NAME	MAAZ SHAIKH
UID	2021700059
CLASS	SY CSE(DS)
BATCH	D

Exp1-A

AIM	To implement the various functions e.g. linear, non-linear, quadratic, exponential etc.
PROGRAM	

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

float func1(int n);//(3/2)^n function
float func2(int n);// n cube function
int func3(int n);// n(linear) function
float func4(int n);// 2^n function
double func5(double n);//logn function
int func6(int n);//2^2^n+1 function
double func7(double n);//n*logn function
double func8(double n);//log(log(n)) function
double func9(double n);//2^logn function
double func10(double n);//n^log(log(n)) function
double factorial(int n);//Factorial function



float func1(int n){
    return pow(1.5,n);
}

float func2(int n){
    return n*n*n;
}

int func3(int n){
    return n;
}

float func4(int n){
    return pow(2,n);
}

double func5(double n){
    return (log(n))/log(2);
}

int func6(int n){
    return pow(2,pow(2,n+1));
}
```

```
double func7(double n){
    return n*(log(n));
}

double func8(double n){
    return log(log(n));
}

double func9(double n){
    return pow(2,log(n));
}

double func10(double n){
    return pow(n,log(log(n)));

}

double factorial(int n){
    if(n==0 || n==1)
        return 1;
    else
        return n*factorial(n-1);

}

int main(){
    int n;
    float output;
    int choice;


    do{
    printf("\nEnter 1 for (3/2)^n function\nEnter 2 for
n^3(cubic) function\nEnter 3 for n(linear) function\nEnter 4
for 2^n function\nEnter 5 for lgn function\nEnter 6 for
2^2^n+1 function\nEnter 7 for nlogn function\nEnter 8 for
loglogn function\nEnter 9 for 2^logn function\nEnter 10 for
n^loglogn function\nEnter 11 for n! function\nEnter 0 for
exit\n");
    scanf("%d",&choice);

    switch(choice){
        case 1:printf("Input\toutput\n");
               for(int n=0;n<=100;n+=10){
                     output=func1(n);
                     printf("%d\t%0.2f\n",n,output);
                  }
```

```
            break;
        case 2:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func2(n);
                printf("%d\t%0.2f\n",n,output);
            }
            break;
        case 3:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func3(n);
                printf("%d\t%0.2f\n",n,output);
            }
             break;
        case 4:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func4(n);
                printf("%d\t%0.2f\n",n,output);
            }
             break;
       case 5:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func5(n);
                printf("%d\t%0.2f\n",n,output);
            }
            break;
       case 6:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func6(n);
                printf("%d\t%0.2f\n",n,output);
            }
            break;
       case 7:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func7(n);
                printf("%d\t%0.2f\n",n,output);
            }
            break;
        case 8:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func8(n);
                printf("%d\t%0.2f\n",n,output);
            }
            break;
        case 9:printf("Input\toutput\n");
            for(int n=0;n<=100;n+=10){
                output=func9(n);
                printf("%d\t%0.2f\n",n,output);
            }
```

```
                break;
        case 10:printf("Input\toutput\n");
                for(int n=0;n<=100;n+=10){
                     output=func10(n);
                    printf("%d\t%0.2f\n",n,output);
                   }
                break;
        case 11:printf("Input\toutput\n");
                for(int i=0;i<=20;i+=2){
                   output=factorial(i);
                   printf("%d\t%0.2f\n",i,output);
                }
                break;
       default: printf("program finished");
                break;

    }}while(choice!=0);
    return 0;



}
```

OUTPUT

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
1
Input   output
0       1.00
10      57.67
20      3325.26
30      191751.06
40      11057332.00
50      637621504.00
60      36768468992.00
70      2120255143936.00
80      122264599134208.00
90      7050392827330560.00
100     406561191922499580.00
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
2
Input   output
0       0.00
10      1000.00
20      8000.00
30      27000.00
40      64000.00
50      125000.00
60      216000.00
70      343000.00
80      512000.00
90      729000.00
100     1000000.00
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
3
Input   output
0       0.00
10      10.00
20      20.00
30      30.00
40      40.00
50      50.00
60      60.00
70      70.00
80      80.00
90      90.00
100     100.00
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
4
Input   output
0       1.00
10      1024.00
20      1048576.00
30      1073741824.00
40      1099511627776.00
50      1125899906842624.00
60      1152921504606847000.00
70      1180591620717411300000.00
80      1208925819614629200000000.00
90      1237940039285380300000000000.00
100     1267650600228229400000000000000.00
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
5
Input   output
0       -1.#J
10      3.32
20      4.32
30      4.91
40      5.32
50      5.64
60      5.91
70      6.13
80      6.32
90      6.49
100     6.64
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
6
Input   output
0       4.00
10      -2147483648.00
20      -2147483648.00
30      -2147483648.00
40      -2147483648.00
50      -2147483648.00
60      -2147483648.00
70      -2147483648.00
80      -2147483648.00
90      -2147483648.00
100     -2147483648.00
```

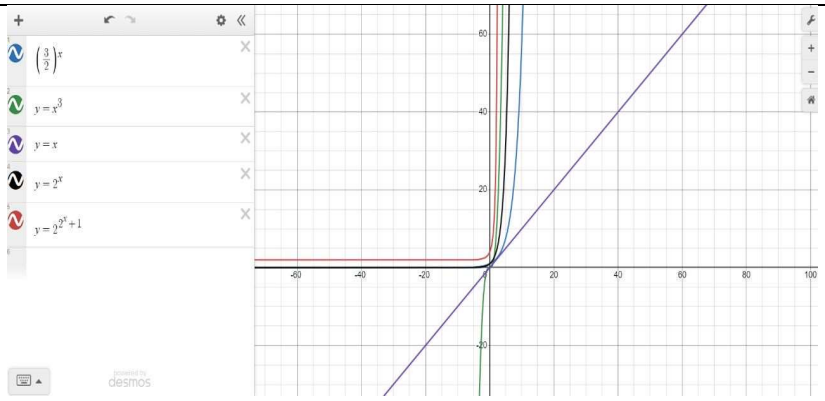
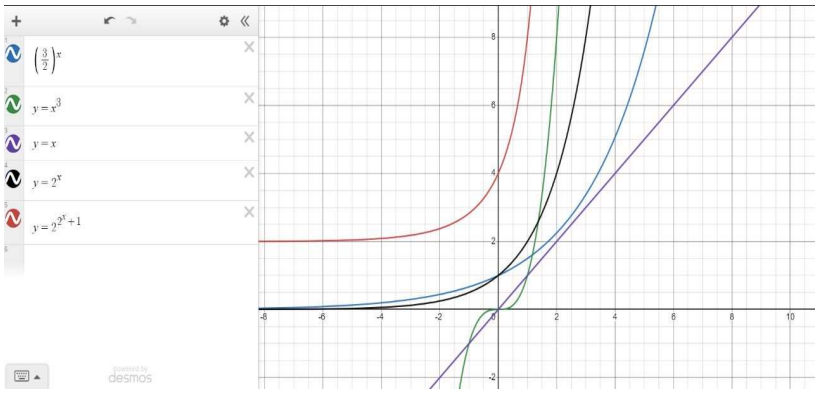
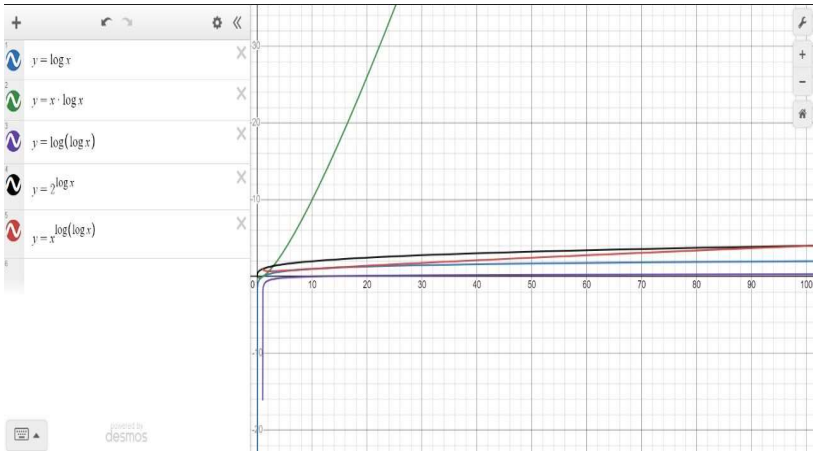
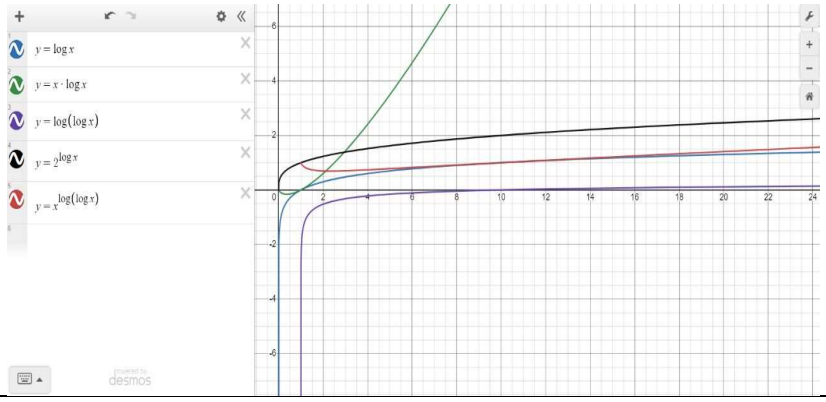
```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
7
Input   output
0       -1.#J
10      23.03
20      59.91
30      102.04
40      147.56
50      195.60
60      245.66
70      297.39
80      350.56
90      404.98
100     460.52
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
8
Input   output
0       -1.#J
10      0.83
20      1.10
30      1.22
40      1.31
50      1.36
60      1.41
70      1.45
80      1.48
90      1.50
100     1.53
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
9
Input   output
0       0.00
10      4.93
20      7.98
30      10.56
40      12.90
50      15.05
60      17.08
70      19.01
80      20.85
90      22.62
100     24.34
```

```
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
10
Input   output
0       -1.#J
10      6.82
20      26.76
30      64.30
40      123.37
50      207.72
60      320.99
70      466.72
80      648.39
90      869.46
100     1133.34
```

```
Enter 1 for (3/2)^n function
Enter 2 for n^3(cubic) function
Enter 3 for n(linear) function
Enter 4 for 2^n function
Enter 5 for lgn function
Enter 6 for 2^2^n+1 function
Enter 7 for nlogn function
Enter 8 for loglogn function
Enter 9 for 2^logn function
Enter 10 for n^loglogn function
Enter 11 for n! function
Enter 0 for exit
11
Input   output
0       1.00
2       2.00
4       24.00
6       720.00
8       40320.00
10      3628800.00
12      479001600.00
14      87178289152.00
16      20922790576128.00
18      6402373530419200.00
20      2432902023163674600.00
```

GRAPHS	   
CONCLUSION	1)Among all,2^2^x+1 increases to infinity the fastest.

	2)amongst all,log(log(x)) tends to zero as the number increases from 0-100 3)x.logx function does not saturate as the other log functions. 4)logx and x^log(logx) intersects 5)Other Increasing and decreasing nature of the graphs is observed.

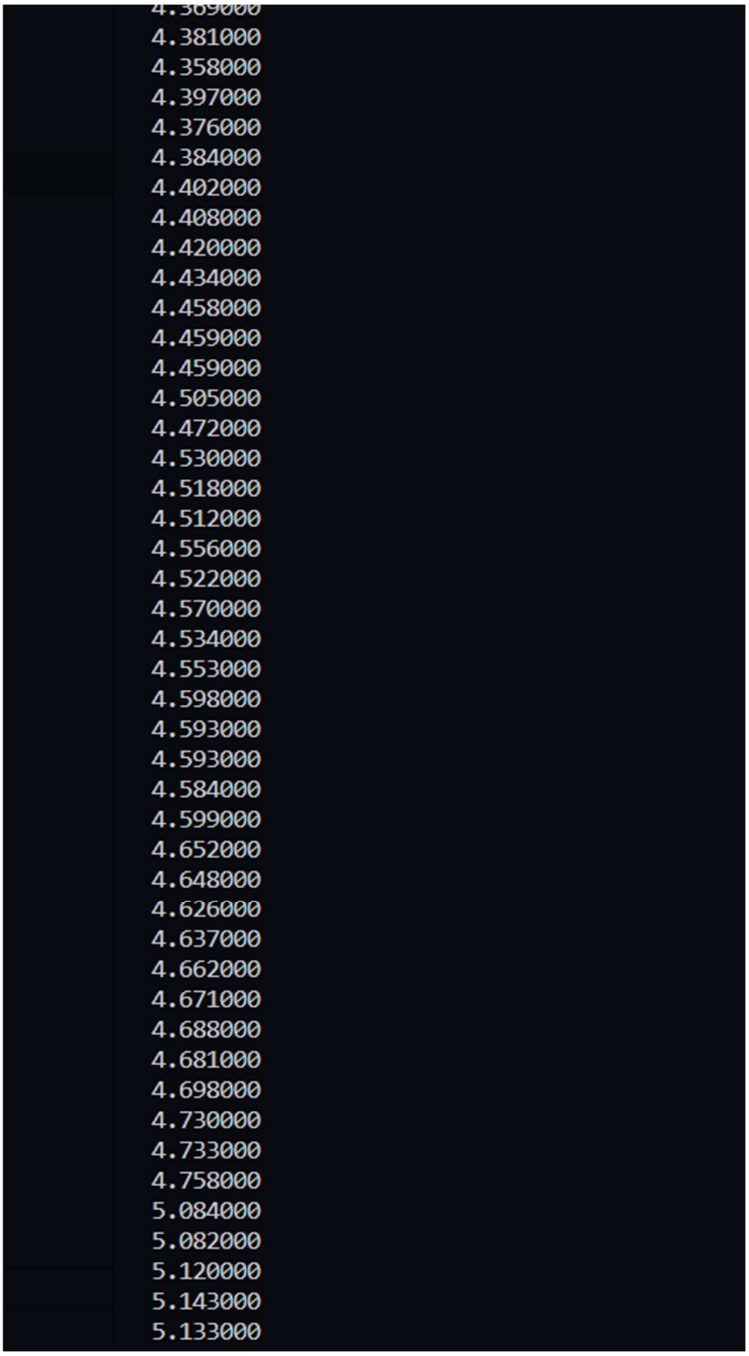
Exp1-B

AIM	Experiment on finding the running time of an algorithm.(Insertion and Selection sort)
ALGORITHM	**Insertion sort-** **Step 1 -** If the element is the first element, assume that it is already sorted. Return 1. **Step2 -** Pick the next element, and store it separately in a **key.** **Step3 -** Now, compare the **key** with all elements in the sorted array. **Step 4 -** If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right. **Step 5 -** Insert the value. **Step 6 -** Repeat until the array is sorted. **Selection sort-** **Step 1** − Set MIN to location 0 **Step 2** − Search the minimum element in the list **Step 3** − Swap with value at location MIN **Step 4** − Increment MIN to point to next element **Step 5** − Repeat until list is sorted

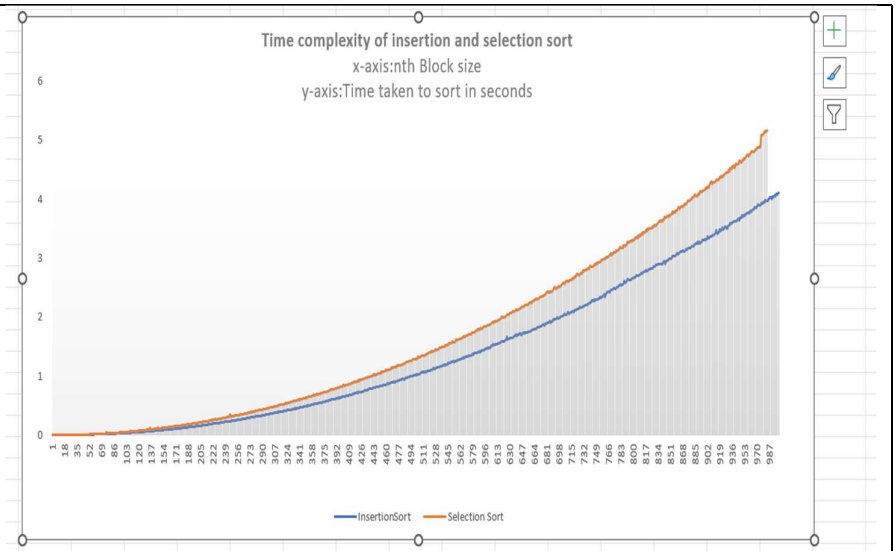
PROGRAM	`#include<stdio.h>`
	`#include<stdlib.h>`
	`#include<time.h>`
	
	`void swap(int*a , int*b){`
	`  int temp = *a;`
	`  *a = *b;`
	`  *b = temp;`
	`}`
	
	`void selSort(int* arr , int size){`
	`  for(int i=0;i<size-1;i++){`
	`  int minId = i;`
	`  for(int j=i+1;j<size;j++){`

```
            if(arr[j]<arr[minId]){
                minId = j;
            }
        }
        if(i!=minId){
            swap(&arr[i],&arr[minId]);
        }
    }
}

void insertSort(int *arr, int n){
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}



int main(){
    for(int i=1;i<=1000;i++){
        int j=0;
        int numberArray[100000];
        FILE *f;
        f = fopen("new.txt","r");
        for (j = 0; j < 100000; j++){
            fscanf(f, "%d,", &numberArray[j] );
            }
        fclose(f);
        clock_t t;
        t = clock();
        // insertSort(numberArray,i*100);
        selSort(numberArray,i*100);
        t = clock() - t;
        double time_taken = ((double)t)/CLOCKS_PER_SEC;
        printf("%f\n",time_taken);
    }
}
```

RESULT	Selection Sort-



	Insertion sort- 0.164000 0.164000 0.167000 0.169000 0.172000 0.174000 0.175000 0.174000 0.177000 0.180000 0.180000 0.182000 0.183000 0.186000 0.189000 0.190000 0.193000 0.195000 0.195000 0.196000 0.198000 0.199000 0.200000 0.204000 0.208000 0.207000 0.209000 0.213000 0.213000 0.216000 0.217000 0.220000 0.221000 0.226000 0.228000 0.229000 0.230000 0.238000 0.232000 0.236000 0.237000 0.241000

GRAPH	
CONCLUSION	For 6000-1000 inputs,time taken by selection sort and insertion sort are nearly same(still insertion sort takes less time) but as the number of input size increases,the time difference of sorting also increases.From the graph we can see for large number of inputs insertion sort takes less time to sort.Hence,Insertion sort is better than selection sort.