DSA hand-in 1

David Sermoneta

March 30, 2024

Exercise 1

```
function multiplier(a, b):
  1 y = 0
  2 atimesb = 0
  3 WHILE y < b:
  4     x = 0
  5     WHILE x < a:
  6         atimesb = inc(atimesb)
  7         x = inc(x)
  8     y = inc(y)
  9 RETURN atimesb
```

We need three things, a counter to keep track of how many times we've added 1 to `a` (should be `a` times), and a counter to keep track of how many times to add `a` to itself (should be `b` times).

The first counter that achieves this is the counter `y`, as we run the first while loop, we will add `1` to `y` until it stops being smaller than `b`, then we'll have known that we've added `a` to itself exactly `b` times.

Exercise 2

Part a)

Let $A = (\texttt{apppe, apple, appl, apap, appap})$. After each iteration of the `FOR` loop we get

```
j = 0: (apppe, apple, appl, apap, appap)
    1: (apple, apppe, appl, apap, appap)
    2: (appl, apple, apppe, apap, appap)
    3: (apap, appl, apple, apppe, appap)
    4: (apap, appap, appl, apple, apppe)
```

Part b)

To get the worst case runtime of insertion sort, we want the `WHILE` loop to execute as much as possible, (as all the other steps have fixed runtimes after a choice of the input size). To get that, we want `key` $\prec$ `A[i]` for all `i` $\geqslant$ `0`. This is ensured to be the case if we let A be in reversed order. That is A' should be

```
A' = (apppe, apple, appl, appap, apap).
```

Exercise 3

Part a)

We know that $T_B(n)$ is in $O(T_A(n))$ if and only if $T_A(n)$ is in $\Omega(T_B(n))$, that is, if for some positive constants c, n_0,

$$T_B(n) \leqslant cT_A(n), \text{ for all } n \geqslant n_0.$$

To solve this, we write out all terms and rewrite a bit to get

$$\frac{5}{2}n^2 < \frac{cn^2}{10}\log_{10}(n) \iff 25n^2 < cn^2\log_{10}(n).$$

Here it is easy to see that $c = 1$ and $n_0 = 10^{25}$ is a choice of constants that help us satisfy the inequality (since $\log_{10}(10^{25}) = 25$). Therefore, $T_B(n) \in O(T_A(n))$, and $T_A(n) \in \Omega(T_B(n)$.

To be done, we just need to show that $T_A(n)$ is not in $O(T_B(n))$. We do so via contradiction. Suppose it was true, then for some constants $c, n_0 > 0$, we have that for all $n \geqslant n_0$,

$$\frac{n^2}{10}\log_{10}(n) \leqslant \frac{5c}{2}n^2 \iff \log_{10}(n) \leqslant 25c.$$

However, this is clearly not possible, as for $n \geqslant 10^{25c}$, the inequality does not hold. Therefore we are done, T_B is the better algorithm for general n.

Part b)

First we note that both runtimes of each algorithm are monotone increasing. From this it becomes clear that for $n \leqslant 10^9$, algorithm A is the recommended one, since

$$T_A(10^9) = \frac{10^{18}}{10}\log_{10}(10^9) = 10^{17} \cdot 9 < 10^{17} \cdot 25 = \frac{5 \cdot 10^{18}}{2} = T_B(10^9),$$

and we are done.

Exercise 4

Part a)

For `function(100, 0)`, we get the following values of `n` and `N`:

```
n: 100 N: 0
n: 33  N: 1
n: 11  N: 2
n: 3   N: 3
n: 1   N: 4 .
```

Part b)

We note that the runtime of `function(n,N` only depends on the input of `n`. Therefore, we can denote the runtime by $T(n)$, and we get that the check and increment operations only take $\Theta(1)$ time, and so we get that

$$T(n) = 2\Theta(1) + T\left(\left\lfloor \frac{n}{3} \right\rfloor\right).$$

Iterating this a few times, we get the following pattern

$$T(n) = 2N'\Theta(1) + T\left(\left\lfloor \frac{n}{3^{N'}} \right\rfloor\right).$$

It is clear that the algorithm will terminate at $n = 1$, so we want to find for which N', $\frac{n}{3^{N'}} \leqslant 1$, which gives us that $\log_3(n) \leqslant N'$. Putting $N' = \log_3(n)$ in our formula gives us

$$T(n) = 2\log_3(n)\Theta(1) + T(1) \in O(\log_3(n)).$$

And we are done, we have shown that the best case scenario for `function(n,N)` is $O(\log_3(n))$.

Exercise 5

Let $S = \{A_0, A_1, \ldots, A_9\}$ correspond to the 10 stacks of items, such that for each A_i,

$$A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,10}\}.$$

Keeping to the constraints of our tools, we want to minimize the amount of times we need to weigh items. So optimally our algorithm would only need to do a single weighing. And this is certainly possible, if we take one item from A_0, 2 items from A_1 and so on until we get to 10 items from A_9, we will get the following equation:

$$\text{Balance} = \sum_{k=1}^{10}(k) + (i+1),$$

where $(i+1)$ stands for the extra grams depending on which stack A_i has the counterfeit items. Written in pseudo-code, we get

```
function find_counterfeit(S):
    1 items = []
    2 FOR (i = 0 to 9) DO:
    3    A = S[i]
    4    items.add(A[0,i]
    5 counterfeit weight = weigh(items) - 55
    6 print("Stack number",{counterfeit weight - 1},contains
    7 all counterfeit items."
    8 RETURN (counterfeit weight - 1)
```

So the function returns the the index of the stack A_i that has all the counterfeit items, and we are done.