**1. Topic – Plotting Graph of Execution times of Pushing Different n Number of Random Numbers using Growable Stack Concept**

* **Problem Statement**

**Write a C program to create a Stack using array. Use Growth strategy to increase the size of the array. From main program generate n random numbers  and push them in the stack. Report the time taken for different n values. For example n = 100, 500, 1000, 10000, 25000, 50000 pushes. Plot a graph of n Vs. time.**

**Input example:** /\* Here user will give the number of observatons and after that will enter how many random numbers are to be pushed\*/

Enter the number of observations

5

Enter the total number of items to be pushed

5000

Enter the total number of items to be pushed

10000

Enter the total number of items to be pushed

25000

Enter the total number of items to be pushed

50000

Enter the total number of items to be pushed

100000

**Output example:**

The total time of execution for pushing 5000 random numbers is 0.0001800000 seconds

The total time of execution for pushing 10000 random numbers is 0.0003020000 seconds

The total time of execution for pushing 25000 random numbers is 0.0008550000 seconds

The total time of execution for pushing 50000 random numbers is 0.0016290000 seconds

The total time of execution for pushing 100000 random numbers is 0.0030400000 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*The graph of value of total items vs execution time (scale 1:10000) is:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \* \*

\* \* \* \*

\* \* \* \* \*

**5000 10000 25000 50000 100000**

* **Proposed C Code**

**/\* ---------- stackgraph.c--------------- \*/**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int main()

{

int n;

// n is the number of observations i.e how many execution times have to be noted for the plot of graph

int num;

//num is the number of elements to be pushed in a particular observation

int data;

int top = -1,size=1;

printf("Enter the number of observations\n");

scanf("%d",&n);

int \*frequency = (int\*)malloc(n\*sizeof(int));

int \*arr = (int\*)malloc(n\*sizeof(int));

// arr and frequency pointer for plotting graph of number of elements vs the time to push them

double\* execution\_time = (double\*)malloc(n\*sizeof(double));

// dynamically creating array to store exact times to push different number of random numbers in a stack

int max=0;

// max variable is defined for finding the maximum of the execution times which will be stored in frequency array

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

{

printf("Enter the total number of items to be pushed\n");

scanf("%d",&num);

arr[i]=num;

//storing the numbers corresponding to the number of random numbers for the graph

srand(time(0));

int\*stack;

stack= (int\*)malloc(size\*sizeof(int));

//stack variable to create a stack dynamically with initial size = 1

clock\_t start\_time,end\_time;

start\_time = clock();

// starting the clock to see execution time

for( int j = 0 ; j < num ; j++ )

{

data = rand();

// creating random numbers

// Algorithm for the growth strategy to increase the size of stack

if(top==size-1)

{

size = 2\*size;

int\* newstack;

newstack = (int\*)malloc(size\*sizeof(int));

// copying the elements of stack to newstack which has double size of previous stack

for(int i=0; i<=top; i++)

{

newstack[i] = stack[i];

}

stack = newstack;

// now stack is pointing to the elements of the newstack,size of stack is doubled

}

stack[++top]=data;

// pushing data to stack

}

end\_time = clock();

execution\_time[i] = (double)(end\_time - start\_time)/CLOCKS\_PER\_SEC;

// ending the clock to determine execution time

frequency[i] = execution\_time[i]\*10000;

//storing the proper scaled execution time in integer frequency array by multiplying with 10000

if(max<frequency[i])

{

max=frequency[i];

// finding the maximun of frequency array

}

free(stack);

// freeing the stack

}

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

{

printf("The total time of execution for pushing %d random numbers is %0.10f seconds\n",arr[i],execution\_time[i]);

// printing the times to push different random numbers

}

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*The graph of value of total items vs execution time (scale 1:10000) is:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

int storemax = max;

// storing the maximum exeecution time in storemax

// Plotting the vertical graph of different n random numbers vs the time to push them

for (int j = 0; j < storemax ; j++)

{

for ( int k = 0 ; k < n ; k++ )

{

if(max>frequency[k])

printf("\t");

else

printf("\* ");

}

printf("\n");

max--;

}

for (int j = 0; j < n ; j++)

{

printf("%d\t",arr[j]);

}

free(frequency);

free(arr);

free(execution\_time);

// freeing the memory pointed by frequency,arr and execution\_time pointers

return 0;

}

**/\*------------------------------------------------------------------------------------------------------------------------- \*/**

* **Conclusion**

**The proposed algorithm has overall runtime of O(n^2) where n is number of observations under consideration but if we consider the algorithm for only the time complexity of pushing the elements to the growable stack in this growth strategy then it will be O(n) where n is the number of elements to be pushed.**

* **Limitations : As the execution time is order of microseconds for n = 100,200 etc and 0.003 seconds for n=100000,200000 so it will be difficult to show their execution times simultaneously in a single graph. For n=100,200,500 if we use scale 1:100000 then it will be properly visible in graph again for n = 100000, 200000 if we use scale 1:1000 then it will be properly visible in graph.**
* **Assumptions: Here we are assuming the input size to be large i.e more than 5000 like 5000,10000,25000 etc and the execution time is multiplied by 10000 i.e in 1:10000 scale so that the graph will be properly visible.**