Jigar Siddhpura
60004210155
C22

Experiment 1c — Amortized Analysis
(Potential method)

Aim: To perform amortized analysis with potential method

Theory: It is a method for analyzing the time complexity of algorithms over a sequence of operations. It provides a way to average out the cost of expensive operations by considering the total cost spread over all operations, thereby giving a more accurate picture of the overall performance.

Potential method is a common technique using in amortized analysis. It involves defining a potential function that represents 'unused' or 'saved' resources at each step of the algorithm. Potential function should be non-negative at ~~each step of the algorithm~~ & should reflect the cost savings achieved by the algo.

For example: Consider a data structure like a dynamic ~~array~~ array that doubles its size whenever it runs out of space. Naively, we think each insertion takes $O(n)$ time because of the occassional need to resize the array. However, with potential function, we can show that each insertion actually takes $O(1)$ time on average.

We can define the potential function
$\phi(i) = 2^{\log(i)} - 1$
where 'i' is the current size of the array.
Initially at $i = 1$, $\phi(1) = 0$

Everytime we double the size of the array, the potential increases by a factor of 2, compensating for the cost of resizing. The amortized cost of each iteration is then the actual cost plus the change in potential i.e. $O(1)$.

Conclusion:

We performed amortized analysis using potential function which provides a way to analyze the average cost of operations in a sequence, taking into consideration the fluctuation in individual operations.

CODE :

```python
doubling_costs = []
current_length = 1
potential = []

for i in range(1, 11):
    if current_length < i:
        current_length *= 2
        doubling_costs.append(i-1)
    else:
        doubling_costs.append(0)
    potential.append(2*i- current_length)

total_cost = [x+1 for x in doubling_costs]
print('Doubling Cost\t Iteration\t Total Cost\t Potential\tAmortized Cost')

print(f'{doubling_costs[0]}\t\t {1}\t\t {total_cost[0]}\t\t{potential[0]}\t\t {total_cost[0] + potential[0]}')

for j in range(1, 10):
    amortized_cost = total_cost[j] + potential[j]- potential[j-1]
    print(f'{doubling_costs[j]}\t\t {1}\t\t {total_cost[j]}\t\t{potential[j]}\t\t {amortized_cost}')
```

OUTPUT :

```
PS D:\SEM-6\AA\EXPERIMENTS> python -u "d:\SEM-6\AA\EXPERIMENTS\Amortized_potential.py"
Doubling Cost    Iteration       Total Cost      Potential       Amortized Cost
0                1               1               1               2
1                1               2               2               3
2                1               3               2               3
0                1               1               4               3
4                1               5               2               3
0                1               1               4               3
0                1               1               6               3
0                1               1               8               3
8                1               9               2               3
0                1               1               4               3
PS D:\SEM-6\AA\EXPERIMENTS>
```