Sk10101_hw3.pdf

Saturday, October 15, 2022 8:30 PM

Q1) 1.) Worst Case : $O(n^2)$

The while loop, in its worst case, is run n times & inserting at the beginning of the list each time runs in $O(n)$ time complexity, since it shifts n elements to the right each time. Hence, $n * n = n^2$

2.) Worst case : $O(n)$

The while loop, in its worst case, is run n times & each time append is going to take a constant time $O(1)$ to add a value to the end of list. Hence, $n * 1 = n$

Q2 c) Extra-Credit Question

1. When there are n append operations in a list, the cost of i^{th} operation can be defined as follows:

So, when the list is full, i.e exact power of 2^{th} operation (1^{st} append, 2^{nd} append, 4^{th} append, 8^{th} append...) we copy $(i-1)$ elements to the double-sized temporary array. This takes i cost of insertion operation

$\therefore$ Sum $= 1+2+1+4+1+1+1+8+1+1+1+1..$

$\therefore$ overall time complexity: $(1+2+4+8+\ldots+n/2+n) + (1+1+1+1+\ldots)$

$$= n + \frac{1(1-2^{\log_2 n})}{1-2} + (1+1+1+1+1+1\ldots < n)$$

$= 2n + n = 3n$

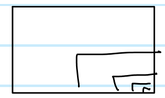
Hence, $O(3n) = O(n)$ — a

- For popping let's consider the sequence of cost per operation:

$$(1+1+1+1+1\ldots)\frac{3n}{4} + \frac{n}{4} + (1+1+1+1+1+1+\ldots)\frac{3n}{16} + \frac{n}{16}\ldots$$

$\therefore$ Overall, time complexity $= \left(\frac{n}{4} + \frac{n}{16} + \frac{n}{64} + \ldots + 1\right) + (1+1+1+1\ldots < n)$

$$= \frac{n}{2} + n = \frac{3n}{2}$$



Hence, $O(\frac{3n}{2}) = O(n)$ — (b)

(a) + (b) $= O(n+n) = O(2n)$

Hence, the series of $2n$ operations takes $O(n)$ time overall.

2. For pop/append, let's consider the sequence for cost of operation:

$$(1+1+1+1+1+1+1+1\ldots)\frac{n}{2} + \frac{n}{2} + (1+1+1+1+1+1+1)\frac{n}{4} + \frac{n}{4} + (1+1+\ldots)\frac{n}{8} +$$

$\therefore$ Overall, time complexity $= \left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + 1\right) + \left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots < n\right)$

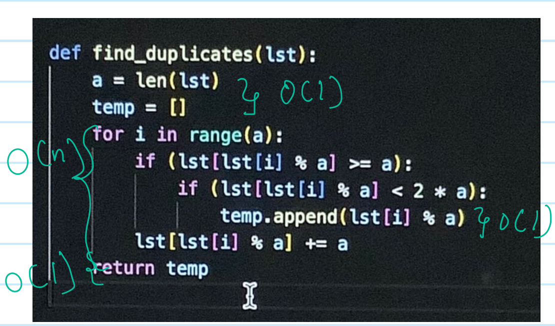
$$= n + n^2$$

Hence, for $n+n^2$, time complexity $= \Omega(n^2)$ as $n^2+n^2 = 2n^2$ and $\Omega(2n^2) = \Omega(n^2)$

Q3) b) The for loop runs for n times in its worst case.

The append method runs in $O(1)$ time each time (total times $< n$) & if, += operators take constant $O(1)$ time complexity each time.

return takes $O(1)$ complexity.



```
def find_duplicates(lst):
    a = len(lst)
    temp = []
    for i in range(a):
        if (lst[lst[i] % a] >= a):
            if (lst[lst[i] % a] < 2 * a):
                temp.append(lst[i] % a)
            lst[lst[i] % a] += a
    return temp
```


$\boxed{\therefore \text{Worst Case}: O(n)}$

Q4) (a) since remove works in $O(n)$ complexity, in its worst case, that is, & while loop runs n times in its worst case, when the value to be removed is at the last index.

```
def remove_all(lst, value):
    end = False }O(1)
    while(end == False):
        try:
O(n²)      O(n){lst.remove(value)
        except ValueError:
           O(1){end = True
```

$n*n = n^2$

$\boxed{\therefore \text{ Worst Case } = O(n^2)}$

(c) The for loop, in its worst case, runs n times.

constant time complexity, $O(1) \Rightarrow$ =, +=, swapping, if, else, pop( ), <=, ValueError.

as the while loop runs in < n times, it's complexity is also $O(1)$.

as a result, worst run-time complexity = $O(n)$

```
def remove_all(lst, value):

    last_val=0 }O(1)
O(n){for i in range(len(lst)):
        if lst[i]!=value:
      O(1){lst[i], lst[last_val] = lst[last_val], lst[i]
            last_val+=1
O(1){if last_val==0:
        raise ValueError('value not present in list')
    else:
O(1){   while last_val<=len(lst)+1:
            lst.pop() }O(1)
            last_val+=1
```

$\boxed{\therefore \text{ Worst Case: } O(n)}$