Experiment 1

AA Experiment 1. DATE:

Kreena Shah
60004210243
Advance Algorithm

Aim : Experiment on Amortized Analysis (Aggregate method)

Theory :

Amortized Analysis is used for algorithms where an occasional operation is very slow, but most of the other operations are faster.

Here, we analyze a sequence of operations & guarantee a worst-case average time that is lower than the worst time of a particularly expensive operation

The idea is to spread the cost of these expensive operations over multiple operations, so that the average cost of each operation is constant or less.

It requires a knowledge of which series of operations are possible. This is the case with data structures, which have state that persists between operations.

The basic idea is that a worst case operation can alter the state in such a way that the worst case cannot occur again for a long time, thus "amortizing" its cost.

Following are the methods to perform amortized analysis :

(1) Aggregate method
(2) Accounting method
(3) Potential method

DATE:

Amortized Analysis using Aggregate Method : Multipop Stack

We determine an upper bound $T(n)$ on the total cost of a sequence of n-operations.
In worst case : amortized cost per operation $\Rightarrow T(n)/n$
The amortized cost applies to each operation ; even when there are several types of operations in the sequence

Operations in Multipop Stack
- Push (x, S)
- Pop ( )
- Multipop (S, K)

Let us analyze a sequence of n Push, Pop, Multipop operations on an initially empty stack.
The worst case cost of a Multipop operation in the sequence is $O(n)$, since the stack is at most of 'n' size
The worst case time of any stack operation is $O(n)$ & hence a sequence of n operations cost $O(n^2)$, since we have $O(n)$ multipop operations costing $O(n)$ each
Although this analysis is correct, the $O(n^2)$ result, obtained by considering the worst case cost of each operation individually, is not tight

Each object can be popped at most one time for each time it is pushed
The no. of push operations is $O(n)$ at most
So, the no. of pops either from pop or multipop, is at most $O(n)$

Overall Complexity : $O(n)$
Amortized Cost : $O(n)/n \Rightarrow O(1)$

Conclusion : Thus, we understood & performed amortized analysis using aggregate method.

Code :

```
#include <bits/stdc++.h>
using namespace std;

class Multipop {
    public:
    int c = 0;
    vector <int> stack;

    void push(int a) {
        stack.push_back(a);
        for (int i: stack) cout << "|" << i;
        cout << "|" << endl;
        c++;
    }

    void pop(int k) {
        for (int i = 0; i < k; i++) {
            if (stack.size() > 0) {
                cout << stack.back() << " popped." << endl;
                stack.pop_back();
                for (int i: stack) cout << "|" << i;
                cout << "|" << endl;
            }
            c++;
        }
    }

    void multipop(vector<int> arr) {
        for (int i = 0; i < arr.size(); i++) {
            if(arr[i] >= stack.size()) {
                cout << arr[i] << " pushed." << endl;
                push(arr[i]);
            }
            else {
                cout << "Multi pop called " << arr[i] << " times." << endl;
                pop(arr[i]);
                cout << arr[i] << " pushed." << endl;
                push(arr[i]);
            }
        }
        cout << "Asymptotic cost: " << c << endl;
        cout << "Amortized cost: " << c / arr.size();
    }
};

int main() {
    int n;
    cout << "Enter array size: ";
    cin >> n;
    vector <int> a(n,0);
```

```
    cout << "Enter array elements: " << endl;;
    for (int i = 0; i < n; i++) cin >> a[i];
    Multipop m;
    m.multipop(a);
    return 0;
}
```

Output:

```
Enter array size: 5
Enter array elements:
5
6
7
2
9
5 pushed.
|5|
6 pushed.
|5|6|
7 pushed.
|5|6|7|
Multi pop called 2 times.
7 popped.
|5|6|
6 popped.
|5|
2 pushed.
|5|2|
9 pushed.
|5|2|9|
Asymptotic cost: 7
Amortized cost: 1
```