Venkata Krishna
Chaitanya
C012366591

Assignment - 8

Problem :-

Consider a set of 'n' numbers stored in an array A[1, 2, ... n]. You are going to store the values A[j] in another set B such that for any values A[i] with i<j, we have A[i]<A[j]. An example is as below:
A = [3, 5, 2, 8, 7] Then B = [3, 5, 8].
Design an algorithm to find out B in linear time. Use amortized analysis to show your algorithm's running time.

Solution :-

The intent of the problem is to traverse through the list, if and the element of the list in that index or iteration is higher or larger than all the elements to its left, then we PUSH it into a new list. And we return that list.

So, the number of iterations would be the size of the list/array ⇒ n elements

→ Assign a dummy variable for comparision & updation
→ Iterate/traverse through the list
→ Now, compare the current iteration element with the dummy variable created.

if iteration_element > dummy_variable
Update dummy_variable & PUSH to new list
else :
continue for further iterations

```
def ReturnAlgo(list):
    dummy_variable = 0 ; new_list = []
    i = 0
    While [i < len(list)):
        if [list[i] > dummy_variable):
            ~~new_list.append(list[i])~~
        ~~i=i+1~~
            dummy_variable = list[i]
            new_list.append(dummy_variable)
        i = i+1
    return new_list
```

As, we are iterating through the list only once and pushing the required elements into a new_list linearly. The running time would be 'O(n)'.

Now, using Amortized Analysis,
we can average the running times of operations in a sequence over that sequence. which complements the worst-case & average-case analysis.

Three general methods performing amortized analysis :-

-> Aggregate Analysis
-> Accounting Method
-> Potential Method.

Aggregate Analysis :-

It determines the upper bound $T(n)$ on the total cost of a sequence of n operations, then calculates the amortized cost to be $T(n)/n$.

So, in my case, the amortized cost would be.

for each iteration:

cost (comparing list[i] & dummy_variable)

+

cost (updating dummy_variable)

+

cost (appending dummy_variable to new_list)

+

cost (incrementing i with +1)

Let these costs sum upto $T(n)$ in each iteration

And the total cost for all the iterations would be number of elements $\times T(n) \Rightarrow n \times T(n)$

And as we divide by the $\Rightarrow n$ size of the list

Amortized cost would be $\frac{n \times T(n)}{n} \Rightarrow T(n)$

Potential Approach :-

Modified costs for each operation as follows: for an operation

$$\delta_{ante} \xrightarrow{c} \delta_{post}$$

with true cost c, define the modified cost to be

$$c' = c + \phi(\delta_{post}) - \phi(\delta_{ante})$$

These modified costs are valid amortized costs.

$$c' = c + \Delta\phi$$

So, for the three operations

→ Comparision operation
→ Updating dummy variable
→ Appending to new list

the $c' = c_{comp} + c_{update} + c_{append} + \Delta\phi_{comp} + \Delta\phi_{update} + \Delta\phi_{append} + c_i + \Delta\phi_i$

Here, for all the ~~three~~ four operations the relative modifying cost would be negligible.

i.e.,

$$\Delta\phi_i = \Delta\phi_{comp} = \Delta\phi_{update} = \Delta\phi_{append}$$

Therefore, the

amortized cost = c'

$= c_{comp} + c_{update} + c_{append} + c_i$

Where,

c_{comp} = cost (comparing list[i] & dummy_variable)

c_{update} = cost (updating dummy_variable)

c_{append} = cost (appending dummy_variable to new_list).

c_i = cost (incrementing 'i' by +1)