Practice Problems 1

Vancouver Summer Program 2022 – Algorithms – UBC

Notation.

- $\mathbb{N} = \{1, 2, \dots\} \subset \{0, 1, 2, \dots\} = \mathbb{Z}_+$
- $\mathbb{R}_+ = [0, \infty)$
- $[n] = \{1, \dots, n\}$.

Practice Problems

1. (worked example) $3n \in O(n^2)$.

   All we need to do is find c and N as in the definition. For $1 \leq n$, $3n \leq 3n^2$ (multiply both sides of $1 \leq n$ by $3n$), so $c = 3$ and $N = 1$ works. These choices are not unique. For instance, $c = 1$ and $N = 3$ is another.

2. (worked example) $\log_{12} n^{20} \in O\big(\log_2(n)\big)$.

   $\log_{12} n^{20} = 20 \log_{12} n = 20 \frac{\log_2 n}{\log_2 12}$, so $c = \frac{20}{\log_2 12}$ and $N = 1$.

   Moral: The base of logarithm does not matter in asymptotic running time. That's why when $\log$ is inside $O(\cdot)$, we do not usually write the base and instead we write $O(\log n)$.

3. (worked example [partially]). $n \log n \in \Theta\big(\log(n!)\big)$.

   We have to show that both $n \log n \in O\big(\log(n!)\big)$ and $\log(n!) \in O(n \log n)$. We will show the second; the first is an exercise (and a good one!)

   We have

   $$\begin{aligned} \log(n!) = \log\big(n(n-1)\cdots 1\big) &= \log(n) + \log(n-1) + \cdots + \log(1) \\ &\leq \underbrace{\log(n) + \log(n) + \cdots + \log(n)}_{n \text{ terms}} \\ &= n \log n. \end{aligned}$$

   So $c = N = 1$ works.

4. (worked example) $f(n) \notin o\big(f(n)\big)$, but $f(n) \in O\big(f(n)\big)$

   $\lim_n \frac{f(n)}{f(n)} = 1 \neq 0$, and since the limit exists, $\limsup_n \frac{f(n)}{f(n)} = \lim_n \frac{f(n)}{f(n)} = 1 < \infty$.

5. (worked example) $\log n \in o(n^\varepsilon)$ for any $\varepsilon > 0$.

   Choose $0 < \delta < \varepsilon$. Using the fact that $\log x \leq x$ for all $x \geq 1$ (prove this using calculus), we have $\log n^\delta \leq n^\delta$, and therefore $\log n \leq n^\delta/\delta$. Then

$$0 \leq \frac{\log n}{n^\varepsilon} \leq \frac{1}{\delta n^{\varepsilon-\delta}} \xrightarrow{n\to\infty} 0.$$

6. Show that $8n^4 + 13n^2 + 9 \in O(n^4)$. [Hint: one choice is $c = 30$ and $N = 1$]

7. Show that $3^n \notin O(2^n)$ [Hint: use the limit definition or assume otherwise and arrive at a contradiction.] Conclude that for any $c > 0$, $2^{n+c} \in O(2^n)$, and if $c > 1$, $2^{cn} \notin O(2^n)$.

   Moral: Whereas the base of logarithms does not matter in asymptotic complexity, the base of exponentials does!

8. If $f \in o(g)$ then $f \in O(g)$. The converse is not true.

9. Give an example of a function in $o(1)$. Do you think there is an algorithm whose running time is $o(1)$?

10. We discussed that the resources used by algorithms are typically time, space, and random bits. Argue that the space required by an algorithm is a *lower* bound on the time it takes to run.

11. Consider the following procedure

```
Foo(N)
k ← 0
for i ← 1 to N
    for j ← 1 to N
        k ← i + j
return k
```

    Argue that Foo runs in *exponential* time.

12. (Binary Search) Consider the following algorithm that searches for an element (needle) in an *ordered* list. Assume that list$[1] \leq \cdots \leq$ list$[n]$, where n is the number of elements in the list. L denotes Left, R denotes Right, and m denotes middle.

```
BinarySearch(needle, list[1 ... n])
L ← 1, R ← n
while L ≤ R
    m = ⌊(L + R)/2⌋
    if needle < list[m]
        R ← m − 1        ⟪needle is in the first half⟫
    if list[m] < needle
        L ← m + 1
    if list[m] = needle
        return m
return −1 (not found)
```

(a) Argue that if needle is in list, the BINARYSEARCH successfully finds it (correctness).

It will be easier to reason about the following version, where an answer is returned after the loop terminates entirely (convince yourself that the two versions are equivalent, except for a constant difference in running time).

```
BinarySearch(needle, list[1 ... n])
L ← 1, R ← n
while L < R
    m = ⌊(L + R)/2⌋
    if needle ≤ list[m]        ⟪needle is in the first half⟫
        R ← m
    if list[m] < needle
        L ← m + 1
if list[m] = needle
    return m
else
    return −1 (not found)
```

One approach is to identify a **loop invariant**, a condition that holds everywhere throughout the execution of the algorithm. The invariant is some kind of an assertion that, if preserved by the algorithm, can be used to establish the correctness of the algorithm. A loop invariant for BINARYSEARCH would be

Invariant: *if the value needle occurs in list (i.e., list*$[1 \dots n]$*), then it lies in list*$[L \dots R]$.

Let us call the follows statement the **precondition**: list is sorted and is of length at least 1, indexed from 1 to n. We will assume that this precondition always holds prior to the execution of the algorithm.

For correctness, what would really want to show is that the following statement holds upon execution of the algorithm, which we call the **postcondition:** Return an integer $m \in [n]$ such that list$[m]$ = needle if such m exists, and -1 otherwise.

To that end, we will show that the loop invariant holds at every iteration of the algorithm.

Lemma 1 *Suppose the precondition holds at the beginning of the algorithm. For each i, if the loop is executed i times, then:*

(i) $0 \leq L_i \leq R_i \leq n$,

(ii) if needle is in list, then needle is in list$[L_i \dots R_i]$.

Our predicate is:

$$P(i) : \text{if the loop is executed } i \text{ times, then (i) and (ii) hold.},$$

and we are trying to prove:

$$\text{Precondition} \Rightarrow \forall i \in \mathbb{Z}_+, P(i).$$

The proof of this Lemma is by mathematical induction with Induction Hypothesis $P(i)$. The base case is when $i = 0$ iterations are executed, which is trivial. In carrying out the

inductive step, one assumes that the loop has executed for $i \in \mathbb{N}$ steps—so $P(i)$ is true—and shows that if the algorithm executes for $i+1$ steps, then $P(i+1)$ is true. The details of the proof are left as an exercise to the reader.

How does proving the Lemma imply the correctness of BINARYSEARCH? If the precondition of BINARYSEARCH holds and it *terminates*, then the postcondition holds after execution, as follows. Say the loop exits after i iterations. By the Lemma, we know that $P(i)$ is true; i.e., (i) and (ii) hold. By the while loop condition, $L_i \geq R_i$, and since (i) of the invariant is maintained, $R_i \geq L_i$, so $L_i \geq R_i \geq L_i$, i.e., $R_i = L_i$, from which it follows that $m_i = R_i = L_i$. Furthermore, from (i), $0 \leq m_i \leq n$.

- **Case 1:** needle is in list; i.e., $\exists m \in [n]$ for which list$[m]$ = needle. By part (ii), needle is in list$[L_i \ldots R_i]$. Since $L_i = R_i$, this is just a one element list, and thus needle = list$[m_i]$, and BINARYSEARCH correctly returns m_i.
- **Case 2:** needle is not in list; i.e., $\forall m \in [n]$, list$[m] \neq$ needle. So list$[m_i] \neq$ needle, and the algorithm returns -1.

Termination. How do you show that this algorithm actually terminates in finite time? Observe that the sublist that the algorithm considers at every iteration is strictly smaller than the one in the previous iteration. That is, for every iteration $i \in \mathbb{N}$, $R_{i-1} - L_{i-1} > R_i - L_i$ [this can be shown formally easily]. Using the fact that any strictly decreasing sequence of nonnegative must be finite, and observing that $R_0 - L_0, R_1 - L_1, \ldots$ is a decreasing sequence of distinct nonnegative integers, it follows that the algorithm terminates in finite time.

(b) Show that the running time of BINARYSEARCH is $O(\log n)$. Note that in each iteration, BINARYSEARCH *halves* the sublist that it considers at every step. When is $L > R$? What is the minimum number of iterations (halving steps) such that the latter condition holds?