Assignment No 2

* ~~Title - HoP~~

Q1 What is potential function method in Amortized Analysis ? Find the amortized cost of PUSH, POP & MULTIPOP stack operations using the same method.

In Amortized analysis the potential function method is used to distribute total cost of a series of operations over each individual operation. The idea is to define a potential function Φ that represents amount of stored energy in the data structure this potential can increase or decrease with ~~sa~~ each operation, allowing some operations to be charged more than their actual cost while other are charged less.

Stack operations PUSH, POP, and MULTIPOP

let potential function $\Phi(S)$ be defined as the number of elements in the stack after an operation S.

PUSH operation

Actual cost 1

. Potential fun change - increment by 1.
Amotized Cost
$AC = \text{Actual Cost} + (\Phi_{after} - \Phi_{before}) = 1 (n+1-n) = 2$

* POP Operation.
Actual Cost - 1
Potential function change - increment by 1
Amotized Cost -
$AC = 1 + (\Phi_{after} - \Phi_{before}) = 1 + (n+1-n) = 0$

* MULTIPOP Operation.
Actual Cost : min (k, n) where k is number no of element to pop.
Potential function change i.e. k elements are popped the no of element decreased by k, so Φ decrement by k
Amotized Cost -
$AC = k + (\Phi_{after} - \Phi_{before}) = k + (n-k-n) = 0$

Q2 Psedo Code for Naive String Matching Algorithm.

The naive string matching algorithm checks for a pattern P of length m in text T of length n by sliding pattern over the text one position at a time and checking for a match

Algorithm -

```
NaivStringMatching (T, P)
    n = length (T)
    m = length (P)
    for s = 0 to (n-m)
        if T [s+1 : s+m] == P [1 : m+1]
            print.
```

Time Complexity - $O((n-m+1) * m)$

The algorithm slides pattern across text and compares every substring of length m

Q3 Write and Explain code of multithreaded merge sort algorithm

In multithreded Merge sort, the array is divided into subarrays recursively, and the sorting is doing in parallel using multithreds. After sorting the arrays are merged back together

~~Mero~~ Algorithm -

```
Mergesort (A, P, r)
  if P < r
      q = (p + r)
      Parallel :
          Mergesort (A, P, q)
```

MergeSort (A, q+1, r)
Merge (A, p, q, r)

```
Merge (A, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    L = new Array of size n1 + 1
    R = new Array of size n2 + 1
    for i = 0 to n1 - 1
        L[i] = A[p + i]
    for j = 0 to n2 - 1
        R[j] = A[q + j + 1]
    L[n1] = infinity
    R[n2] = infinity
    i = 0
    j = 0
    for k = p to r:
        if L[i] <= R[j]
            A[K] = L[i]
            i = i + 1
        else
            A[K] = R[j]
            j = j + 1
```

- It recursively divides the array in two haves using threads
- Merge function then merges two sorted halves
- Time Complexity O(n log n)