**2.Stack**

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 :

1. 6

100

500

1000

10000

25000

50000

• Output example :

For n = 100 time needed 7 sec

For n = 500 time needed 23 sec

For n = 1000 time needed 42 sec

For n = 10000 time needed 421 sec

For n = 25000 time needed 1147 sec

For n = 50000 time needed 2023 sec

The n vs time graph-->

\*\*\*

\*\*\*

\*\*\* \*\*\*

\*\*\* \*\*\*

\*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*

\*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*

100 500 1000 10000 25000 50000

(As the size of the graph is very large so I created it’s small miniature)

Proposed C Code:

/\* ------- main.c ------- \*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int \*stack;

int top = -1;

int size = 1;

void init()

{

stack = (int \*)calloc(size, sizeof(int));

}

int \*create(int \*stack)

{

int \*new\_stack = (int \*)calloc(2 \* size, sizeof(int));

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

{

new\_stack[i] = stack[i];

}

size \*= 2;

return new\_stack;

}

void push(int item)

{

if (top == size - 1)

{

stack = create(stack);

}

stack[++top] = item;

}

int pop()

{

if (top == -1)

{

printf("Stack is Empty\n");

return -1;

}

return stack[top--];

}

void display()

{

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

{

printf("%d\t", stack[i]);

}

printf("\n");

return;

}

double time\_required(int n)

{

double time\_spent = 0.0;

clock\_t begin = clock();

init();

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

{

push(rand());

}

clock\_t end = clock();

time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

return time\_spent;

}

int main()

{

int p;

scanf("%d", &p);

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

int \*time = (int \*)malloc(p \* sizeof(int));

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

{

scanf("%d", &n[i]);

time[i] = (int)(time\_required(n[i]) \* 1000000);

}

int max = -1;

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

{

printf("For n = %d time needed %d sec\n",n[i],time[i]);

if (time[i] > max)

{

max = time[i];

}

}

printf("The n vs time graph-->\n");

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

{

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

{

if (i < max - (int)time[j])

{

printf(" ");

}

else

{

printf(" \*\*\* ");

}

}

printf("\n");

}

for(int i=0;i<p;i++){

printf(" %d",n[i]);

}

return 0;

}

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

Conclusion:

The proposed algorithm has a runtime of O(p\*n).

Limitations and assumptions for this algorithm include:

1. The time complexity of this program is of the order 2 rather than linear.
2. Here whenever the stack gets full we are creating a new stack then assign it to previous one. So here too much space is wasted and the space complexity is high.