Advanced Algorithms (UE15CS311)
Assignment - 1

Approach to the problem:

To create a dynamic table I used Flexible Array Member(FAM, a feature introduced in C99). This enables us to declare an array member without dimensions inside a structure. Such an array inside the structure should preferably be declared as the last member of structure and its size is variable(can be changed be at runtime).

The structure used is:

```
typedef struct dynamic_table
{
        size_t current_size;
        size_t maximum_size;
        float factor_decrease;
        float factor_increase;
        int array[];
}d_table;
```

Thus, a malloc for the above structure would look like(for creating an array of size 1):

```
d_table *dt = malloc( sizeof ( d_table ) + sizeof( int ) )
```

There are two operations being applied on the array, namely:

1. Insertion: The function for insertion is:

```
void insert(int data,float init_factor_increase,float init_factor_decrease){
        if(dt->maximum_size == 0)
        {
                free(dt);
                init(init_factor_increase,init_factor_decrease);
        }
        else if(dt->current_size == dt->maximum_size)
        {
                dt->maximum_size = ceil(dt->maximum_size * dt->factor_increase);
                dt = realloc(dt,(sizeof(d_table) + sizeof(int) * (dt->maximum_size)));
                dt->array[dt->current_size] = data;
                dt->current_size++;
        }
        else
        {
                dt->array[dt->current_size] = data;
                dt->current_size++;
        }
}
```

In this function realloc occurs every time the load factor reaches 1 i.e the table is allocated a new size depending upon the factor of increase.
If there is a contiguous block of memory available at the location the memory is allocated and the pointer to the same location is returned. On the other hand if the required memory cannot be allocation at the current location the memory is allocated at a different location and the pointer to that location is returned.
The values that are being inserted are values b/w 0-9 and are generated randomly.

2. Deletion: The function for deletion looks like:

```
void delete()
{
        if(dt->current_size>=1)
        {
                dt->array[dt->current_size - 1] = INT_MIN;
                dt->current_size--;
                if(dt->current_size <= dt->maximum_size * dt->factor_decrease)
                {
                        dt->maximum_size = floor(dt->maximum_size * dt->factor_decrease);
                        dt = realloc(dt,(sizeof(d_table) + sizeof(int) * (dt->maximum_size)));
                }
        }
        //else
        //      printf("Cannot delete, the table is empty\n");
}
```

The number of operations for each ratio and factors combination is 1 million.
The likelihood of the two operations occurring are varied(equally likely, 3:2, 4:2).

Observations:

If the operations are likely:

Factor of decrease Factor of increase	0.25	0.5	0.75
1.25	0.015513 0.000035 0.024072 0.000026	0.010899 0.000031 0.013203 0.000023	0.012798 0.000032 0.015202 0.000024
1.5	0.010895 0.000031 0.010944 0.000023	0.056236 0.000031 0.010929 0.000023	0.029797 0.000032 0.010949 0.000024
1.75	0.013452 0.000031 0.010801 0.000024	0.011207 0.000030 0.025997 0.000023	0.024642 0.000031 0.017885 0.000024
2	0.011106 0.000030 0.016169 0.000024	0.017799 0.000031 0.014564 0.000024	0.026129 0.000031 0.010784 0.000024
3	0.011327 0.000030 0.010738 0.000023	0.011484 0.000031 0.010849 0.000023	0.022871 0.000031 0.011479 0.000025

The entries in each cell are in the following order:

1. Maximum time for insertion
2. Average time for insertion
3. Maximum time for deletion
4. Average time for deletion

If the operations are in 3:2 ratio:

Factor of decrease Factor of increase	0.25	0.5	0.75
1.25	0.011222 0.000065 0.010783 0.000024	0.010770 0.000066 0.012882 0.000026	0.010847 0.000067 0.011471 0.000026
1.5	0.043990 0.000065 0.010834 0.000024	0.039827 0.000065 0.011151 0.000025	0.036389 0.000065 0.023941 0.000025
1.75	0.014284 0.000065 0.011241 0.000024	0.010729 0.000064 0.010854 0.000025	0.029602 0.000065 0.011356 0.000025
2	0.049612 0.000065 0.011547 0.000024	0.018827 0.000065 0.011103 0.000025	0.010818 0.000063 0.010751 0.000025
3	0.044695 0.000063 0.016288 0.000024	0.034098 0.000065 0.010802 0.000025	0.039204 0.000061 0.014356 0.000025

The entries in each cell are in the following order:

1. Maximum time for insertion
2. Average time for insertion
3. Maximum time for deletion
4. Average time for deletion

If the operations are in the 4:2 ratio:

Factor of decrease Factor of increase	0.25	0.5	0.75
1.25	0.037760 0.000056 0.011548 0.000027	0.036478 0.000060 0.013315 0.000031	0.035829 0.000060 0.015773 0.000031
1.5	0.034892 0.000057 0.011143 0.000027	0.041382 0.000061 0.010869 0.000031	0.049220 0.000061 0.011683 0.000031
1.75	0.046988 0.000057 0.011527 0.000027	0.049103 0.000061 0.010808 0.000031	0.036793 0.000059 0.019658 0.000031
2	0.037710 0.000057 0.011619 0.000028	0.038527 0.000061 0.011479 0.000031	0.053986 0.000061 0.017813 0.000032
3	0.037954 0.000056 0.010671 0.000027	0.033986 0.000061 0.011303 0.000031	0.048853 0.000056 0.011383 0.000032

The entries in each cell are in the following order:

1. Maximum time for insertion
2. Average time for insertion
3. Maximum time for deletion
4. Average time for deletion

Analysis:

There is an insignificant computational time difference in the average time taken by the operations to run across the different ratios even though it shows an increasing trend. The maximum time of insert and delete vary more across different ratios.

Conclusion:

This report compares simulations run on dynamic tables for various increasing and decreasing factors. The results obtained are subjective. To obtain the true optimum values this simulation must be run multiple times and the average of the averages must be computed. This report is mainly helpful in assessing the trend followed by the various combination of factors.