Advanced Algorithms Assignment -1

Question: Create a data structure to hold all information required for a dynamic table.
- pointer to the data (data could be int)
- max size of the table
- current size
- factor by which to increase the table size on insertion into a table, which is full
- factor at which to decrease the size on deletion

Approach

The data structure implemented was a Stack (implemented by an array, pointer: 'arrPointer') with two major operations: push and pop. The other operations are init and display. *Although 'display' isn't strictly speaking a Stack operation, it is present for debugging and result checking purposes only.* The data structure definition, named 'dt' along with the function definitions are shown beside.

```
typedef struct dynamicTable
{
    int* arrPointer;
    int maxSize;
    int curSize;
    double incrFact;
    double decrFact;
}dt;

void init(dt*,double,double);
void push(dt*,int);
int pop(dt*);
void display(dt*);
```

'init()' takes three parameters, reference to dynamic table object, the 'increment' factor – 'incrFact' and the 'decrement' factor – 'decrFact'.

The 'arrPointer' in 'dt' is then initialized with a size of 5 in my implementation.

Push operations happen in a regular manner until max size is reached. Once stack is full, the following steps take place:

1. Create a new array with size: maxSize * incrFact with 'malloc'.
2. Copy the elements of old array (pointed to by 'arrPointer' in the 'dt' object) and free the 'arrPointer'.
3. Make the old pointer (i.e. 'arrPointer') point to the new array. Set the original pointer of new array to NULL.

Pop operations again are handled appropriately, until the size of stack reduces beyond a certain factor. The factor is calculated as follows and appropriate resizing is done:

1. Calculate the check size by dividing 'maxSize' and 'decrFact'. If the size of stack reaches 'check size', then proceed to step 2.
2. Check if current size is equal to the check size and resize using realloc to check size.

Experiments and Results

One million operations performed in total and they were done with an array, having 0's and 1's, which indicates the operations namely 0-pop, 1-push. The contents of this array were generated and randomized (with shuffle()) with Python. The ratio of number of pushes to number of pops were taken to be 3:2 and 4:2. And for each ratio, different increment and decrement factors were used to see the most optimal one.

Results for Pushes to Pops ratio 3:2

The number of pushes were 600000 and pops were 400000, totalling 1 million, but having made sure that pushes occur before pops. The different times and graphs are as shown in next page.

3:2 ratio of Pushes to Pops, Totalling 1 Million Operations.					
Time in Milliseconds	Max Time for Insertion	Max Time for Deletion	Average time for Insetion	Average time for Deletion	Average time of all operation
With Increment Factor as:	All Times in Milliseconds				
1.25	0.69219	0.00217	3.00E-05	2.40E-05	2.80E-05
1.5	0.51306	0.00156	2.70E-05	2.30E-05	2.60E-05
1.75	0.46827	0.0024	2.70E-05	2.40E-05	2.50E-05
2	0.61431	0.0018	2.70E-05	2.40E-05	2.60E-05
3	0.36975	0.00173	2.60E-05	2.40E-05	2.50E-05
With Decrement Factor as:	All Times in Milliseconds				
0.25	0.31125	0.0018	2.60E-05	2.40E-05	2.50E-05
0.5	0.60516	0.0016	2.70E-05	2.40E-05	2.60E-05
0.75	0.55411	0.00155	2.90E-05	2.40E-05	2.70E-05

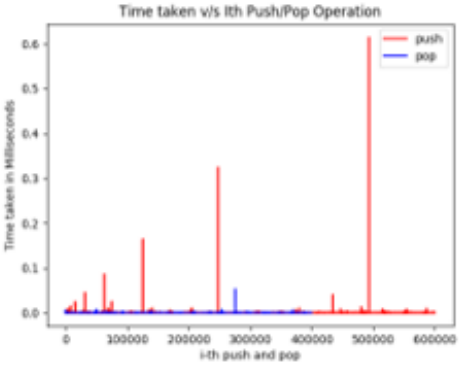

Figure 1With Increment Factor as 2, same as Decrement factor 0.5

Figure 2 With Increment factor as 3.

Results for Pushes to Pops ratio 4:2

Pushes were 666664 and pops were 444442, totalling ~1 million.

4:2 ratio of Pushes to Pops, Totalling ~1 Million Operations (999996 operations)					
Time in Milliseconds	Max Time for Insertion	Max Time for Deletion	Average time for Insetion	Average time for Deletion	Average time of all operation
With Increment Factor as:	All Times in Milliseconds				
1.25	1.075	0.00146	3.30E-05	2.40E-05	3.00E-05
1.5	1.13962	0.00221	3.00E-05	2.40E-05	2.80E-05
1.75	0.80547	0.00163	2.80E-05	2.40E-05	2.70E-05
2	1.22396	0.00383	2.90E-05	2.40E-05	2.70E-05
3	1.05315	0.00148	2.70E-05	2.40E-05	2.60E-05
With Decrement Factor as:	All Times in Milliseconds				
0.25	1.22491	0.00169	2.70E-05	2.40E-05	2.60E-05
0.5	1.20854	0.00255	2.90E-05	2.40E-05	2.70E-05
0.75	0.98641	0.00303	3.10E-05	2.40E-05	2.80E-05

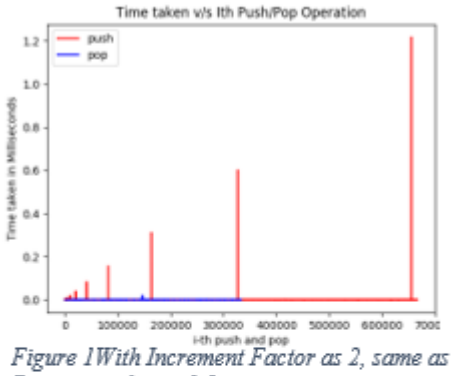

Figure 1With Increment Factor as 2, same as Decrement factor 0.5

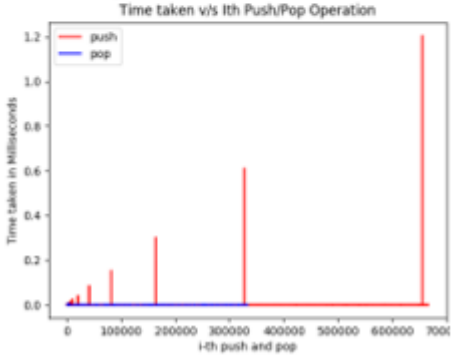

Figure 2 With Increment factor as 3.

Conclusion

Upon observing the Average Insertion times for 3:2 ratio, we see, although factor 3 seems to be performing better in insertion, factor 0.5 seems to perform better in deletion (i.e. the increment factor is 2), thus concluding that if we know beforehand that the number of pushes to pops will be 3:2, then using a common factor of 2 (for insertion, i.e. 0.5 for deletion) will be favourable.
Similarly, for 4:2, a common factor of 3 seems suitable (for insertion) if we know some information on the input beforehand or after pre-precessing.