Name	Siddhesh Sonar
UID no.	2021700063
Experiment No.	1

AIM:	To implement the various functions e.g. linear, non-linear, quadratic, exponential etc.
Program 1A	
PROBLEM STATEMENT :	For this experiment, we have to implement at least 10 functions from the list. The input (i.e. n) to all the above functions varies from 0 to 100 with increment of 10. Then add the function n! in the list and execute the same for n from 0 to 20 with increment of 2.
ALGORITHM/ THEORY:	In this experiment part 1-A we must use 10 functions from the given list write a simple C code for it and then use the output from the code to draw a graph for the 10 functions + the factorial function and draw our conclusion regarding the functions.

PROGRAM:	`#include <stdio.h>`
	`#include <math.h>`
	
	`unsigned long long fact(int n)`
	`{`
	`if (n == 0)`
	`{`
	`return 1;`
	`}`
	`else`
	`{`
	`// for (int i = n - 1; i > 0; i--)`
	`// {`
	`// n = n * i;`
	`// }`

```
        return n * fact(n - 1);
        }
}

float func1(int n)
{
        //(3/2)^n
        return pow(1.5, n);
}

int func2(int n)
{
        // n^3
        return pow(n, 3);
}

float func3(int n)
{
        //(lg^2)*n
        return pow(log2(n), 2);
}

float func4(int n)
{
        // sqrt(log(n))
        return sqrt(log2(n));
}

float func5(int n)
{
        // n log n
        return n * log2(n);
}

float func6(int n)
{
        // ln ln n
        return log(log(n));
}

float func7(int n)
{
        // log n
        return log2(n);
}
```

```
float func8(int n)
{
	// 2^n
	return pow(2, n);
}

float func9(int n)
{
	// ln n
	return log(n);
}

int func10(int n)
{
	// n+5
	return n + 10;
}

void print1(int start, int end, int inc, int (*func)())
{
	for (int i = start; i <= end; i = i + inc)
	{
	printf("%d = %d\n", i, func(i));
	}
}
void print2(int start, int end, int inc, float (*func)())
{
	for (int i = start; i <= end; i = i + inc)
	{
	printf("%d = %f\n", i, func(i));
	}
}

void print3(int start, int end, int inc, unsigned long long (*func)())
{
	for (int i = start; i <= end; i = i + inc)
	{
	printf("%d = %lld\n", i, func(i));
	}
}

int (*intf)(int);
float (*floatf)(int);
unsigned long long (*longf)(int);
int main()
{
```

```
        printf("\n(3/2)^n\n\n");
        floatf = func1;
        print2(0, 100, 10, floatf);
        printf("\nn^3\n\n");
        intf = func2;
        print1(0, 100, 10, intf);
        printf("\n(lg^2)*n\n\n");
        floatf = func3;
        print2(0, 100, 10, floatf);
        printf("\nsqrt(log(n))\n\n");
        floatf = func4;
        print2(0, 100, 10, floatf);
        printf("\nn log n\n\n");
        floatf = func5;
        print2(0, 100, 10, floatf);
        printf("\nln ln n\n\n");
        floatf = func6;
        print2(0, 100, 10, floatf);
        printf("\nlog n\n\n");
        floatf = func7;
        print2(0, 100, 10, floatf);
        printf("\n2^n\n\n");
        floatf = func8;
        print2(0, 100, 10, floatf);
        printf("\nln n\n\n");
        floatf = func9;
        print2(0, 100, 10, floatf);
        printf("\nn+5\n\n");
        intf = func10;
        print1(0, 100, 10, intf);
        printf("\nn!\n\n");
        longf = fact;
        print3(0, 20, 2, longf);
        return 0;
}
```

RESULT:	3/2)^n 0 = 1.000000 10 = 57.665039 20 = 3325.256836 30 = 191751.062500 40 = 11057332.000000 50 = 637621504.000000 60 = 36768468992.000000 70 = 2120255143936.000000 80 = 122264599134208.000000 90 = 7050392827330560.000000 100 = 406561191922499584.000000 n^3 0 = 0 10 = 1000 20 = 8000 30 = 27000 40 = 64000 50 = 125000 60 = 216000 70 = 343000 80 = 512000 90 = 729000 100 = 1000000 (lg^2)*n 0 = inf 10 = 11.035206 20 = 18.679062 30 = 24.077576 40 = 28.322918 50 = 31.853113 60 = 34.891357 70 = 37.568111 80 = 39.966774 90 = 42.144157 100 = 44.140823 sqrt(log(n)) 0 = -nan 10 = 1.822616

20 = 2.078925
30 = 2.215150
40 = 2.306931
50 = 2.375680
60 = 2.430410
70 = 2.475739
80 = 2.514344
90 = 2.547911
100 = 2.577568

n log n

0 = -nan
10 = 33.219280
20 = 86.438560
30 = 147.206711
40 = 212.877121
50 = 282.192810
60 = 354.413422
70 = 429.049805
80 = 505.754242
90 = 584.266785
100 = 664.385620

ln ln n

0 = -nan
10 = 0.834032
20 = 1.097189
30 = 1.224128
40 = 1.305323
50 = 1.364055
60 = 1.409607
70 = 1.446565
80 = 1.477511
90 = 1.504035
100 = 1.527180

log n

0 = -inf
10 = 3.321928
20 = 4.321928
30 = 4.906890
40 = 5.321928
50 = 5.643856

60 = 5.906890
70 = 6.129283
80 = 6.321928
90 = 6.491853
100 = 6.643856

2^n

0 = 1.000000
10 = 1024.000000
20 = 1048576.000000
30 = 1073741824.000000
40 = 1099511627776.000000
50 = 1125899906842624.000000
60 = 1152921504606846976.000000
70 = 1180591620717411303424.000000
80 = 1208925819614629174706176.000000
90 = 1237940039285380274899124224.000000
100 = 1267650600228229401496703205376.000000

ln n

0 = -inf
10 = 2.302585
20 = 2.995732
30 = 3.401197
40 = 3.688879
50 = 3.912023
60 = 4.094345
70 = 4.248495
80 = 4.382027
90 = 4.499810
100 = 4.605170

n+5

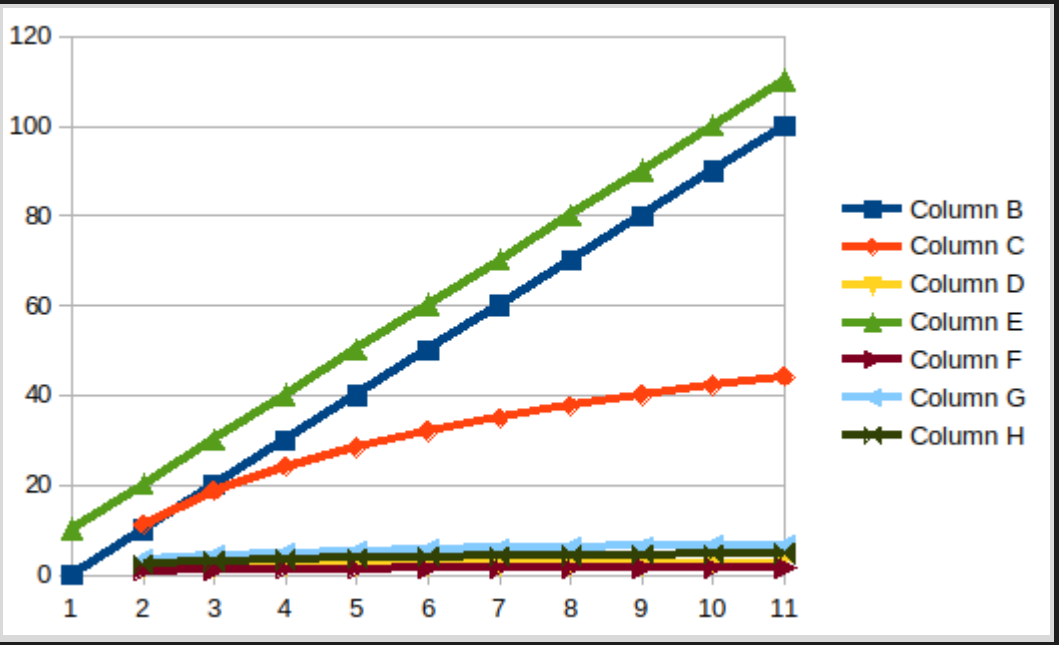
0 = 10
10 = 20
20 = 30
30 = 40
40 = 50
50 = 60
60 = 70
70 = 80
80 = 90
90 = 100

	100 = 110 n! 0 = 1 2 = 2 4 = 24 6 = 720 8 = 40320 10 = 3628800 12 = 479001600 14 = 87178291200 16 = 20922789888000 18 = 6402373705728000 20 = 2432902008176640000

GRAPH:

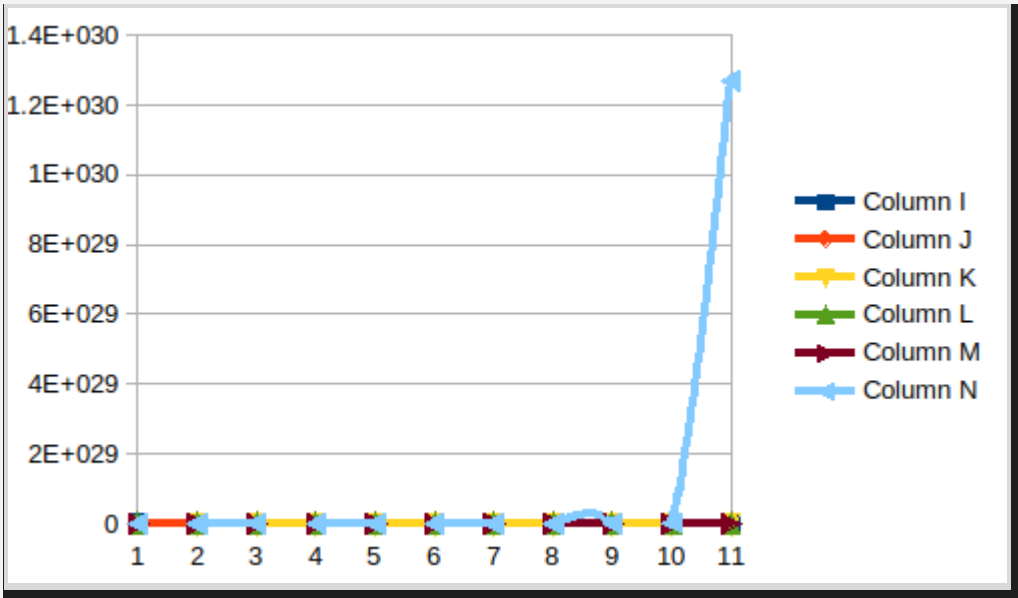
	(lg^2)*n	sqrt(log(n))	n+5	ln ln n	log n	ln n
0	inf	-nan	10	-nan	-inf	-inf
10	11.035206	1.822616	20	0.834032	3.321928	2.302585
20	18.679062	2.078925	30	1.097189	4.321928	2.995732
30	24.077576	2.21515	40	1.224128	4.90689	3.401197
40	28.322918	2.306931	50	1.305323	5.321928	3.688879
50	31.853113	2.37568	60	1.364055	5.643856	3.912023
60	34.891357	2.43041	70	1.409607	5.90689	4.094345
70	37.568111	2.475739	80	1.446565	6.129283	4.248495
80	39.966774	2.514344	90	1.477511	6.321928	4.382027
90	42.144157	2.547911	100	1.504035	6.491853	4.49981
100	44.140823	2.577568	110	1.52718	6.643856	4.60517

The above are columns B, C, D, E, F, G and H respectively

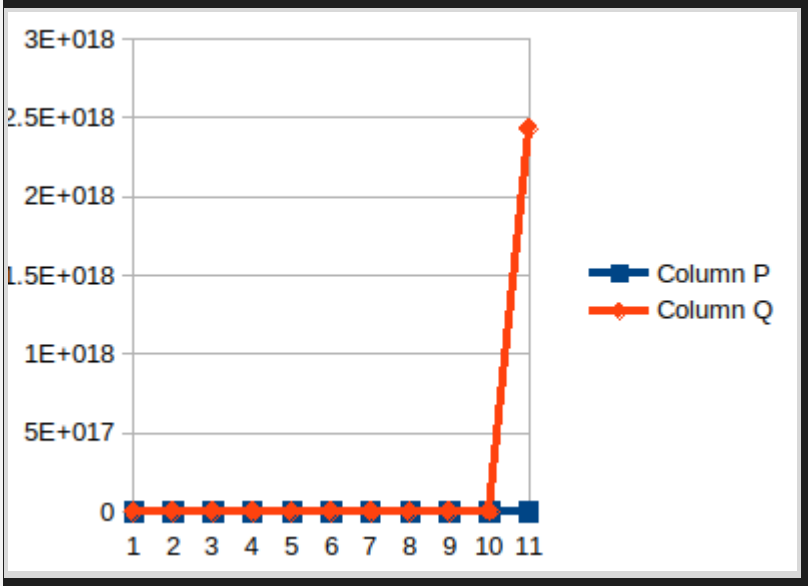


	n^3	n log n	n!	(3/2)^n	2^n
0	0	-nan	1	1	1
10	1000	33.21928	2	57.665039	1024
20	8000	86.43856	24	3325.256836	1048576
30	27000	147.206711	720	191751.0625	1073741824
40	64000	212.877121	40320	11057332	1.09951E+12
50	125000	282.19281	3628800	637621504	1.1259E+15
60	216000	354.413422	479001600	36768468992	1.15292E+18
70	343000	429.049805	87178291200	2.12026E+12	1.18059E+21
80	512000	505.754242	2.09228E+13	1.22265E+14	1.20893E+24
90	729000	584.266785	6.40237E+15	7.05039E+15	1.23794E+27
100	1000000	664.38562	2.4329E+18	4.06561E+17	1.26765E+30

The above are columns I, J, K, L, M and N respectively



The graph below is the graph of factorial function



CONCLUSION:	We observer that due to the large exponential values in some of the functions some of the graph become very huge and we also learned how to create graphs using excel

AIM:	Experiment on finding the running time of an algorithm.
Program 1B	
PROBLEM STATEMENT :	For this experiment, you need to implement two sorting algorithms namely Insertion and Selection sort methods. Compare these algorithms based on time and space complexity. Time required to sorting algorithms can be performed using high_resolution_clock::now() under namespace std::chrono. You have togenerate1,00,000 integer numbers using C/C++ Rand function and save them in a text file. Both the sorting algorithms uses these 1,00,000 integer numbers as input as follows. Each sorting algorithm sorts a block of 100 integers numbers with array indexes numbers A[0..99], A[0..199], A[0..299],..., A[0..99999]. You need to use high_resolution_clock::now() function to find the time required for 100, 200, 300.... 100000 integer numbers. Finally, compare two algorithms namely Insertion and Selection by plotting the time required to sort 100000 integers using LibreOffice Calc/MS Excel. The x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot representsthe tunning time to sort 1000 blocks of 100,200,300,...,100000 integer numbers. Note – You have to use C/C++ file processing functions for reading and writing randomly generated 100000 integer numbers.

ALGORITHM/ THEORY:	Algorithm for Selection sort:
	for(i=0;i<n;i++)
	{
	scanf("%d",&arr[i]);
	}
	for(i=0;i<n;i++)
	{
	min=i;
	for(j=i+1;j<n;j++)
	{
	if(arr[j]<arr[min])
	{
	min=j;
	}
	}
	if(min!=i)
	{
	temp=arr[min];
	arr[min]=arr[i];

arr[i]=temp;
}
}

Algorithm for Insertion sort:
for(i=0;i=0 && arr[j]>temp)
{
scanf("%d",&arr[i]);
}
for(i=1;i=0 && arr[j]>temp)
{
temp=arr[i];
j=i-1;
while(j>=0 && arr[j]>temp)
{
arr[j+1]=arr[j];
j--;
}
arr[j+1]=temp;
}

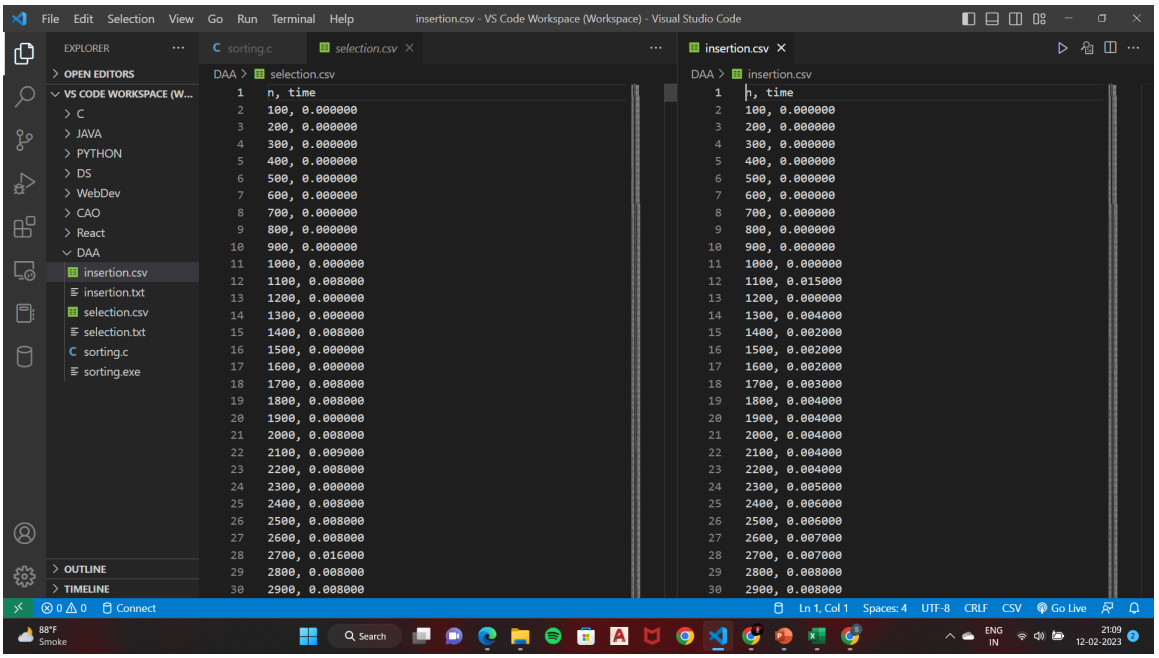
PROGRAM:

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

void filling(int a1[], int a2[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int rnum = rand();
        a1[i] = a2[i] = rnum;
    }
}

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

void selection(int arr[], int n)
{
    int min;
```

```
    FILE *fp = fopen("./selection.csv", "w");
    fprintf(fp, "n, time\n");
    for (int k = 100; k < n; k += 100)
    {
        clock_t start, end;
        double time_taken;
        start = clock();
        for (int i = 0; i < k; i++)
        {
            min = i;
            for (int j = i + 1; j < k; j++)
            {
                if (arr[j] < arr[min])
                {
                    min = j;
                }
            }
            swap(&arr[min], &arr[i]);
        }
        end = clock();
        time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
        fprintf(fp, "%d, %f\n", k, time_taken);
        printf("\nSorted 0 to %d in %.2f seconds\n", k, time_taken);
    }
    fclose(fp);
    fp = fopen("./selection.txt", "w");
    for (int i = 0; i < n; i++)
    {
        fprintf(fp, "%d\n", arr[i]);
    }
    fclose(fp);
}

void insertion(int arr[], int n)
{
    FILE *fp = fopen("./insertion.csv", "w");
    fprintf(fp, "n, time\n");
    for (int k = 100; k < n; k += 100)
    {
        clock_t start, end;
        double time_taken;
        start = clock();
        for (int i = 0; i < k - 1; i++)
        {
            if (arr[i + 1] < arr[i])
            {
```

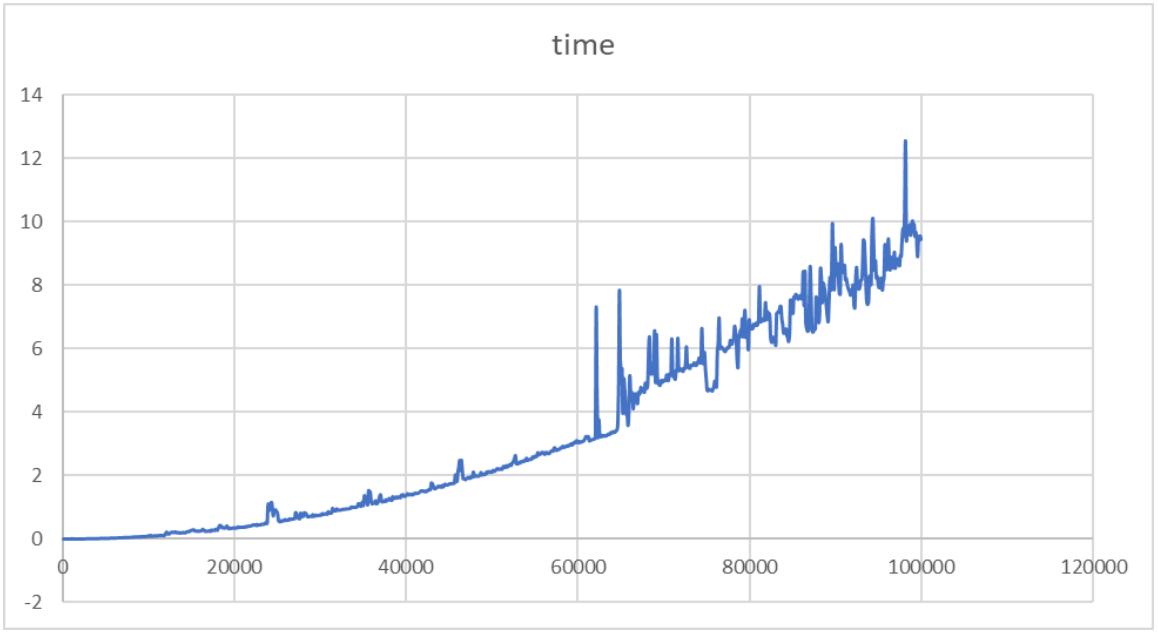
```
                swap(&arr[i], &arr[i + 1]);
            }
            for (int j = i; j > 0; j--)
            {
                if (arr[j] < arr[j - 1])
                {
                    swap(&arr[j], &arr[j - 1]);
                }
            }
        }
        end = clock();
        time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
        fprintf(fp, "%d, %f\n", k, time_taken);
        printf("\nSorted 0 to %d in %.2f seconds\n", k, time_taken);
    }
    fclose(fp);
    fp = fopen("./insertion.txt", "w");
    for (int i = 0; i < n; i++)
    {
        fprintf(fp, "%d\n", arr[i]);
    }
    fclose(fp);
}

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

int main()
{
    int n = 100000;
    int a1[n], a2[n];   // will be same
    filling(a1, a2, n); // a1 and a2 filling with 1lakh random
numbers
    insertion(a1,n);
    selection(a2,n);
    return 0;
}
```

RESULT:

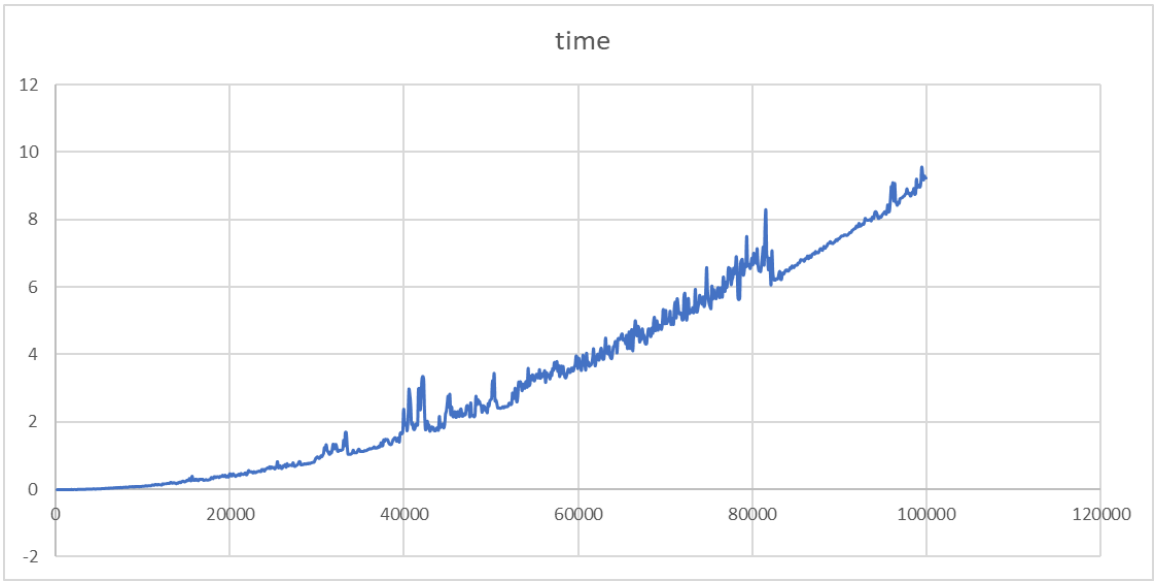

File Edit Selection View Go Run Terminal Help
insertion.csv - VS Code Workspace (Workspace) - Visual Studio Code
EXPLORER
OPEN EDITORS
VS CODE WORKSPACE (W...
C
JAVA
PYTHON
DS
WebDev
CAO
React
DAA
insertion.csv
insertion.txt
selection.csv
selection.txt
sorting.c
sorting.exe
OUTLINE
TIMELINE
sorting.c
selection.csv
DAA > selection.csv
n, time
100, 0.000000
200, 0.000000
300, 0.000000
400, 0.000000
500, 0.000000
600, 0.000000
700, 0.000000
800, 0.000000
900, 0.000000
1000, 0.000000
1100, 0.008000
1200, 0.000000
1300, 0.000000
1400, 0.008000
1500, 0.000000
1600, 0.000000
1700, 0.008000
1800, 0.008000
1900, 0.000000
2000, 0.008000
2100, 0.009000
2200, 0.008000
2300, 0.000000
2400, 0.008000
2500, 0.008000
2600, 0.008000
2700, 0.016000
2800, 0.008000
2900, 0.008000
insertion.csv
DAA > insertion.csv
h, time
100, 0.000000
200, 0.000000
300, 0.000000
400, 0.000000
500, 0.000000
600, 0.000000
700, 0.000000
800, 0.000000
900, 0.000000
1000, 0.000000
1100, 0.015000
1200, 0.000000
1300, 0.004000
1400, 0.002000
1500, 0.002000
1600, 0.002000
1700, 0.003000
1800, 0.004000
1900, 0.004000
2000, 0.004000
2100, 0.004000
2200, 0.004000
2300, 0.005000
2400, 0.006000
2500, 0.006000
2600, 0.007000
2700, 0.007000
2800, 0.008000
2900, 0.008000
Connect
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
CSV
Go Live
88°F
Smoke
Search
ENG
IN
21:09
12-02-2023

GRAPH:

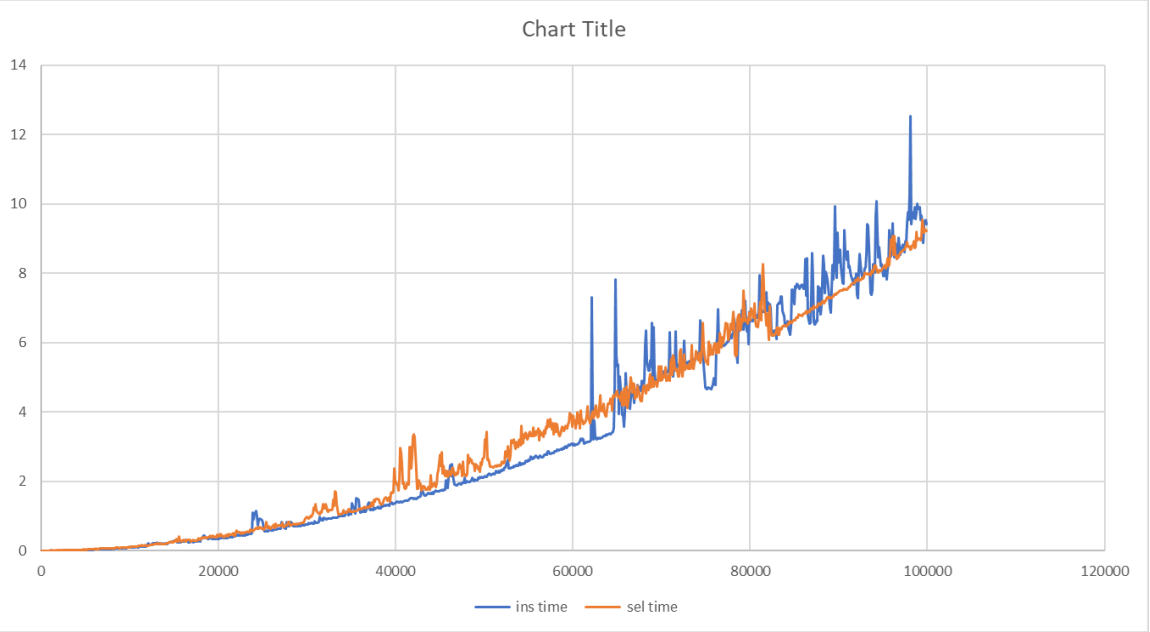
Graph for insertion sort:



Graph for selection sort:



Graph of both the sorting methods:



CONCLUSION:	Successfully performed insertion sort and selection sort on 100000 random numbers, calculated their time complexity, plotted their graphs and observed that insertion sort is more efficient and faster than selection sort.