**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJ19ITL502 DATE:29/9/22**

**COURSE NAME: Advanced Data Structures Laboratory CLASS/Div: A**

**SAP ID:60003200076 Academic Year 2022-23**

**EXPERIMENT NO. 1**

**CO/LO: LO1**

## AIM: To perform amortized analysis for stack operations

**Theory:**

**Amortized analysis** : The concept of amortized analysis is to compute the time complexity on a sequence of operations rather than finding a single case of worst case time ,which maybe very high but rare within the sequence.

In amortized analysis ,both the costly and less costly operations are considered together on a sequential run of the algorithm and compute their average ,which often provides a tighter bound of worst case complexity.

In amortized analysis ,time required to perform a sequence of data structure operations is averaged over all the successive operations performed .That is ,a large cost of one operation is spread out over many operations(amortized), where the others are less expensive.

There are three methods for amortized analysis viz

1. Aggregate method:
2. Accounting method:
3. Potential method:

**Aggregate method:**

In aggregate analysis, there are two steps. First, we must show that a sequence of n operations takes T(n) time in the worst case. Then, we show that each operation takes T(n)/n time, on average. Therefore, in aggregate analysis, each operation has the same cost

**Accounting method:**

Here, each operation is assigned a charge, called the amortized cost. Some operations can be charged more or less than they actually cost. If an operation's amortized cost exceeds its actual cost, we assign the difference, called a credit, to specific objects in the data structure. Credit can be used later to help pay for other operations whose amortized cost is less than their actual cost. Credit can never be negative in any sequence of operations.

**Potential method:**

The potential method is similar to the accounting method. However, instead of thinking about the analysis in terms of cost and credit, the potential method thinks of work already done as potential energy that can pay for later operations

**Program:**

Aggregate method:

import numpy

import random as r

cost=0

s=[]

ele=1

def push(ele):

    global cost

    s.append(ele)

    cost=cost+1

def pop():

    global cost

    if len(s)!=0:

        print(s[len(s)-1])

        del s[len(s)-1]

        cost=cost+1

    else:

        print("underflow")

def multipop(no):

    global cost

    n=min(no,len(s))

    for i in range(0,n):

        pop()

#k=int(input("enter no of operations : "))

k=5

operations=numpy.random.uniform(0,3,k)

o=operations.astype(int)

for i in o:

    if i==0:

        push(ele)

    elif i==1:

        pop()

    elif i==2:

        x=r.randrange(1,10)

        print("muptipopin ",x," times")

        multipop(x)

print("0: push \t1: pop\t2: multipop")

print("operations : ", o)

print("aggregate amortized cost : ",cost/k)

Accounting method:

import numpy

import random as r

cost=0

c=[]

s=[]

ele=1

def push(ele):

    global cost

    s.append(ele)

    cost=cost+2

    if(len(c)!=0):

        c.append(1+c[len(c)-1])

    else:

        c.append(1)

def pop():

    global cost

    if len(s)!=0:

        print(s[len(s)-1])

        del s[len(s)-1]

        cost=cost+0

        c.append(c[len(c)-1]-1)

    else:

        print("underflow")

def multipop(no):

    global cost

    n=min(no,len(s))

    for i in range(0,n):

        pop()

#k=int(input("enter no of operations : "))

k=5

operations=numpy.random.uniform(0,3,k)

o=operations.astype(int)

for i in o:

    if i==0:

        push(ele)

    elif i==1:

        pop()

    elif i==2:

        x=r.randrange(1,10)

        print("muptipopin ",x," times")

        multipop(x)

print("0: push \t1: pop\t2: multipop")

print("operations : ", o)

print("accounting amortized cost : ",c)

Potential method:

import numpy

import random as r

pe=0

p=[]

cost=0

acost=0

s=[]

ele=1

def push(ele):

    global cost,pe

    s.append(ele)

    #cost=cost+1

    #acost=acost+2

    pe=pe+1

    p.append(pe)

def pop():

    global cost,pe

    if len(s)!=0:

        print(s[len(s)-1])

        del s[len(s)-1]

        #cost=cost+1

        #acost=acost+0

        pe=pe-1

        p.append(pe)

    else:

        print("underflow")

def multipop(no):

    global cost

    n=min(no,len(s))

    for i in range(0,n):

        pop()

#k=int(input("enter no of operations : "))

k=5

operations=numpy.random.uniform(0,3,k)

o=operations.astype(int)

for i in o:

    if i==0:

        push(ele)

    elif i==1:

        pop()

    elif i==2:

        x=r.randrange(1,10)

        print("muptipopin ",x," times")

        multipop(x)

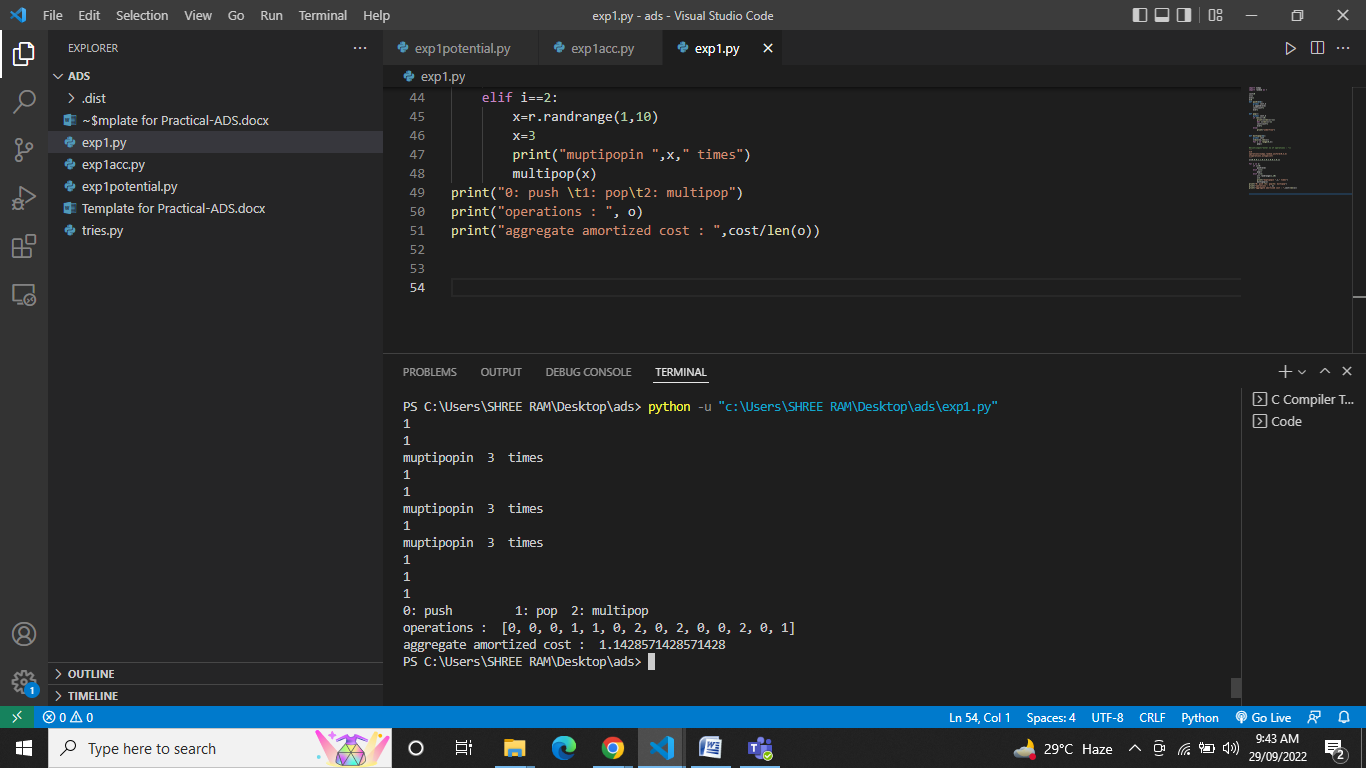
print("0: push \t1: pop\t2: multipop")

print("operations : ", o)

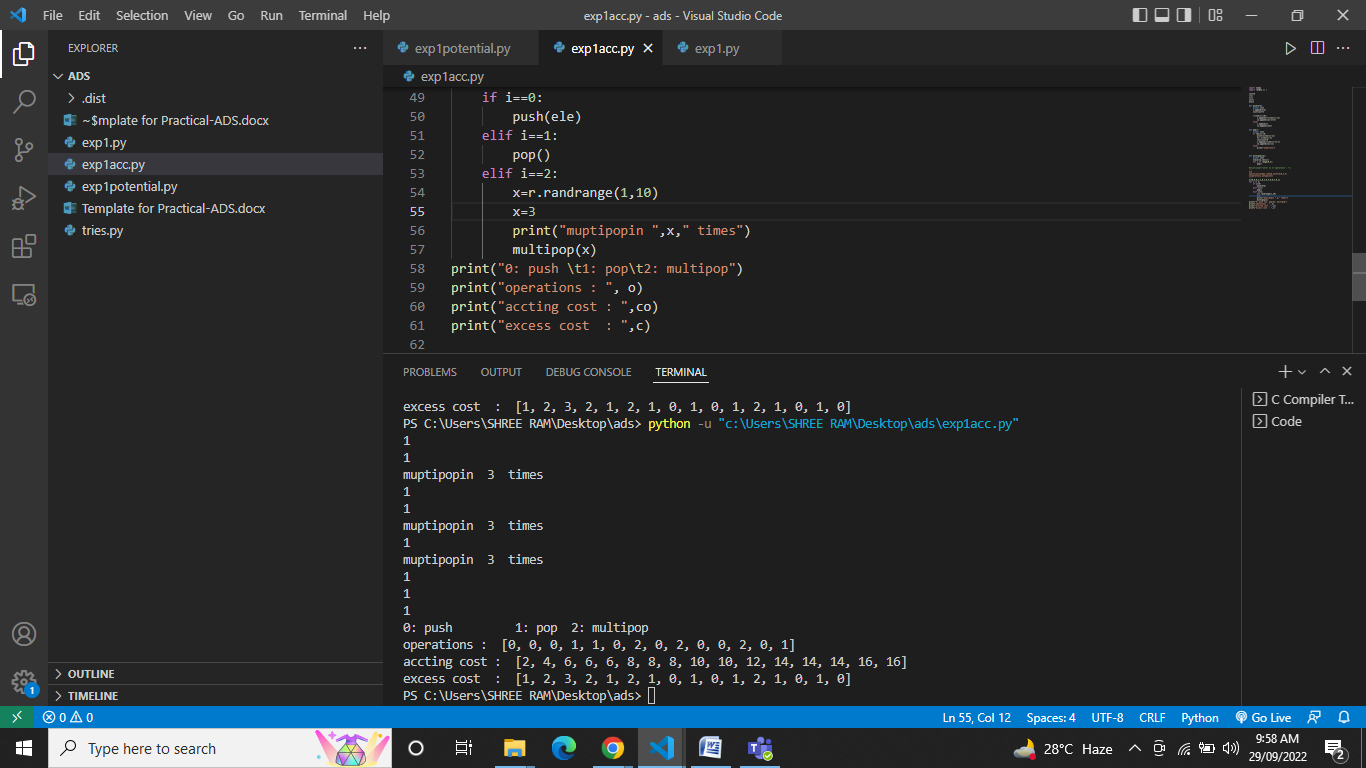
print("potential energy : ",p)

**Output:**

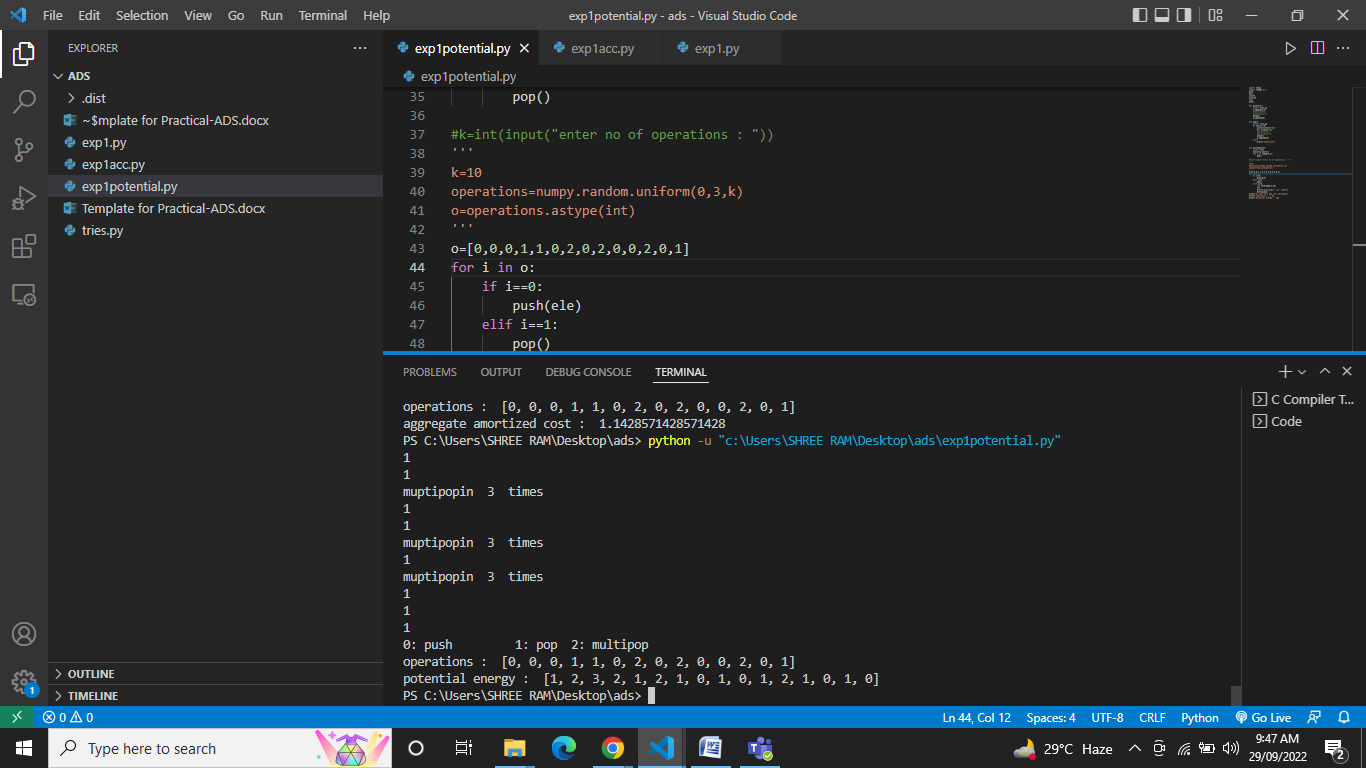
Aggregate method:



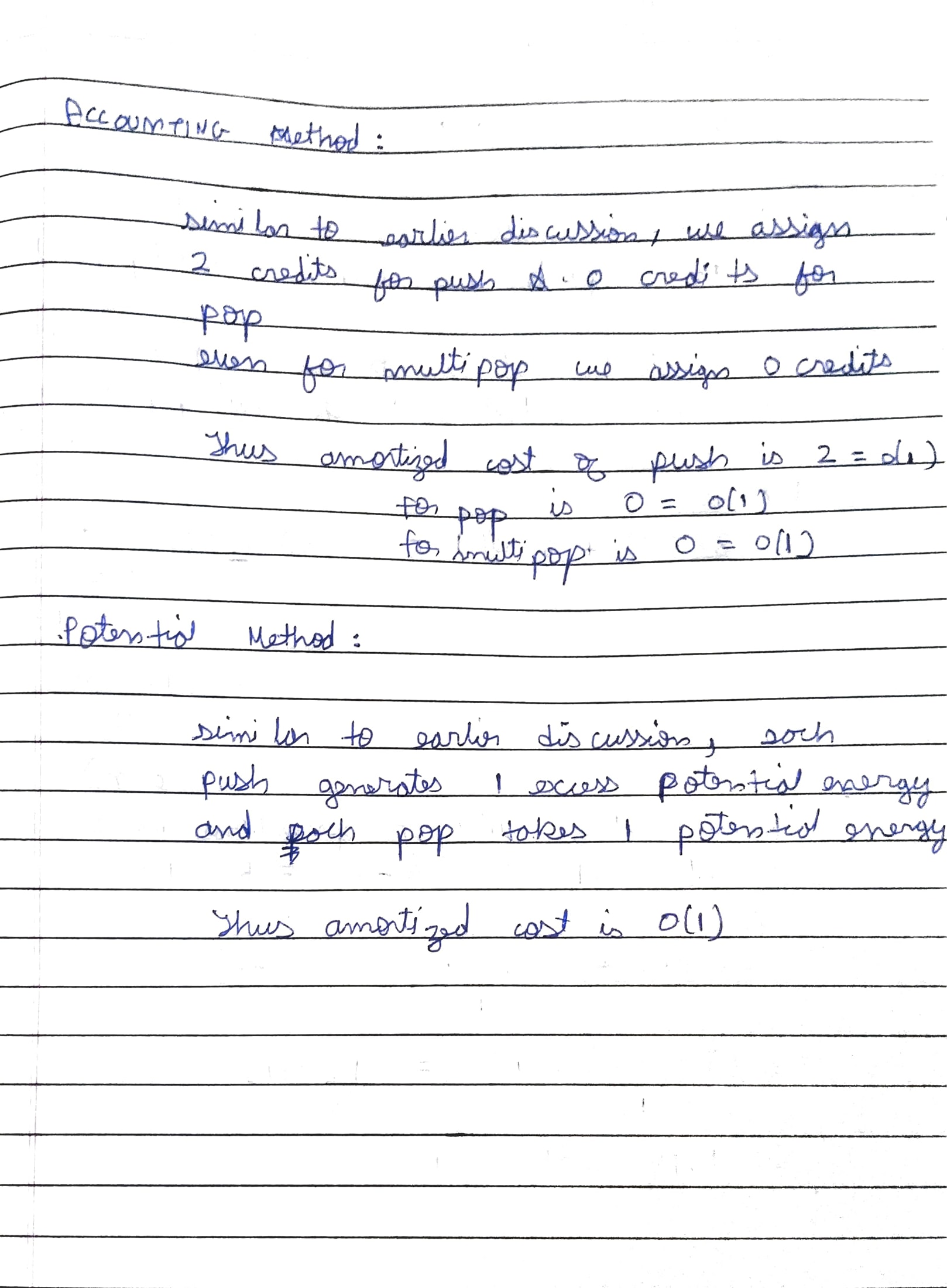
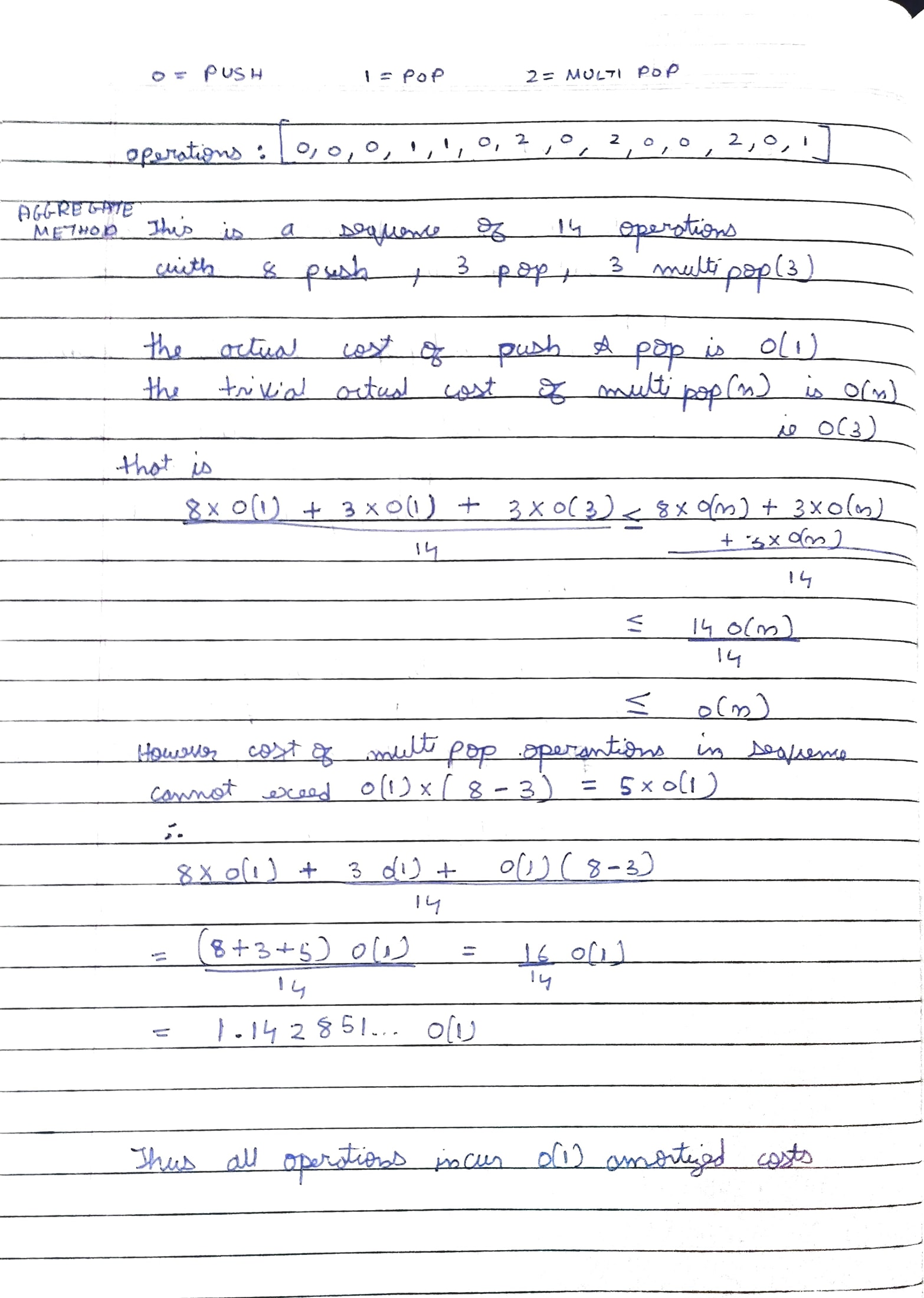
Accounting method:



Potential method:



**Analysis:**

****

**Conclusion:**

## Thus we have performed amortized analysis for stack operations of push, pop and multipop using aggregate method and observed similar behavior in accounting and potential method.

**References:**

<https://brilliant.org/wiki/amortized-analysis/#:~:text=There%20are%20three%20main%20types,method%2C%20and%20the%20potential%20method.>