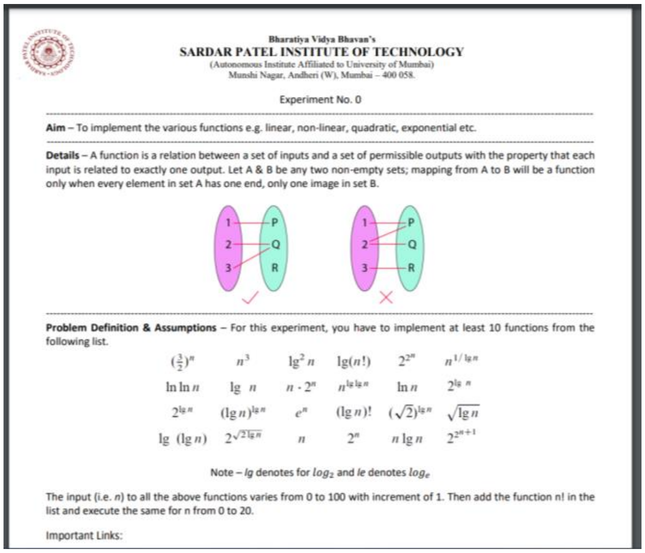
Name	Kinnari Shah
UID no.	2021700058
Experiment No.	1
Subject	DAA

AIM of 1A:	To implement the various functions e.g. linear, non-linear, quadratic, exponential etc

Program 1

PROBLEM STATEMENT :



Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.

Experiment No. 0

Aim – To implement the various functions e.g. linear, non-linear, quadratic, exponential etc.

Details – A function is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output. Let A & B be any two non-empty sets; mapping from A to B will be a function only when every element in set A has one end, only one image in set B.

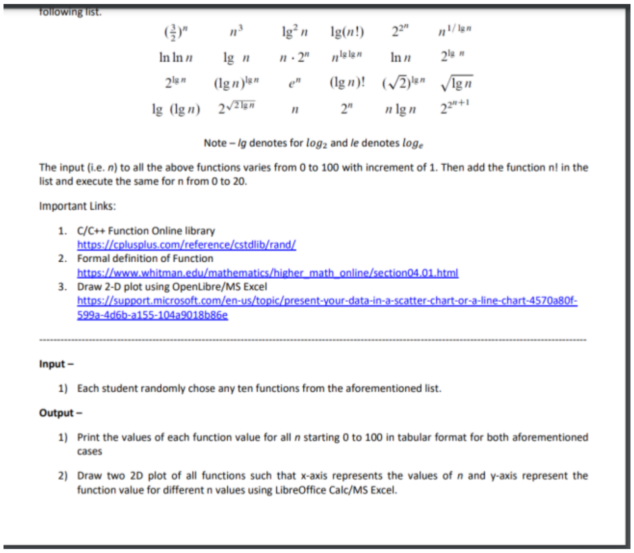
Problem Definition & Assumptions – For this experiment, you have to implement at least 10 functions from the following list.

$(\frac{3}{2})^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg\ n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	$2^{\lg\ n}$
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$(\lg n)!$	$(\sqrt{2})^{\lg n}$	$\sqrt{\lg n}$
$\lg\ (\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

Note – *lg* denotes for log_2 and *le* denotes log_e

The input (i.e. *n*) to all the above functions varies from 0 to 100 with increment of 1. Then add the function n! in the list and execute the same for n from 0 to 20.

Important Links:



following list.

$(\frac{3}{2})^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg\ n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	$2^{\lg\ n}$
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$(\lg n)!$	$(\sqrt{2})^{\lg n}$	$\sqrt{\lg n}$
$\lg\ (\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

Note – *lg* denotes for log_2 and *le* denotes log_e

The input (i.e. *n*) to all the above functions varies from 0 to 100 with increment of 1. Then add the function n! in the list and execute the same for n from 0 to 20.

Important Links:

1. C/C++ Function Online library
   https://cplusplus.com/reference/cstdlib/rand/
2. Formal definition of Function
   https://www.whitman.edu/mathematics/higher_math_online/section04.01.html
3. Draw 2-D plot using OpenLibre/MS Excel
   https://support.microsoft.com/en-us/topic/present-your-data-in-a-scatter-chart-or-a-line-chart-4570a80f-599a-4d6b-a155-104a9018b86e

Input –

1) Each student randomly chose any ten functions from the aforementioned list.

Output –

1) Print the values of each function value for all *n* starting 0 to 100 in tabular format for both aforementioned cases
2) Draw two 2D plot of all functions such that x-axis represents the values of *n* and y-axis represent the function value for different n values using LibreOffice Calc/MS Excel.

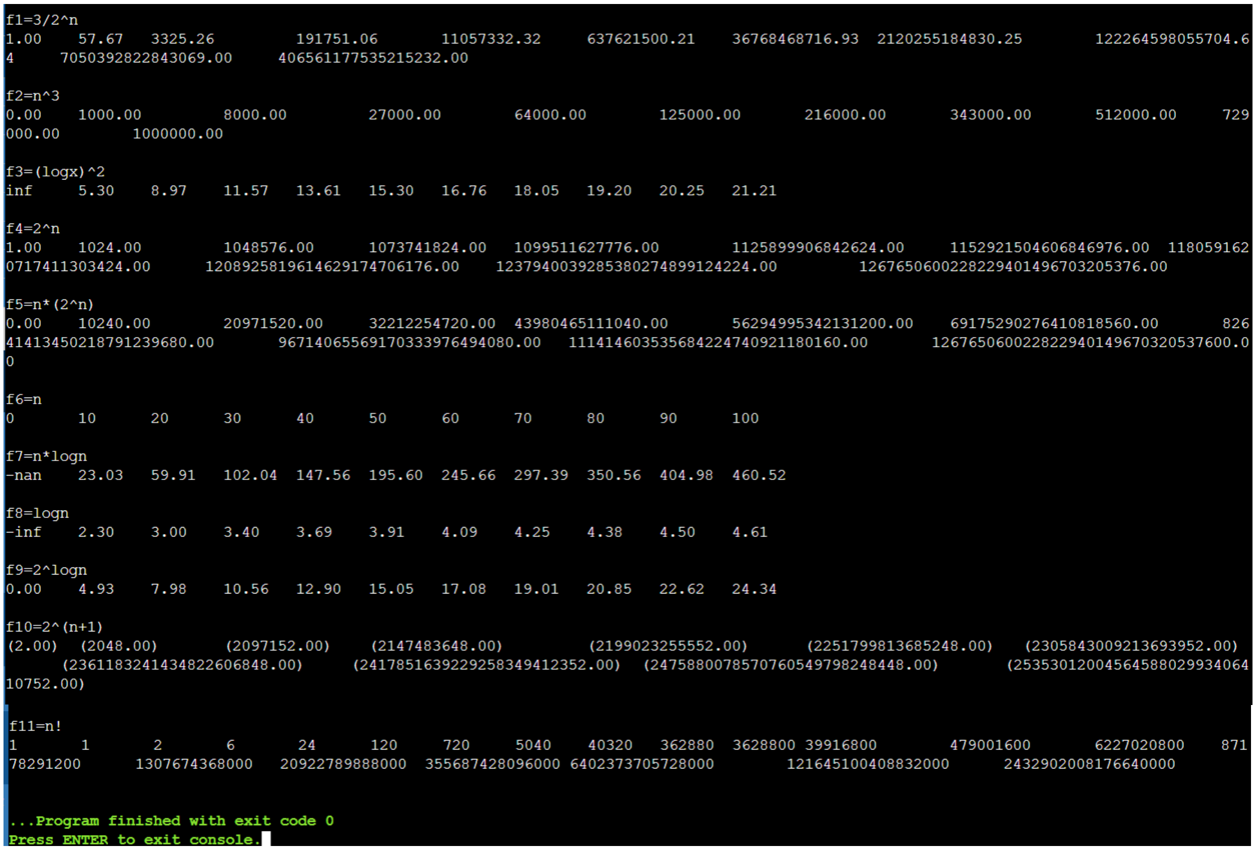
PROGRAM:

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

double f1(int arr[])
{
    printf("\nf1=3/2^n\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(3.0/2,arr[i]);
      printf("%.2lf \t",r);
    }
    printf("\n");
}

double f2(int arr[])
{
    printf("\nf2=n^3\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(arr[i],3);
      printf("%.2lf \t",r);
    }
    printf("\n");
}

double f3(int arr[])
{
    printf("\nf3=(logx)^2\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(log(arr[i]),2);
      printf("%.2lf \t",r);
    }
    printf("\n");
}
```

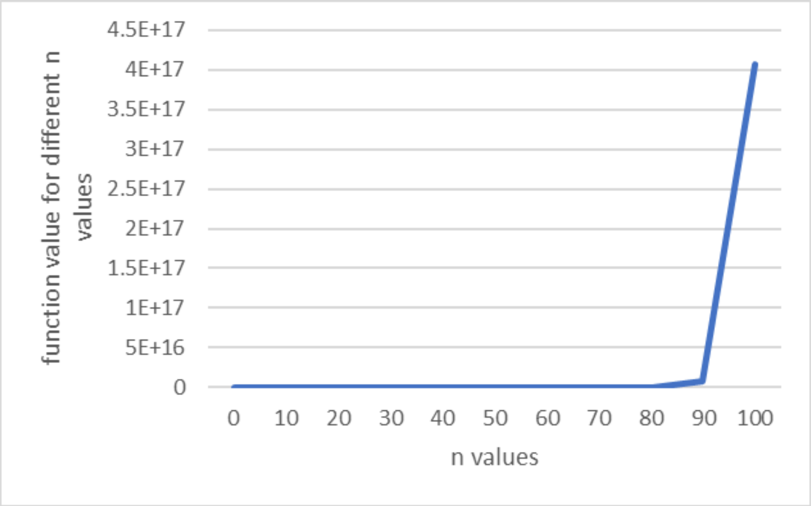
```
double f4(int arr[])
{
    printf("\nf4=2^n\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(2,arr[i]);
      printf("%.2lf \t",r);
    }
    printf("\n");
}

double f5(int arr[])
{
    printf("\nf5=n*(2^n)\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(2,arr[i]);
      printf("%.2lf \t",arr[i]*r);
    }
    printf("\n");
}

double f6(int arr[])
{
    printf("\nf6=n\n");
    for(int i=0;i<=10;i++)
    {
      printf("%d \t",arr[i]);
    }
    printf("\n");
}

double f7(int arr[])
{
    printf("\nf7=n*logn\n");
    for(int i=0;i<=10;i++)
    {
      double r=arr[i]*log(arr[i]);
      printf("%.2lf \t",r);
    }
    printf("\n");
}

double f8(int arr[])
{
    printf("\nf8=logn\n");
    for(int i=0;i<=10;i++)
    {
      double r=log(arr[i]);
      printf("%.2lf \t",r);
    }
    printf("\n");
}

double f9(int arr[])
{
    printf("\nf9=2^logn\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(2,log(arr[i]));
      printf("%.2lf \t",r);
    }
    printf("\n");
}

double f10(int arr[])
{
    printf("\nf10=2^(n+1)\n");
    for(int i=0;i<=10;i++)
    {
      double r=pow(2,(arr[i]+1));
      printf("(%.2lf) \t",r);
    }
    printf("\n");
}

long fact(int n)
{
        if(n==0)
        return 1;

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

int main()
{
    int arr[11]={0,10,20,30,40,50,60,70,80,90,100};
    f1(arr);
    f2(arr);
    f3(arr);
    f4(arr);
    f5(arr);
    f6(arr);
    f7(arr);
    f8(arr);
    f9(arr);
    f10(arr);

    printf("\nf11=n!\n");
    for(int i=0;i<=20;i++)
    {
        fact(i);
        printf("%ld\t",fact(i));
    }
    printf("\n");

    return 0;
```

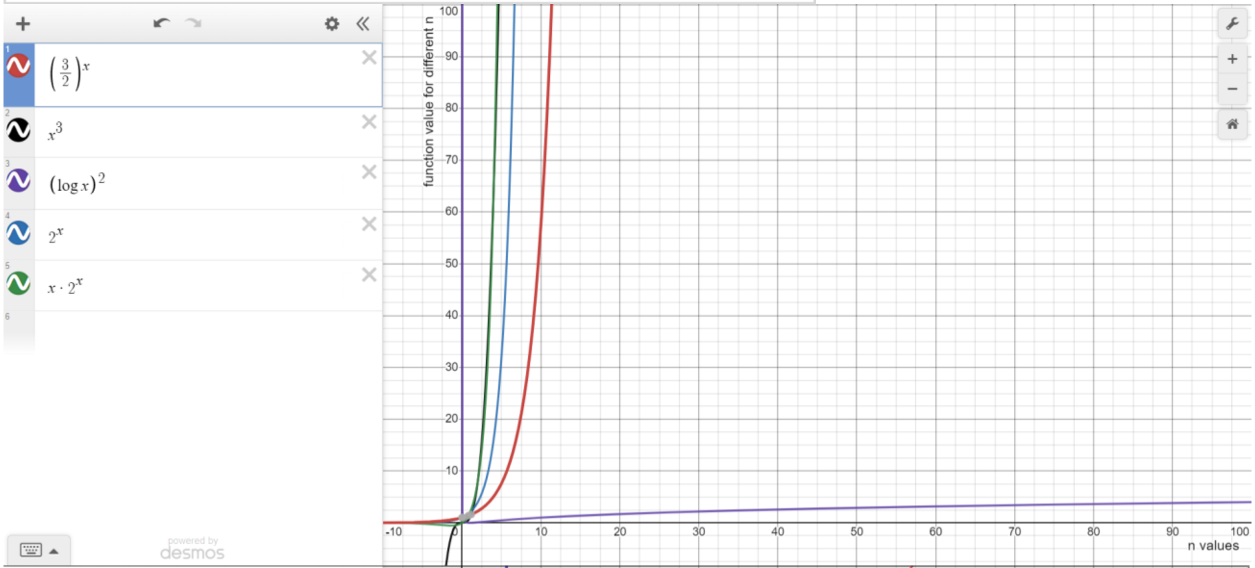
	}

RESULT:



```
f1=3/2^n
1.00     57.67   3325.26         191751.06       11057332.32      637621500.21    36768468716.93  2120255184830.25        122264598055704.6
4     7050392822843069.00     406561177535215232.00

f2=n^3
0.00     1000.00          8000.00          27000.00         64000.00         125000.00        216000.00        343000.00        512000.00      729
000.00        1000000.00

f3=(logx)^2
inf      5.30     8.97     11.57    13.61    15.30    16.76    18.05    19.20    20.25    21.21

f4=2^n
1.00     1024.00          1048576.00       1073741824.00    1099511627776.00        1125899906842624.00      1152921504606846976.00  118059162
0717411303424.00      1208925819614629174706176.00    1237940039285380274899124224.00         1267650600228229401496703205376.00

f5=n*(2^n)
0.00     10240.00         20971520.00      32212254720.00  43980465111040.00       56294995342131200.00    69175290276410818560.00        826
41413450218791239680.00       96714065569170333976494080.00   111414603535684224740921180160.00       126765060022822940149670320537600.0
0

f6=n
0        10       20       30       40       50       60       70       80       90       100

f7=n*logn
-nan     23.03    59.91    102.04   147.56   195.60   245.66   297.39   350.56   404.98   460.52

f8=logn
-inf     2.30     3.00     3.40     3.69     3.91     4.09     4.25     4.38     4.50     4.61

f9=2^logn
0.00     4.93     7.98     10.56    12.90    15.05    17.08    19.01    20.85    22.62    24.34

f10=2^(n+1)
(2.00)  (2048.00)        (2097152.00)     (2147483648.00)          (2199023255552.00)       (2251799813685248.00)    (2305843009213693952.00)
       (2361183241434822606848.00)      (2417851639229258349412352.00)  (2475880078570760549798248448.00)        (25353012004564588029934064
10752.00)

f11=n!
1        1        2        6        24       120      720      5040     40320    362880   3628800 39916800         479001600        6227020800     871
78291200      1307674368000    20922789888000   355687428096000 6402373705728000         121645100408832000       2432902008176640000


...Program finished with exit code 0
Press ENTER to exit console.
```
]





1 $\left(\frac{3}{2}\right)^x$

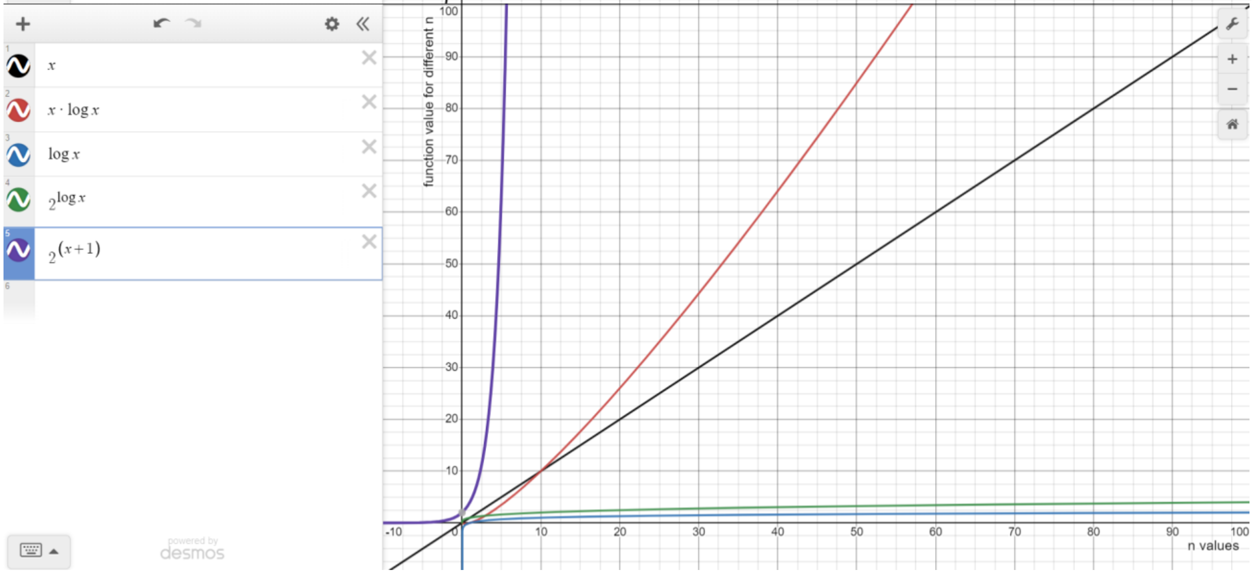
2 x^3

3 $(\log x)^2$

4 2^x

5 $x \cdot 2^x$

6



1 x

2 $x \cdot \log x$

3 $\log x$

4 $2^{\log x}$

5 $2^{(x+1)}$

6

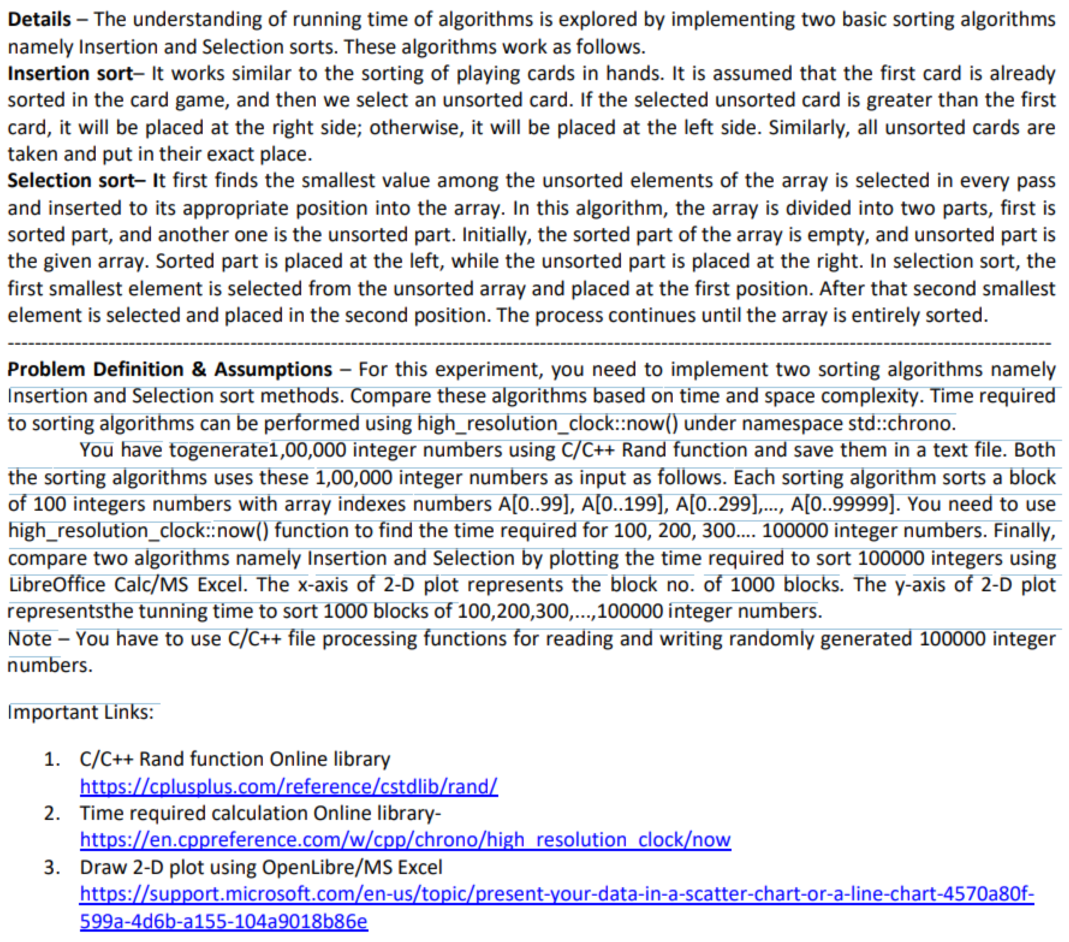
OBSERVATIONS:

- From these graphs we observe that some functions are increasing faster than others. For eg 2^(x+1) is increasing faster and is more steeper than x*logx. Also 2^(x+1) is steeper than 2^logx and logx.
- Also x*2^x is steeper than x, 2^x, x^3 and (logx)^2.
- All the graphs are increasing in nature from values 0 to 100.

Program 2

AIM of 1B: Experiment on finding the running time of an algorithm.

PROBLEM STATEMENT :



Details – The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Insertion and Selection sorts. These algorithms work as follows.

Insertion sort– It works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

Selection sort– It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right. In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

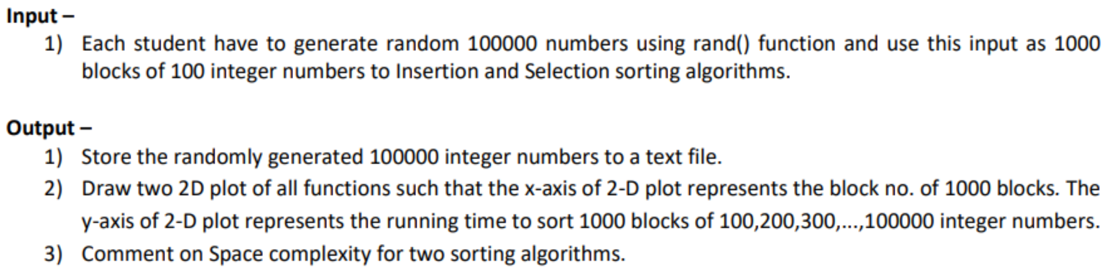
Problem Definition & Assumptions – For this experiment, you need to implement two sorting algorithms namely Insertion and Selection sort methods. Compare these algorithms based on time and space complexity. Time required to sorting algorithms can be performed using high_resolution_clock::now() under 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.

Important Links:

1. C/C++ Rand function Online library
   https://cplusplus.com/reference/cstdlib/rand/
2. Time required calculation Online library-
   https://en.cppreference.com/w/cpp/chrono/high_resolution_clock/now
3. Draw 2-D plot using OpenLibre/MS Excel
   https://support.microsoft.com/en-us/topic/present-your-data-in-a-scatter-chart-or-a-line-chart-4570a80f-599a-4d6b-a155-104a9018b86e

	 **Input –** 1) Each student have to generate random 100000 numbers using rand() function and use this input as 1000 blocks of 100 integer numbers to Insertion and Selection sorting algorithms. **Output –** 1) Store the randomly generated 100000 integer numbers to a text file. 2) Draw two 2D plot of all functions such that the x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot represents the running time to sort 1000 blocks of 100,200,300,...,100000 integer numbers. 3) Comment on Space complexity for two sorting algorithms.
ALGORITHM:	To create a text file containing 100000 random numbers Step 1: Start. Include stdlib.h library to use the rand function. Step 2:Initialise an array of size 100000 number[100000] Step 3:Run a for loop from i=0 to 100000 Step 4: Call the rand() function and store the value in number[i] array . Step 5:Create a input.txt file and save the 100000 randomly generated values which is in the range from 0 to 100000 .Stop. Selection Sort: Step 1: Start. Write a function to swap two numbers using a temporary variable. Step 2: Declare a function selectionSort(int arr[],n) which takes the array and given size of the array as argument. Step 3:Run for loop which runs from i=0 to n-1 and take the minimun element at first position of array by initialising minIndex=i Step 4:Run another for loop from j=i+1 to n which will keep checking the next two consecutive numbers in array. Step 5: Check if arr[j]<arr[minIndex] and find the smallest element Step 6: If smallest element is found update minIndex=j and call the swap function by swapping it with first element of array. Step 7:Now in main ,acess the text file input.txt where the random numbers are stored using fopen Step 8:Initialise block =1 and size of data array as 100 i.e data[size] Start 9:Run a while loop while(block<=1000) to access 1000 blocks , in this loop 1. Run a for loop from i=0 to i=size and start reading the random numbers in data[i] array. 2. Declare clock_t t to get starting time of program 3. Call selectionsort(data,size) to get the sorted array of first 100 numbers 4. Get the time elapsed by t=clock()-t 5. Calculate the time take for block by t/clocks_per_sec and print the runtime for first

block
6. Update size=size+100 and block++
7. Update the pointer by fseek to move the cursor to the start of the file.
Step 10: Stop

Insertion Sort :

Step 1: Start. Declare insertionSort(int arr[],int n) function
Step 2: Iterate from arr[1] to arr[n] over the array.
Step 3:Compare the current element (key) to its predecessor. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.
Step 4: Repeat the steps 7,8,9 as above in main function and call insertionSort(data,size) in main
Step 5:Stop

PROGRAM:

Selection Sort

```
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
void swap(int *a, int *b)
{
int temp = *a;
*a = *b;
*b = temp;
}
void selectionSort(int arr[],int n)
{
for(int i=0;i<n-1;i++)
{
int minIndex=i;
for(int j=i+1;j<n;j++)
{
if(arr[j]<arr[minIndex])
{
minIndex=j;
}
}
```

```
swap(&arr[minIndex],&arr[i]);
}

}
int main()
{
//to generate 1000000 random numbers
/*int numbers[100000];
for(int i=0;i<100000;i++)
{
numbers[i]=rand()%100000;
printf("%d \n",numbers[i]);

}*/
FILE* ptr;
// file in reading mode
ptr = fopen("input.txt", "r");
if (NULL == ptr)
{
printf("file can't be opened \n");
}
int block=1;
int size=100;
while(block<=1000)
{
int data[size];
for(int i=0;i<size;i++)
{
fscanf(ptr,"%d ",&data[i]);
//printf("%d ",data[i]);
}
clock_t t;
t = clock();
selectionSort(data,size);
if(block<3) //this prints sorted first 2 blocks only
{
printf("\n\nafter sorting block %d:\n",block);
for(int i=0;i<size;i++)
{
printf("%d \n",data[i]);
}
}

t = clock() - t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
printf("\nRuntime for block number %d : %f\n",block,time_taken);
size=size+100;
block++;
fseek(ptr,0,SEEK_SET); //moving cursor again to start pointer of txt file
}
fclose(ptr);

}
```

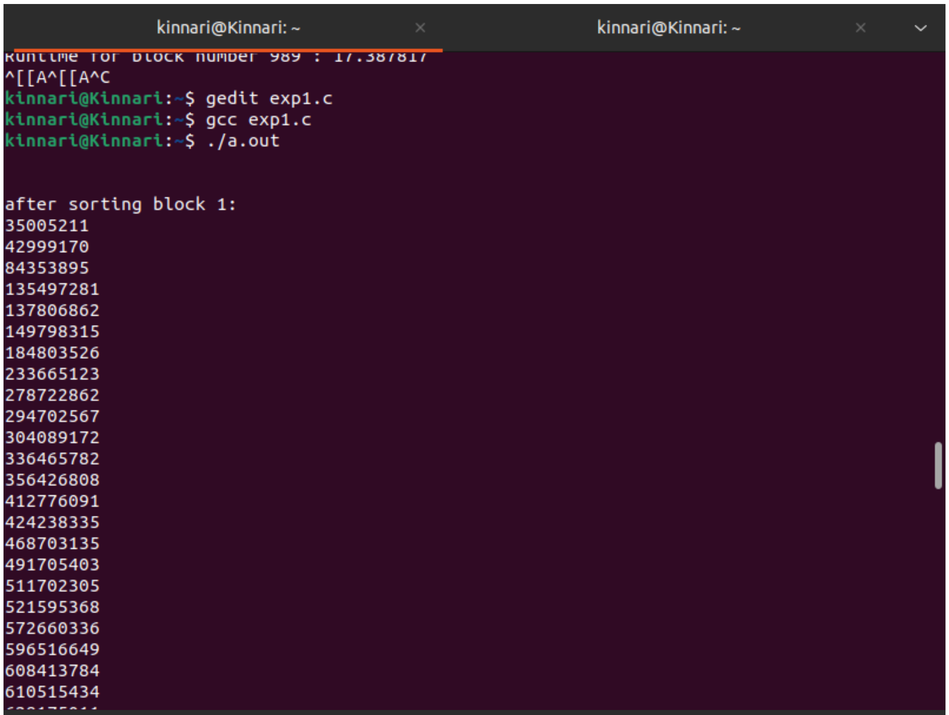
Insertion Sort

```
#include <math.h>
#include <stdio.h>
#include<time.h>
void insertionSort(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()
```

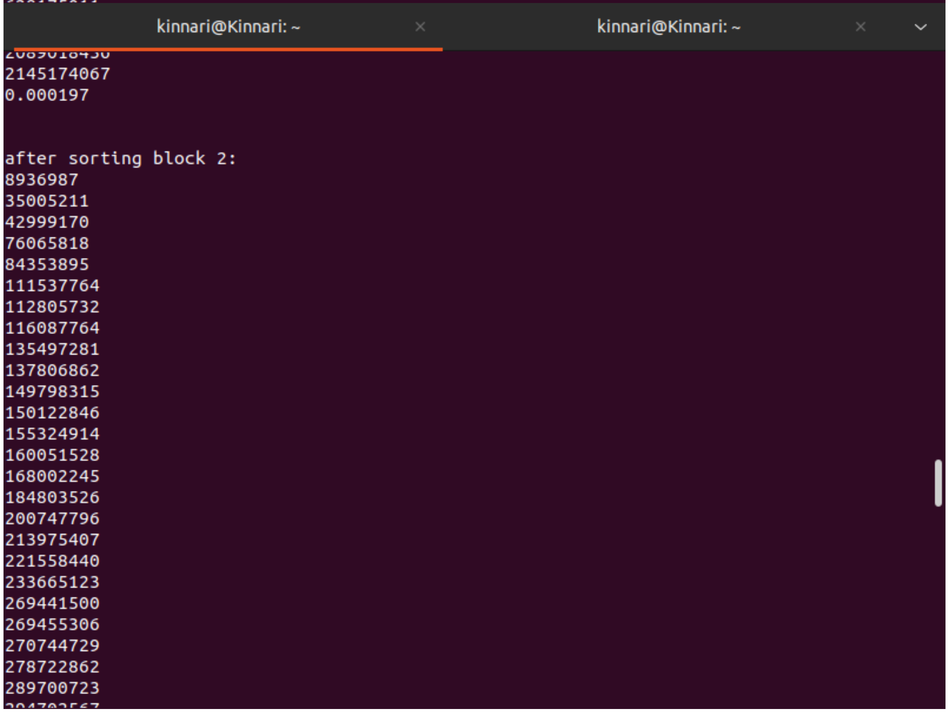
```
{
FILE* ptr;
// file in reading mode
ptr = fopen("input.txt", "r");
if (NULL == ptr)
{
printf("file can't be opened \n");
}
int block=1;
int size=100;
while(block<=1000)
{
int data[size];
for(int i=0;i<size;i++)
{
fscanf(ptr,"%d ",&data[i]);
//printf("%d ",data[i]);
}

clock_t t;
t = clock();
insertionSort(data,size);
if(block<3) //this prints sorted first 2 blocks only
{
printf("\n\nafter sorting block %d:\n",block);
for(int i=0;i<size;i++)
{
printf("%d \n",data[i]);
}
}

t = clock() - t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
printf("\nRuntime for block number %d : %f\n",block,time_taken);
size=size+100;
block++;
fseek(ptr,0,SEEK_SET); //moving cursor again to start pointer of txt file
}
fclose(ptr);
}
```

RESULT:

Selection Sort

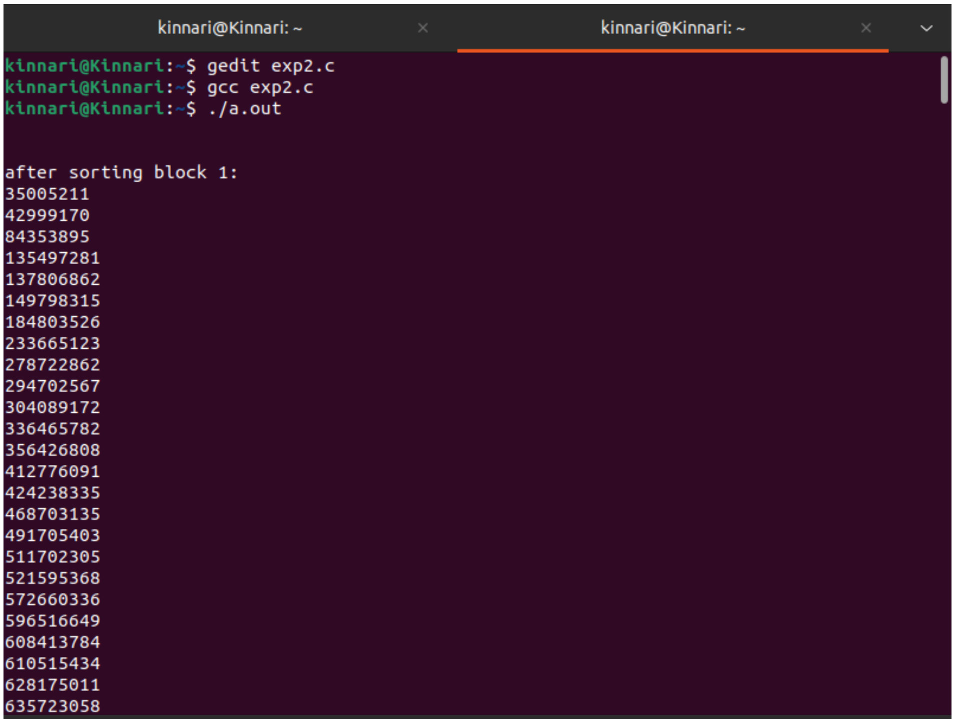


```
Runtime for block number 989 : 17.387817
^[[A^[[A^C
kinnari@Kinnari:~$ gedit exp1.c
kinnari@Kinnari:~$ gcc exp1.c
kinnari@Kinnari:~$ ./a.out


after sorting block 1:
35005211
42999170
84353895
135497281
137806862
149798315
184803526
233665123
278722862
294702567
304089172
336465782
356426808
412776091
424238335
468703135
491705403
511702305
521595368
572660336
596516649
608413784
610515434
```

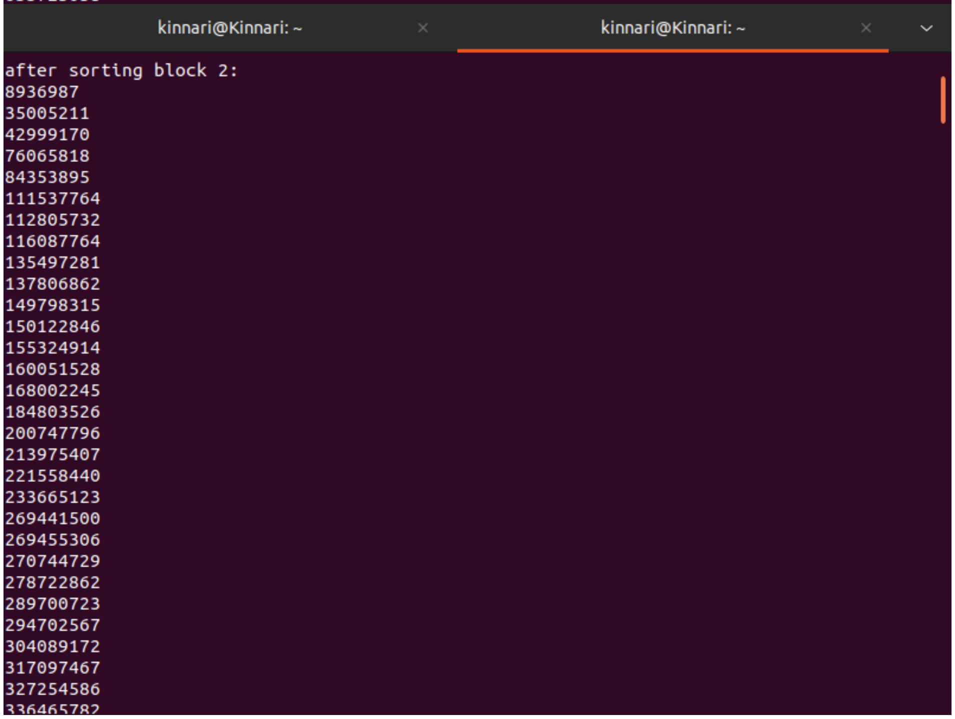


```
2089018430
2145174067
0.000197


after sorting block 2:
8936987
35005211
42999170
76065818
84353895
111537764
112805732
116087764
135497281
137806862
149798315
150122846
155324914
160051528
168002245
184803526
200747796
213975407
221558440
233665123
269441500
269455306
270744729
278722862
289700723
```

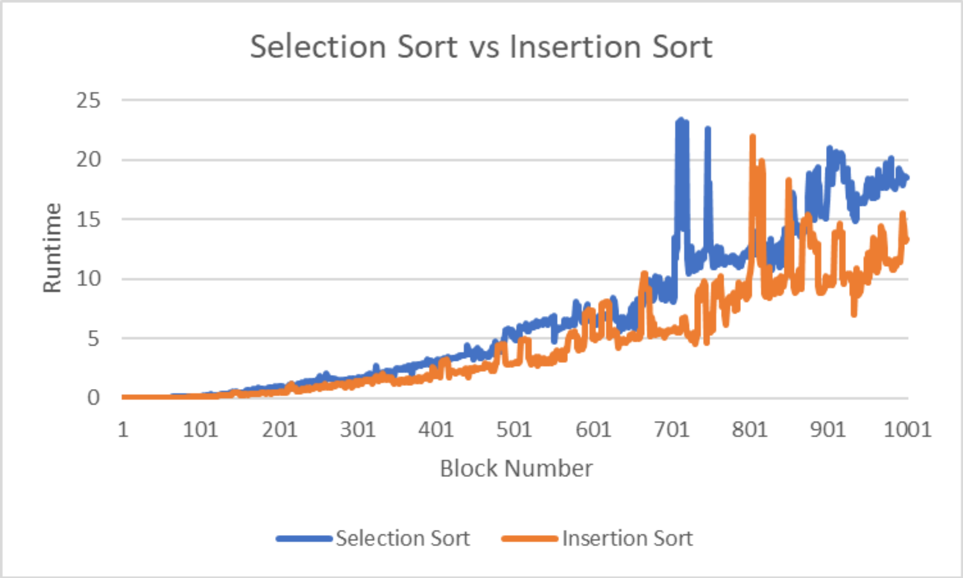
Insertion Sort



```
kinnari@Kinnari:~$ gedit exp2.c
kinnari@Kinnari:~$ gcc exp2.c
kinnari@Kinnari:~$ ./a.out


after sorting block 1:
35005211
42999170
84353895
135497281
137806862
149798315
184803526
233665123
278722862
294702567
304089172
336465782
356426808
412776091
424238335
468703135
491705403
511702305
521595368
572660336
596516649
608413784
610515434
628175011
635723058
```



```
after sorting block 2:
8936987
35005211
42999170
76065818
84353895
111537764
112805732
116087764
135497281
137806862
149798315
150122846
155324914
160051528
168002245
184803526
200747796
213975407
221558440
233665123
269441500
269455306
270744729
278722862
289700723
294702567
304089172
317097467
327254586
336465782
```



Observation:

From above graph ,it is clearly visible that the running time for selection sort algorithm representing blue line is more than the running time for insertion sort algorithm.

The insertion sort inserts the values in a pre sorted file to sort a set of values. On the other hand, the selection sort finds the minimum number from the list and sort it in some order which requires multiple scanning of the array and therefore more comparisons. As a result the best case time complexity of selection sort is o(n^2).On the other hand in insertion sort the number of times an element is moved or swapped is greater than the comparisons made. Due to this time complexity in best case for insertion sort is O(n).

Among both of the sorting algorithm, the insertion sort is fast, efficient, stable while selection sort only works efficiently when the small set of elements is involved or the list is partially previously sorted.

CONCLUSION: In this experiment, we implemented 10 functions in C program and drew a graph for those functions in the values between 0 to 100 and also compared the run time for selection sort and insertion sort.