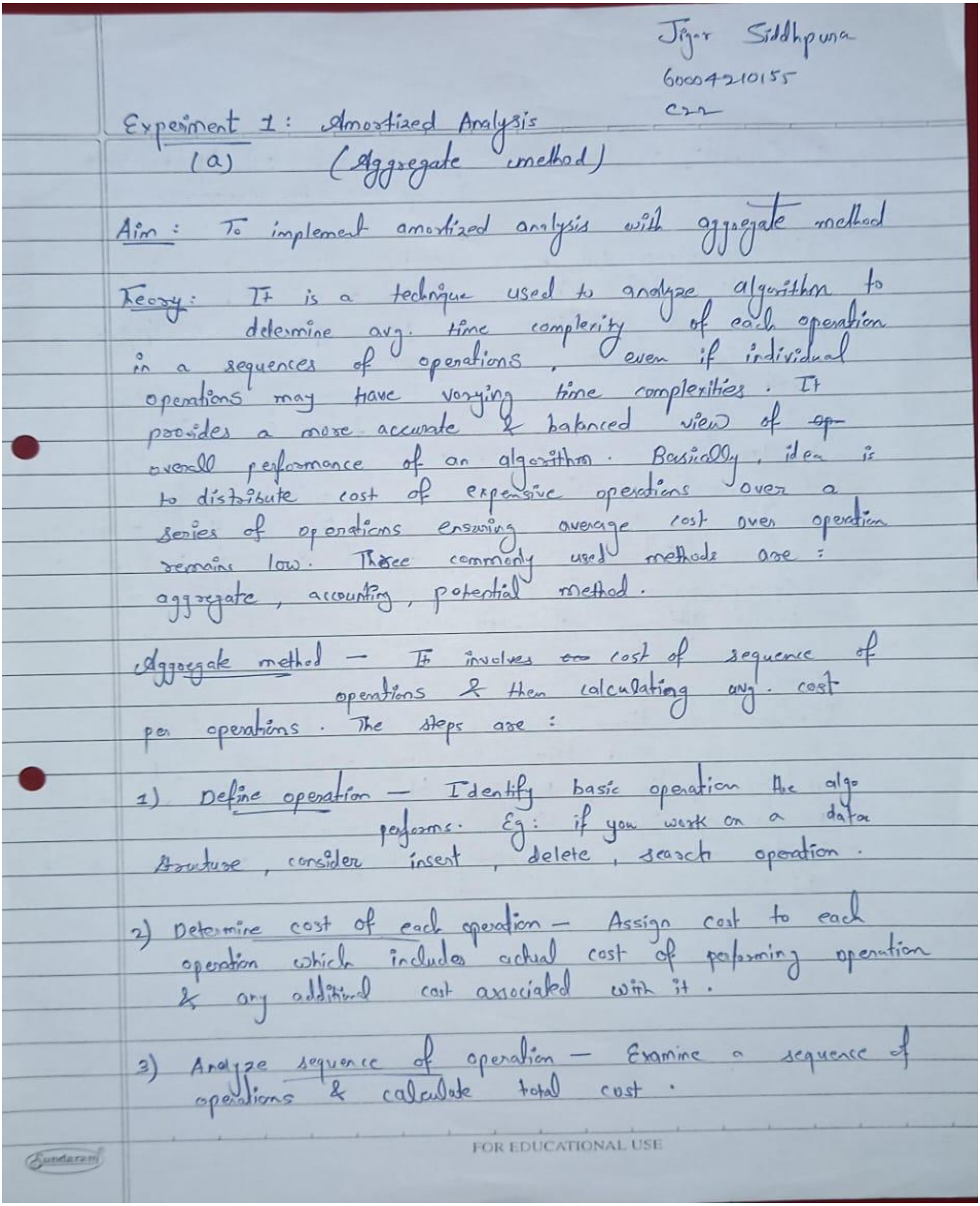
Name: Jigar Siddhpura **SAPID:** 60004200155

DIV: C/C2 **Branch:** Computer Engineering

AA - Experiment 1A - Amortized Analysis



Jigar Siddhpura
60004210155
C22

Experiment 1: Amortized Analysis
(a) (Aggregate method)

Aim : To implement amortized analysis with aggregate method

Theory: It is a technique used to analyze algorithm to determine avg. time complexity of each operation in a sequences of operations, even if individual operations may have varying time complexities. It provides a more accurate & balanced view of ~~op~~ overall performance of an algorithm. Basically, idea is to distribute cost of expensive operations over a series of operations ensuring average cost over operation remains low. Three commonly used methods are: aggregate, accounting, potential method.

Aggregate method — It involves cost of sequence of operations & then calculating avg. cost per operations. The steps are:

1) Define operation — Identify basic operation the algo performs. Eg: if you work on a data structure, consider insert, delete, search operation.

2) Determine cost of each operation — Assign cost to each operation which includes actual cost of performing operation & any additional cost associated with it.

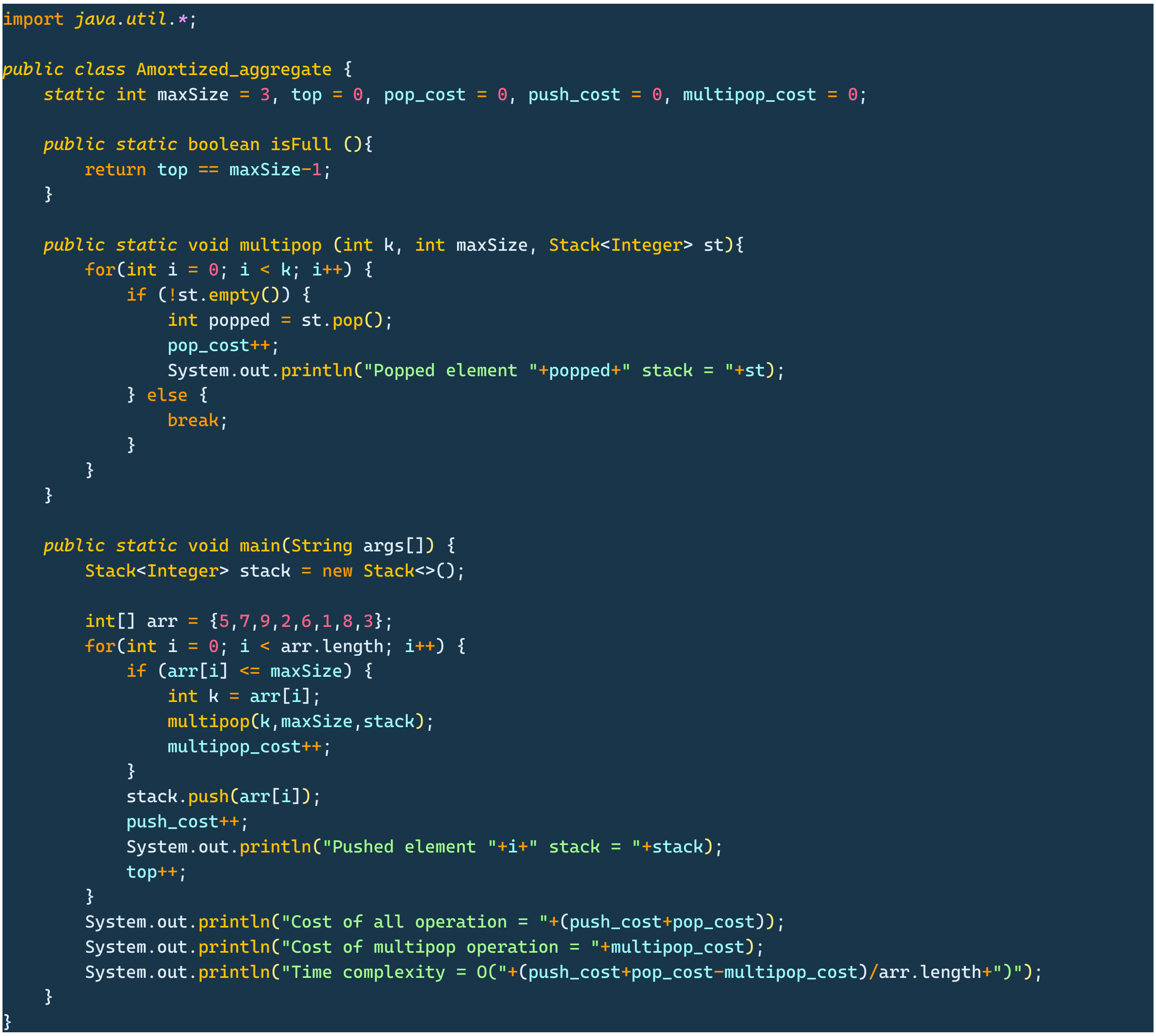
3) Analyze sequence of operation — Examine a sequence of operations & calculate total cost.

4) Calculate amortized cost — Divide total cost by numbers of operation in sequence.

The code implemented shows amortized analysis based on aggregate method which is applied on a stack data structure, & operations are push, pop & multipop. Goal is to maintain stack operations within given length constraint & analyze their amortized analysis. So a loop is run on a array of element, if value is less than or equal to length of stack, multipop is performed. The total no. of units spent on operation is tracked at the end the amortized complexity is calculated as T(n) = sum of units / total no. of operations. It provides concise representation of overall efficiency of stack operations.

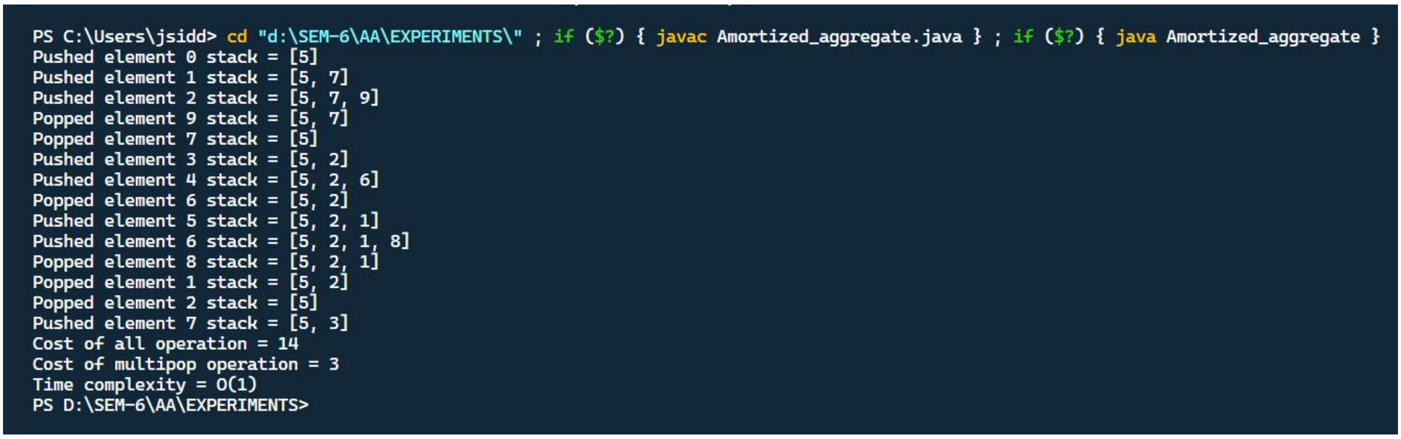
Conclusion — Hence we peformed amortized analysis on stack data structure with push, pop & multipip operation by tracking total units spend & total no. of operations. Final result provides valuable insights into average efficiency of operations over sequence of input.

CODE :



```java
import java.util.*;

public class Amortized_aggregate {
    static int maxSize = 3, top = 0, pop_cost = 0, push_cost = 0, multipop_cost = 0;

    public static boolean isFull (){
        return top == maxSize-1;
    }

    public static void multipop (int k, int maxSize, Stack<Integer> st){
        for(int i = 0; i < k; i++) {
            if (!st.empty()) {
                int popped = st.pop();
                pop_cost++;
                System.out.println("Popped element "+popped+" stack = "+st);
            } else {
                break;
            }
        }
    }

    public static void main(String args[]) {
        Stack<Integer> stack = new Stack<>();

        int[] arr = {5,7,9,2,6,1,8,3};
        for(int i = 0; i < arr.length; i++) {
            if (arr[i] <= maxSize) {
                int k = arr[i];
                multipop(k,maxSize,stack);
                multipop_cost++;
            }
            stack.push(arr[i]);
            push_cost++;
            System.out.println("Pushed element "+i+" stack = "+stack);
            top++;
        }
        System.out.println("Cost of all operation = "+(push_cost+pop_cost));
        System.out.println("Cost of multipop operation = "+multipop_cost);
        System.out.println("Time complexity = O("+(push_cost+pop_cost-multipop_cost)/arr.length+")");
    }
}
```

OUTPUT :



```
PS C:\Users\jsidd> cd "d:\SEM-6\AA\EXPERIMENTS\" ; if ($?) { javac Amortized_aggregate.java } ; if ($?) { java Amortized_aggregate }
Pushed element 0 stack = [5]
Pushed element 1 stack = [5, 7]
Pushed element 2 stack = [5, 7, 9]
Popped element 9 stack = [5, 7]
Popped element 7 stack = [5]
Pushed element 3 stack = [5, 2]
Pushed element 4 stack = [5, 2, 6]
Popped element 6 stack = [5, 2]
Pushed element 5 stack = [5, 2, 1]
Pushed element 6 stack = [5, 2, 1, 8]
Popped element 8 stack = [5, 2, 1]
Popped element 1 stack = [5, 2]
Popped element 2 stack = [5]
Pushed element 7 stack = [5, 3]
Cost of all operation = 14
Cost of multipop operation = 3
Time complexity = O(1)
PS D:\SEM-6\AA\EXPERIMENTS>
```