
```

```

        printf("Time taken to sort already sorted data is = %f", (double)
(clock() - start) / CLOCKS_PER_SEC);
    }
    break;

    case 7:{
        sortDescending(arr, 0, n);
        //Start the clock
        clock_t start = clock();
        heapSort(arr, n);
        //Stop the clock and print time
        printf("Time taken to sort already descending data is = %f", (double)
(clock() - start) / CLOCKS_PER_SEC);
    }
    break;
    case 8:
        analyze();
        break;

    case 9:{
        buildHeap(arr, n);
        int min = heap_extract_min(arr, n);
        printf("min element is %d\n", min);
    }
    break;

    case 10:{
        int index;
        printf("Enter index of value to be changed\n");
        scanf("%d", &index);
        printf("Enter new value\n");
        int num;
        scanf("%d", &num);
        arr[index] = num;
        buildHeap(arr, n);
        printArr(arr, n);
    }
    break;

    case 11:{
        int num;
        printf("Enter number to be inserted\n");
        scanf("%d", &num);
        printf(";;\n");
        arr[n + 1] = num;
        buildHeap(arr, n + 1);
        printArr(arr, n + 1);
    }
    break;

```

```
    case 12:{
        int index;
        printf("Enter index of element to delete\n");
        scanf("%d", &index);
        arr[index] = __INT32_MAX__;
        n--;
        buildHeap(arr, n);
        printArr(arr, n);
    }
    break;

    default:
        break;
}

} while (ch);
}
```

### **Conclusion/Observation**

write here your conclusion/inference/final comment.



## LAB - 6

# Greedy Methods

(Fractional knapsack problem, Activity selection problem, Huffman's code)

## PROGRAM EXERCISE

### CONTENTS

Prog. No.	Program Title	Page No.
6.1	Write a program to implementation of Fractional Knapsack algorithm.	
6.2	Write a program to implement the activity-selection problem stated as follows: You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. Example: Consider the following 6 activities (0, 1, 2, 3, 4, 5). start[] = {1, 3, 0, 5, 8, 5}; finish[] = {2, 4, 6, 7, 9, 9}; The maximum set of activities that can be executed by a single person is {0, 1, 3, 4}.	
6.3	Write a program to implement the file or code compression using Huffman's algorithm.	

## PROGRAM SOLUTIONS

### 6.1 Program Title:

Write a program to implementation of Fractional Knapsack algorithm.

#### Input/Output Screenshots:

RUN-1:

```
sahil@Sahil-Probook:~/sem5_lab/DAA/lab6_1007$ cd '  
Add object 2 (₹500000, 5 weight). Space left: 55.  
Add object 1 (₹5000, 50 weight). Space left: 5.  
Add 0% (₹5000, 1000 weight) of object 3.  
Filled the knapsack with items worth ₹505025.00.
```

#### Source code

```
#include <stdio.h>  
  
int weight[] = {50,5,1000};  
int price[] = {5000,500000,5000};  
  
void Greedy_Knapsack(int M,int n) {  
    int current_value;  
  
    //Total value of items in bag  
    float total_value;  
    int i, maximum;  
  
    //Check if item has been picked up  
    int used[n];  
    for (i = 0; i < n; ++i)  
        used[i] = 0;  
    current_value = M;  
    while (current_value > 0) {  
        maximum = -1;  
        for (i = 0; i < n; ++i)  
            if ((used[i] == 0) &&  
                ((maximum == -1) || (price[i]*1.0/weight[i] > price[maxi-  
mum]*1.0/weight[maximum])))  
                maximum = i;  
        //Set that item has been picked up  
  
        used[maximum] = 1;  
        current_value -= weight[maximum];  
        total_value += price[maximum];  
  
        if (current_value >= 0)  
            printf("Add object %d (₹%d, %d weight). Space left: %d.\n", maximum +  
1, price[maximum], weight[maximum], current_value);  
        else {
```

```

        printf("Add %d%% (₹%d, %d weight) of object %d.\n", (int)((1 + current_value*1.0/weight[maximum]) * 100), price[maximum], weight[maximum], maximum + 1);

        total_value -= price[maximum];
        total_value += (1 + (float)current_value/weight[maximum]) * price[maximum];
    }
}

printf("Filled the knapsack with items worth ₹%.2f.\n", total_value);
}

int main() {
    //Max weight
    int M = 60;
    //Number of items
    int n = 3;
    Greedy_Knapsack(M,n);
    return 0;
}

```

### Conclusion/Observation

In Fractional Knapsack, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.

A brute-force solution would be to try all possible subset with all different fraction but that will be too much time taking.

An efficient solution is to use Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.

## 6.2 Program Title:

Write a program to implement the activity-selection problem stated as follows:

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. Example: Consider the following 6 activities (0, 1, 2, 3, 4, 5). start[] = {1, 3, 0, 5, 8, 5}; finish[] = {2, 4, 6, 7, 9, 9}; The maximum set of activities that can be executed by a single person is {0, 1, 3, 4}.

### Input/Output Screenshots:

RUN-1:

```
sahil@sahil-Probook:~/sem5_lab/DAA/lab6_1007$  
activity 0  
activity 1  
activity 3  
activity 4
```

### Source code

```
#include<stdio.h>  
  
void greedy_activity_selector(int s[],int f[],int n){  
    int i = 0; printf("activity 0\n");  
    for(int m = 1;m<n;m++){  
        if(s[m] >= f[i]){  
            printf("activity %d\n",m);  
            i = m;  
        }  
    }  
}  
  
int main(){  
    int start[] = {1,3,0,5,8,5};  
    int finish[] = {2,4,6,7,9,9};  
    greedy_activity_selector(start,finish,6);  
}
```

### Conclusion/Observation

For the given activities that uses a single resource individually, Activity selection problem find outs the maximum subset of mutually compatible activities (non-conflicting). Each activity is assumed with a start time and an end time. The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

### 6.3 Program Title:

Write a program to implement the file or code compression using Huffman's algorithm.

#### Input/Output Screenshots:

##### RUN-1:

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

rohan@Laptop ~/daa/lab6
$ gcc 0603.c
rohan@Laptop ~/daa/lab6
$ ./a.out
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
rohan@Laptop ~/daa/lab6
$
```

##### RUN-2:

```
rohan@Laptop ~/daa/lab6
$ gcc 0603.c
rohan@Laptop ~/daa/lab6
$ ./a.out
m: 00
e: 0100
c: 01010
d: 01011
f: 011
r: 1
```

## Source code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TREE_HT 100

struct MinHeapNode{
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};

struct MinHeap{
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

struct MinHeapNode* newNode(char data, unsigned freq){
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

struct MinHeap* createMinHeap(unsigned capacity){
    struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct MinHeapNode));
    return minHeap;
}

void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b){
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* minHeap, int idx){
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;
    if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;
    if (smallest != idx){
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
    }
}
```

```

        minHeapify(minHeap, smallest);
    }
}

int isSizeOne(struct MinHeap* minHeap){
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap){
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode){
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq){
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap* minHeap){
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

void printArr(int arr[], int n){
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}

int isLeaf(struct MinHeapNode* root){
    return !(root->left) && !(root->right);
}

struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size){
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)

```

```

{
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap)){
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}

void printCodes(struct MinHeapNode* root, int arr[], int top){
    if (root->left){
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right){
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)){
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

void HuffmanCodes(char data[], int freq[], int size){
    struct MinHeapNode* root = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}

int main(){
    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };
    int size = sizeof(arr) / sizeof(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}

```

### Conclusion/Observation

We can observe that the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.



## LAB - 7

# Dynamic Programming

(Matrix Chain Multiplication, Longest Common Subsequence)

### PROGRAM EXERCISE

### CONTENTS

Prog. No.	Program Title	Page No.
7.1	Finding longest common subsequence using dynamic programming.	

## PROGRAM SOLUTIONS

### 7.1 Program Title:

Finding longest common subsequence using dynamic programming.

#### Input/Output Screenshots:

##### RUN-1:

```
Enter the string 1
BCAB
Enter the string 2
ABDC

0      0      0      0      0
0      0^     1\     1<     1<
0      0^     1^     1^     2\
0      1\     1^     1^     2^
0      1^     2\     2<     2^

str1 : BCAB
str2 : ABDC
LCS: AB
```

##### RUN-2:

```
Enter the string 1
ABCDEF
Enter the string 2
FBCADE

0      0      0      0      0      0      0
0      0^     0^     0^     1\     1<     1<
0      0^     1\     1<     1^     1^     1^
0      0^     1^     2\     2<     2<     2<
0      0^     1^     2^     2^     3\     3<
0      0^     1^     2^     2^     3^     4\
0      1\     1^     2^     2^     3^     4^

str1 : ABCDEF
str2 : FBCADE
LCS: BCDE
```

## Source code

```
#include <stdio.h>
#include <string.h>

void lcstable(char str1[],char str2[]){
    int m = strlen(str1);
    int n = strlen(str2);
    char B[m+1][n+1];
    int C[20][20];
    for (int i = 0; i <= m; i++)
        C[i][0] = 0;
    for (int i = 0; i <= n; i++)
        C[0][i] = 0;

    // Building the mtrix in bottom-up way
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++) {
            if (str1[i - 1] == str2[j - 1]) {
                C[i][j] = C[i - 1][j - 1] + 1;
                B[i][j]='\\';
            }
            else if (C[i - 1][j] >= C[i][j - 1]) {
                C[i][j] = C[i - 1][j];
                B[i][j]='^';
            }
            else {
                C[i][j] = C[i][j - 1];
                B[i][j]='<';
            }
        }

    int index = C[m][n];
    char lcsAlgo[index + 1];
    lcsAlgo[index] = '\\0';
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (str1[i - 1] == str2[j - 1]) {
            lcsAlgo[index - 1] = str1[i - 1];
            i--;
            j--;
            index--;
        }
        else if (C[i - 1][j] > C[i][j - 1])
            i--;
        else
            j--;
    }
}
```

```

for (int i = 0; i <=m; i++)
{
    for (int j = 0; j <=n; j++)
    {
        if(i==0 || j==0) printf("%d\t",C[i][j]);
        else printf("%d%c\t",C[i][j],B[i][j]);
    }
    printf("\n");
}
printf("str1 : %s \nstr2 : %s \n", str1, str2);
printf("LCS: %s\n", lcsAlgo);
}
void main(){
    char str1[50],str2[50];
    printf("Enter the string 1\n");
    scanf("%s",str1);
    printf("Enter the string 2\n");
    scanf("%s",str2);
    lcstable(str1,str2);
}

```

### Conclusion/Observation

Dynamic programming is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

We use the method of dynamic programming to find the longest common subsequence (LCS), defined as the longest subsequence that is common to all the given sequences, provided that the elements of the subsequence are not required to occupy consecutive positions within the original sequences.

The time taken by a dynamic approach is the time taken to fill the table (i.e.  $O(mn)$ ).

.