Lecture 03 Exercise Solutions

Mark Eramian

Exercise 1

a) When offset 0 of the array a is equal to s.

b) When the string s does not appear at all in a.

c) Statement Counting Approach (Best case):

   - Loop condition is never true.
   - Number of loop iterations: 0
   - +1 for the loop condition being false.
   - +3 statements outside the loop
   - Total: $T^{B}_{contains}(n) = 4$

   Statement Counting Approach (Worst case):

   - Let n be the length of of the array a.
   - Number of loop iterations: n
   - Statements per loop iteration: 2
   - Total statements for the loop: $2n + 1$ (+1 for when the loop condition is false).
   - +3 statements outside the loop
   - Total: $T^{W}_{contains}(n) = 2n + 4$

Exercise 2

Here is the analysis of insertion sort. We begin with the inner loop.

- Number of statements per inner loop iteration: 3
- Number of iterations of inner loop (worst case): i
- Plus one statement when the loop condition is false.
- Total: 3i + 1;

  Now the outer loop:

- Number of statements per outer loop iteration: 5 + cost of inner loop = $5 + 3i + 1$

- Since cost of inner loop depends on which iteration of the outer loop we are on, we must use summation notation.
- Cost for all iterations of outer loop: $\sum_{i=1}^{n-1}(5+3i+1)$
- Plus one more for when the outer loop being false.
- Total: $\sum_{i=1}^{n-1}(5+3i+1)+1$

Now for the overall method:

- Total cost of outer loop, plus one initialization statement: $\sum_{i=1}^{n-1}(5+3i+1)+2$

Now it's just a matter of simplification:

$$\begin{aligned}
\sum_{i=1}^{n-1}(5+3i+1)+2 &= 2+\sum_{i=1}^{n-1}(6+3i) \\
&= 2+\sum_{i=1}^{n-1}6+\sum_{i=1}^{n-1}3i \\
&= 2+6(n-1)+3\sum_{i=1}^{n-1}i \\
&= 2+6n-6+3\frac{(n-1)(n)}{2} \\
&= 2+6n-6+\frac{3}{2}n^2-\frac{3}{2}n \\
&= 1.5n^2+4.5n-4
\end{aligned}$$

Exercise 3

Analysis of `contains()` method from Exercise 1 using active operation approach:

The active operation is the loop condition which, in the worst case, will execute $n+1$ times, where n is the length of the array `a`. No other statement executes more often.

Analysis of `insertionSort()` method from Exercise 2 using active operation approach:

The active operation for the insertion sort method is the condition of the inner while loop: `j >= 0 && temp.compareTo(a[j]) < 0`. How many times does it execute? Each time the inner loop is executed by the outer loop, the active operation is executed $i+1$ times in the worst case. To analyze the outer loop we must use summation notation because the number of active operations in each iteration of the outer loop depends on `which` iteration is executing. The number of active operations in the outer loop is:

$$\sum_{i=1}^{n-1}(i+1)$$

And now we just need to simplify, and substitute the closed-form of the summation

$$\begin{aligned}\sum_{i=1}^{n-1}(1+i) &= (n-1)+\frac{(n-1)(n)}{2} \\ &= n^2/2+n/2-1\end{aligned}$$

Exercise 4

- $n \in O(n)$
- $47n\log n + 10000n \in O(n\log n)$
- $100n + 500\log n + 1000 \in O(n)$
- $\log n + 100\sqrt{n} + 76 \in O(\sqrt{n})$
- $n^2 \in O(n^2)$
- $2^n + n + \log n \in O(2^n)$
- $5 \in O(1)$
- $T^B_{count}(n) = 3n + 4 \in O(n)$
- $T^W_{count}(n) = 4n + 4 \in O(n)$
- $T^B_{contains}(n) = 4 \in O(1)$
- $T^W_{contains}(n) = 3n + 4 \in O(n)$
- $T^W_{insort}(n) = 1.5n^2 + 4.5n - 4 \in O(n^2)$

Exercise 5

Analysis of method `s`:

- active operation: `x = x + k(i) * k(2n - i)`
- number of active operations executed: $n+1$
- cost of active operation: $2 \times O(\log m)$.
- total cost: $(n+1)\cdot 2\cdot O(\log m)) = O(n\log m)$.

Exercise 6

Statement Counting Approach

- How many times is the loop body executed?
  - Let r be the number of times the loop body is executed.
  - Consider the values of p when the loop condition is true:

    $$n, n/2, n/4, \ldots, n/2^{r-1}$$

    All these values (there are r of them) must be greater than 1, or the loop would stop.
  - This implies: $n/2^r \leq 1 < n/2^{r-1}$. Now we need to solve for r.
  - If the previous inequality is true, it means that $n \leq 2^r$ and $2^{r-1} < n$
  - Taking logarithm of each inequality, we get $\log n \leq r$ and $r - 1 < \log n$, or, equivalently:

    $$r - 1 < \log n \leq r$$

  - Since r must be a non-negative integer (it's the number of loop iterations), $r = \lceil \log n \rceil$ for $n \geq 1$.
- What is the cost of one loop iteration?
  - 3. The loop has 3 statements including the loop condition.
- Total for the loop: $3\lceil \log n \rceil + 1$ (multiply loop cost by number of iterations, plus 1 for when the loop condition is false)
- Total for the whole method: $\text{loop cost} + 3 = 3\lceil \log n \rceil + 1 + 3 = 3\lceil \log n \rceil + 4$ which is $O(\log n)$.

Active Operation Approach

- The active operation is: the loop condition (line 5)
- Number of executions of the active operation: $\lceil \log n \rceil$ (we showed this in the statement counting approach)
- Cost of active operation: 1
- Cost of all active operations: $1 \times \lceil \log n \rceil$ (cost of active operation multiplied by number of executions).
- Therefore the entire algorithm is $O(\log n)$.

Exercise 7

It is not immediately obvious which statement should be the active operation because there are three statements that call methods, all of different cost. So we consider all three possible active operations and their total cost. The first two are fairly straightforward:

- Active operation: `q()`
  - number of times executed: 1
  - cost per execution: $O(n \log m)$
  - total cost: $O(n \log m)$
- Active operation: `s()`
  - number of times executed: 1
  - cost per execution: $O(m \log m)$
  - total cost: $O(m \log m)$

The third one is trickier, because it's not immediately obvious how many times the loop containing the active operation executes:

- Active operation: `p()`
  - number of times executed: ??
  - cost per execution: $O(n^2)$
  - total cost: $?? \cdot O(n^2)$

In order to find out how many times the active operation `p()` is executed, we must determine how many times the loop body executes.

- Let $x =$ number of times loop in method `r` is executed.
- Consider the values of `i` in each loop iteration where the loop condition is ture (there are x of them):
  $1, 2, 4, 8, 16, \ldots, 2^{x-1}$.
- When loop stops, $i \geq m$. This implies that:

$$\begin{aligned} 2^{x-1} < &\quad m &\leq 2^x \\ x - 1 < &\quad \log m &\leq x \\ x &\quad = &\lceil \log m \rceil \end{aligned}$$

- This result follows because x must be a positive integer at least a big as $\log m$. $x = \lceil \log m \rceil$ is the only integer that causes $x - 1$ to be clearly less than $\log m$ ($x = \lceil \log m \rceil$ cannot exceed $\log m$ by more than 1, therefore $x - 1 < \log m$) and causes x to be greater or equal to $\log m$, and thus the only integer to satisfy both inequalities.

Now we can plug our value for x back into the original analysis:

- Active operation: `p()`
  - number of times executed: $\log m$
  - cost per execution: $O(n^2)$
  - total cost: $(\log m) \cdot O(n^2)$ or $O(n^2 \log m)$

So now we have three active operations with total costs $O(n \log m)$, $O(m \log m)$, and $O(n^2 \log m)$, respectively. Our final answer for the time complexity of method `r` should be whichever of these grows the fastest. But which grows fastest? Clearly $O(n \log m)$ can be eliminated since $O(n^2 \log m)$ definitely grows faster. But which of $O(n^2 \log m)$ and $O(m \log m)$ grows faster? The answer is: **we don't know!**. There is no way of knowing because n and m are independent. All we can do is write our final answer as: $O(n^2 \log m) + O(m \log m)$ because we don't know if one of these terms grows more quickly than the other – we do not know the relationship (if any) between m and n. The best we can do to simplify this further is to use our rules for combining Big-Oh expressions:

$$O(n^2 \log m) + O(m \log m) = O(\max(n^2, m) \cdot \log m)$$

Exercise 8

Time Complexity of `LinkedList` Class Methods

All of the methods in `LinkedLIst` are $O(1)$ because they contain no loops, and no method calls that are slower than $O(1)$. Some of these call methods form the `LinkedNode` class, but these are also $O(1)$ because they are just accessor and mutator (a.k.a. getters and setter) methods.

Time Complexity of `ArrayList` Class Methods

All of the methods in `ArrayList` are $O(1)$ except for `insertFirst` and `deleteFirst`.

The loop in `insertFirst` runs `listTail` times. Since the value of `listTail` is equal to the number of items in the list, n, the method is $O(n)$. This makes sense because we have to move each of the existing items in the list to make room for the new item.

Similarly, the loop in `deleteFirst` runs `listTail` $- 1$ times. Thus, `deleteFirst` is $O(n)$ where n is the number of items in the list.