Learning Goals

- Review Big-Oh and learn big/small omega/theta notations
- Discuss running time analysis of algorithms

Best, Average, Worst Case Run Time Analysis

When analyzing algorithms, we are often interested in analyzing the best, average, and worst cases of running time.

Typically, best case performance is not really of concern due to triviality. Algorithms may be modified to make best case performance trivial for small input by hardcoding. In these cases, the best case performance is effectively meaningless!

Often, it is of more concern to perform worst case analysis, i.e. identifying the inputs that cause the algorithm to have the longest running time (or use the most space) and identifying the running time and usage bounds. Why is this useful?

- The worst case running time of an algorithm gives an upper bound on the running time for any input. Knowing this provides a guarantee that the algorithm never takes any longer.
- For some algorithms, the worst case may occur fairly often.
- Often, the “average case” is roughly as bad as the worst case.

Finally, in some cases we may be interested in the average case running time of an algorithm, where we would use probabilistic analysis to examine particular algorithms. This doesn’t surface too much, as what constitutes an “average input” is often not apparent. Often, we would then assume that all inputs of a particular size are equally likely (uniform distribution). This assumption is often violated in practice, so we might modify an algorithm to be randomized, to enable a probabilistic analysis and expected running time.

Big-Oh Definitions

Presented below is a brief overview of asymptotic notation that will be fundamental in this course.

---

### Big-Oh Notation

**Definition (1).** \( f(n) \in O(g(n)) \) if there exist positive constants \( n_0 \) and \( c \) such that \( f(i) \leq cg(i) \) for all \( i \geq n_0 \).

**Definition (2).** \( f(n) \in O(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \) or a constant.

Simplified: If \( f(n) \) is in \( O(g(n)) \), \( g(n) \) is an asymptotic upper bound for \( f(n) \).

### Little-o Notation

**Definition (1).** \( f(n) \in o(g(n)) \) if for any positive constant \( c \), there exists a positive constant \( n_0 \) such that \( f(i) < cg(i) \) for all \( i \geq n_0 \).

**Definition (2).** \( f(n) \in o(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \).

Note: \( f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n)) \)
Big-Omega Notation

**Definition (1).** \( f(n) \in \Omega(g(n)) \) if there exist positive constants \( n_0 \) and \( c \) such that \( f(i) \geq cg(i) \) for all \( i \geq n_0 \).

**Definition (2).** \( f(n) \in \Omega(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \) or a constant.

Simplified: If \( f(n) \) is \( \Omega(g(n)) \), \( g(n) \) is an asymptotic lower bound for \( f(n) \).

Little-\( \omega \) Notation

**Definition (1).** \( f(n) \in \omega(g(n)) \) if for any positive constant \( c \), there exists a positive constant \( n_0 \) such that \( f(i) > cg(i) \) for all \( i \geq n_0 \).

**Definition (2).** \( f(n) \in \omega(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \).

Note: \( f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n)) \)

Big-Theta Notation

**Definition (1).** \( f(n) \in \Theta(g(n)) \) if and only if \( f(n) \in O(g(n)) \) and \( f(n) \in \Omega(g(n)) \).

**Definition (2).** \( f(n) \in \Theta(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = a \) nonzero constant.

Simplified: If \( f(n) \) is \( \Theta(g(n)) \), \( g(n) \) is an asymptotically tight bound for \( f(n) \).

The notations refer to classes of functions. When you read \( f(n) = O(g(n)) \), this is equivalent to the statement: \( f(n) \in O(g(n)) \). Specifically, \( f(n) \) is in the class of functions which are asymptotically bounded above by \( g(n) \). Likewise, Big-\( \Omega \), Big-\( \Theta \), etc., all reflect classes of functions.

Problems (Big-Oh)

Problem 1a
Prove that \( 3n^2 + 100n = \Theta(5n^2) \)

Solution.

\[
\lim_{n \to \infty} \frac{3n^2 + 100n}{5n^2} = \lim_{n \to \infty} \frac{3n + 100}{5n} = \lim_{n \to \infty} \frac{3n}{5n} = 3/5 \]

\( 3n^2 + 100n = \Theta(5n^2) \)

Problem 1b
Prove that \( n \log n = \Omega(n) \)
Solution.

We will prove this by induction.

**Base Case:** \( n = 4 \). \( 4 \log 4 = 8 > 4 \), so this holds.

**Induction Hypothesis:** Assume that \( k \log k \geq k \) for some \( k \geq 4 \).

**Induction Step:** We need to show that \((k + 1) \log(k + 1) \geq k + 1 \)

Log is monotonically increasing so \( \log(k + 1) > \log k \)

\[
(k + 1) \log(k + 1) > (k + 1) \log k \\
= k \log k + \log k \\
> k \log k + 1 \\
> k + 1
\]

Problem 2

Prove or disprove the following statement:

\( \lg(n!) \) is \( \Theta(n \lg n) \).

**Solution.**

We first show \( \lg(n!) \) is \( O(n \lg n) \). Picking \( c = 1 \) and \( n_0 = 1 \), we have

\[
\lg(n!) = \sum_{i=1}^{n} \lg i \leq n \lg n
\]

This is clearly true for all \( n > n_0 \). Therefore, we are done.

We then show that \( \lg(n!) \) is \( \Omega(n \lg n) \). Our strategy is to find an easier to work with lower-bound for \( \lg n! \) that is larger than some \( cn \lg n \).

\[
\lg n! = \lg 1 + \lg 2 + \cdots + \lg n \\
\geq \lg \frac{n}{2} + \lg \left(\frac{n}{2} + 1\right) + \cdots + \lg n \quad \text{delete the first half of the terms} \\
\geq \frac{n}{2} \cdot \lg \frac{n}{2} \quad \text{replace remaining terms by smallest one}
\]

Choosing \( c = \frac{1}{4} \) and \( N = 4 \), it is clear that \( \frac{n}{4} \lg \frac{n}{2} \geq \frac{n}{4} \lg n \) with some algebraic manipulation:

\[
\frac{n}{2} \lg \frac{n}{2} \geq \frac{n}{4} \lg n \\
\frac{n}{2} \lg n - \frac{n}{2} \geq \frac{n}{4} \lg n \\
2n \geq 2n \\
\lg n \geq 2
\]

Therefore, \( \lg(n!) \) is \( \Omega(n \lg n) \).

Problem 3

Solve the following recurrence:

\[
T(n) = \begin{cases} 
2T\left(\frac{n}{2}\right) + n^2 & n \geq 1 \\
1 & \text{otherwise}
\end{cases}
\]
Solution.

We first solve by iteration. First, we may assume that \( n \) is some power of 2 such that \( n = 2^k \implies k = \lg n \).

\[
T(n) = 2T\left(\frac{n}{2}\right) + n^2
\]

\[
= 2 \left[ 2T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^2 \right] + n^2
\]

\[
= 2 \left[ 2 \left[ 2T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2 \right] + \left(\frac{n}{2}\right)^2 \right] + n^2
\]

We see that if we continue to expand the recurrence, we will end up with a \( 2^kT\left(\frac{n}{2^k}\right) = 2^k \) term in the base case. Fully expanded, we get:

\[
T(n) = 2^k + n^2 + 2\left(\frac{n}{2}\right)^2 + 2^2\left(\frac{n}{2^2}\right)^2 + \cdots + 2^{k-1}\left(\frac{n}{2^{k-1}}\right)^2
\]

\[
= 2^k + n^2 + \frac{n^2}{2} + \frac{n^2}{2^2} + \cdots + \frac{n^2}{2^{k-1}}
\]

\[
= 2^k + n^2 \sum_{i=0}^{k-1} \left(\frac{1}{2}\right)^i
\]

\[
= 2^k + n^2 \frac{1 - \left(\frac{1}{2}\right)^k}{1 - \frac{1}{2}} \quad \text{geometric series}
\]

\[
= 2\lg n + 2n^2 \left[1 - \frac{1}{n}\right] \quad \text{substituting } k = \lg n
\]

\[
= n + 2n^2 - 2n = \Theta(n^2)
\]

That’s a lot of tedious algebra, but it gets the job done. Now, let’s try a different approach—we’ll expand the recurrence using a recursion tree:

On the right side, we can see the total amount of work done at each level of the tree. Our sum becomes immediately apparent without all the initial algebraic soup! I would argue that this approach helps form a much more intuitive understanding of the problem.
Problem 4

You are given the following algorithm for Bubble-Sort:

Algorithm 1 Bubble Sort

```
function Bubble-Sort(A, n)
    for i ← 0 to n - 2 do
        for j ← 0 to n - i - 2 do
                swap(A[j], A[j + 1])
            end if
        end for
    end for
end function
```

Given some sequence \(\langle a_1, a_2, \ldots, a_n\rangle\) in \(A\), we say an inversion has occurred if \(a_j < a_i\) for some \(i < j\). At each iteration, Bubble-Sort checks the array \(A\) for an inversion and performs a swap if it finds one. How many swaps does Bubble-Sort perform in the worst-case and in the average-case?

Solution.

It should be fairly obvious that the worst-case scenario for bubble-sort occurs when \(A\) contains elements in reverse-sorted order. Let \(I\) denote the number of inversions. In this situation, the total number of inversions is the exact number of possible pairs of elements, as each swap removes exactly one inversion:

\[
I_{\text{worst}} = \binom{n}{2}
\]

In the average-case scenario, we determine the expected number of inversions in a random array of \(n\) elements. Specifically, we consider a random permutation of \(n\) distinct elements, \(\langle a_1, a_2, \ldots, a_n\rangle\).

Let \(X_{ij}\) denote an indicator R.V. such that,

\[
X_{ij} = \begin{cases} 
1 & \text{if } a_i, a_j \text{ inverted} \\
0 & \text{otherwise}
\end{cases}
\]

Thus, we have

\[
I_{\text{average}} = \sum_{i,j : i < j} X_{ij}
\]

\[
E[I] = \sum_{i,j : i < j} E[X_{ij}]
\]

\[
= \sum_{i,j : i < j} \Pr[X_{ij} = 1]
\]

\[
= \binom{n}{2} \frac{1}{2}
\]

\[
= \frac{n(n-1)}{4}
\]

In the above, we let \(\Pr[X_{ij} = 1] = \frac{1}{2}\) because in a random permutation (uniform probability), the probability of \(a_i < a_j\) is the same as \(a_i > a_j\).
Problem 5

Solve the following recurrence:

\[
T(n) = \begin{cases} 
  T(n - 1) + 2^n & n \geq 1 \\
  1 & \text{otherwise}
\end{cases}
\]

Solution:

We first represent the recurrence as a summation. As \( n \) is subtracted by 1 in each step, there will be \( n \) terms in the summation:

\[
T(n) = \sum_{j=1}^{n} 2^j
\]

Using the formula for the sum of a geometric series, we get

\[
T(n) = 2^1 + 2^2 + \cdots + 2^n = 2 \cdot \frac{2^n - 1}{2 - 1} = 2^{n+1} - 2 = \Theta(2^{n+1}) = \Theta(2^n)
\]