Basic problems:

1. [Dasgupta et al., Ex. 0.1] In each of the following situations, indicate the relation (if any) between the orders of growth of functions \( f \) and \( g \), i.e. whether \( f = \Theta(g) \), \( f = o(g) \), \( g = o(f) \), or the two functions are incomparable. (The notation “\( \log n \)” denotes by default base-2 logarithms.)

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( g(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) ( n - 100 )</td>
<td>( n - 200 )</td>
</tr>
<tr>
<td>(b) ( n^{1/2} )</td>
<td>( n^{2/3} )</td>
</tr>
<tr>
<td>(c) ( 100n + \log n )</td>
<td>( n + \log^2 n )</td>
</tr>
<tr>
<td>(d) ( n \log n )</td>
<td>( 10n \log 10n )</td>
</tr>
<tr>
<td>(e) ( \log 2n )</td>
<td>( \log 3n )</td>
</tr>
<tr>
<td>(f) ( 10 \log n )</td>
<td>( \log n^2 )</td>
</tr>
<tr>
<td>(g) ( n^{1.01} )</td>
<td>( n \log^2 n )</td>
</tr>
<tr>
<td>(h) ( n^2 / \log n )</td>
<td>( n \log^2 n )</td>
</tr>
<tr>
<td>(i) ( n^{0.1} )</td>
<td>( \log^{10} n )</td>
</tr>
<tr>
<td>(j) ( (\log n)^{\log n} )</td>
<td>( n / \log n )</td>
</tr>
<tr>
<td>(k) ( \sqrt{n} )</td>
<td>( \log^3 n )</td>
</tr>
<tr>
<td>(l) ( n^{1/2} )</td>
<td>( 5^{\log n} )</td>
</tr>
<tr>
<td>(m) ( n 2^n )</td>
<td>( 3^n )</td>
</tr>
<tr>
<td>(n) ( 2^n )</td>
<td>( 2^{n+1} )</td>
</tr>
<tr>
<td>(o) ( n! )</td>
<td>( 2^n )</td>
</tr>
<tr>
<td>(p) ( (\log n)^{\log n} )</td>
<td>( 2^{\log^2 n} )</td>
</tr>
<tr>
<td>(q) ( \sum_{i=1}^{n} i^k )</td>
<td>( n^{k+1} )</td>
</tr>
</tbody>
</table>

**Hints:** Check Lecture 2, Slide 12. For (j) and (p) note that \( a^b = (2^{\log a})^b = 2^{b \log a} \).

2. Show that

\[
\log(n!) = \sum_{i=1}^{n} \log i = \Theta(n \log n).
\]

**Hint:** Construct an upper bound for the value of the sum using the \( \log n \) term, and a lower bound using the term \( \log \frac{n}{2} \).
3. Analyse the worst-case time complexity of the following algorithm:

**Algorithm 1: Bubble sort.**

```python
1 function BUBBLESORT (A[1...n])
    Input: Integer array A[1...n].
    Output: Array A, with same items in increasing order.
2 for i ← n downto 2 do
3    for j ← 1 to i − 1 do
5            Exchange elements A[j] and A[j + 1]
6        end
7    end
8 end
```

Hint: Check Lecture 2, Slides 4–6.

4. The binomial coefficients are defined by the recurrence equation:

\[
\binom{n}{k} = \begin{cases} 
1, & \text{if } k = 0 \text{ or } k = n, \\
\binom{n-1}{k-1} + \binom{n-1}{k}, & \text{if } 0 < k < n.
\end{cases}
\]

Design an algorithm that computes the value of the binomial coefficient \( \binom{n}{k} \) in time \( O(nk) \).

Hint: For increasing values of \( n \), tabulate values of \( \binom{n}{k} \), \( k = 0, \ldots, n \).

Advanced problems:

5. [Dasgupta et al., Ex. 0.3] The Fibonacci numbers \( F_0, F_1, F_2, \ldots \) are defined by the rule

\[
F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \ (n \geq 2).
\]

In this problem we will confirm that this sequence grows exponentially fast and obtain some bounds on its growth.

(a) Use induction to prove that \( F_n \geq 2^{0.5n} \) for \( n \geq 6 \).

(b) Find a constant \( c < 1 \) such that \( F_n \leq 2^{cn} \) for all \( n \geq 0 \). Show that your answer is correct.

(c) What is the largest \( c \) you can find for which \( F_n = \Omega(2^n) \)?

**Solution.** We have \( F_0 = 0, F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, F_7 = 13 \).

Let \( P(n) \) be the claim: for all \( 6 \leq k \leq n \) it holds that \( F_k \geq 2^{0.5k} \). Let us prove that \( P(n) \) holds for all \( n \geq 7 \). We proceed by induction on \( n \). The base case \( P(7) \) is established by checking that \( F_6 = 8 \leq 2^{0.5 \cdot 6} \) and \( F_7 = 13 \leq 2^{0.5 \cdot 7} \). To establish the inductive step, assume that \( P(n) \) holds, \( n \geq 7 \). We must show that \( P(n + 1) \) holds. Because \( P(n) \) holds, we have \( F_k \geq 2^{0.5k} \) for all \( 6 \leq k \leq n \). It remains to verify that \( F_{n+1} \geq 2^{0.5(n+1)} \). From \( k = n \) and \( k = n - 1 \), we have

\[ F_{n+1} = F_n + F_{n-1} \geq 2^{0.5n} + 2^{0.5(n-1)} = 2^{0.5(n-1)}(2^{0.5} + 1) > 2^{0.5(n-1)} \cdot 2 = 2^{0.5(n+1)}, \]

which shows that \( P(n + 1) \) holds. This completes the inductive step.

To establish (b) and (c), let \( x \) be an indeterminate and set \( a_n = x^n \). Suppose that \( a_n = a_{n-1} + a_{n-2} \) holds for \( n \geq 2 \). Then, we must have \( x^n = x^{n-1} + x^{n-2} \). Dividing both sides by \( x^{n-2} \), we conclude
that \(x^2 = x + 1\), or equivalently, \(x^2 - x - 1 = 0\). Solving for \(x\), we find that either \(x = (1 - \sqrt{5})/2\) or \(x = (1 + \sqrt{5})/2\). For \(n \geq 0\) let us set

\[
b_n = \lambda_1 \left( \frac{1 - \sqrt{5}}{2} \right)^n + \lambda_2 \left( \frac{1 + \sqrt{5}}{2} \right)^n
\]

for some constants \(\lambda_1\) and \(\lambda_2\). Let us require that \(b_0 = 0\) and \(b_1 = 1\). That is, we require that \(\lambda_1 + \lambda_2 = 0\) and \(\lambda_1 (1 - \sqrt{5}) + \lambda_2 (1 + \sqrt{5}) = 2\). Solving for \(\lambda_1\), we have \(\lambda_1 (1 - \sqrt{5}) + (-\lambda_1)(1 + \sqrt{5}) = 2\), or equivalently, \(\lambda_1 = -1/\sqrt{5}\). Hence, \(\lambda_2 = 1/\sqrt{5}\). Note that \(b_n\) and \(F_n\) agree for \(n \leq 1\), and hence by induction (indeed, \(b_n = b_{n-1} + b_{n-2}\) and \(F_n = F_{n-1} + F_{n-2}\)) they agree for all \(n \geq 2\). Thus,

\[
F_n = -\frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^n + \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n.
\]

Because \(|(1 - \sqrt{5})/2| < 1\), the first summand goes to 0 as \(n \to \infty\). In fact, for all \(n \geq 0\) we have that \(F_n\) is equal to the second summand \(((1 + \sqrt{5})/2)^n/\sqrt{5}\) rounded to the nearest integer!

Thus, in (b) and (c) we can take \(c = \log((1 + \sqrt{5})/2)\).

6. Analyse the worst-case time complexity of the following algorithm:

---

**Algorithm 2**: Binary search.

1. function **BINARY SEARCH** \(A[1 \ldots n], a\):
2.     Input: Integer array \(A[1 \ldots n]\) with elements in increasing order, integer \(a\).
3.     Output: Index \(k\) such that \(A[k] = a\), or fail if no such \(k\) exists.
4.     Set \(l \leftarrow 1, r \leftarrow n\);
5.     if \(l > r\) then return fail
6.     Set \(k \leftarrow \lfloor (l + r)/2 \rfloor\)
7.     if \(A[k] < a\) then
8.         Set \(l \leftarrow k + 1\), go to line 3
9.     if \(A[k] > a\) then
10.        Set \(r \leftarrow k - 1\), go to line 3
11.    return \(k\).

---

**Solution**. Denote by \(M\) the number of times line 3 is executed. It is immediate from the structure of the algorithm that the running time is \(\Theta(M)\).

It remains to derive bounds for \(M\). We claim that when line 3 is executed for the \(j\)th time, with \(j = 1, 2, \ldots, M\), the values of the variables \(l\) and \(r\) satisfy \(r - l + 1 \leq n/2^{j-1} = 2^{1-j+\log n}\) (If one thinks of \(j\) as an explicit counter initialized to one and incremented at line 6 and 7 before going to line 3, then the inequality can also be thought of as a loop invariant\(^1\) parametrized by \(r, l, j\)).

Let us proceed by induction on \(j\). For \(j = 1\) the claim is immediate. So let us assume that the claim holds for \(j\) and consider the case \(j + 1\). Denote by \(r_j\) and \(l_j\) the values of the variables \(r\) and \(l\) when line 3 is executed for the \(j\)th time.

---

\(^{1}A\ loop\ invariant\ is\ a\ logical\ assertion\ which\ holds\ true\ immediately\ before\ and\ immediately\ after\ each\ iteration\ of\ a\ loop.\ It\ is\ commonly\ used\ in\ proving\ program\ correctness.\)
If line 6 gets executed before the \((j+1)\)th execution of line 3, we have
\[
\begin{align*}
r_{j+1} - l_{j+1} + 1 &= r_j - (k + 1) + 1 \\
&= r_j - \lfloor (l_j + r_j) / 2 \rfloor \\
&\leq r_j - (l_j + r_j - 1) / 2 \\
&\leq (r_j - l_j + 1) / 2 \\
&= 2^{1-j+i\log n} / 2 \\
&= 2^{1-(j+1)+i\log n}.
\end{align*}
\]
If line 8 gets executed before the \((j+1)\)th execution of line 3, we have
\[
\begin{align*}
r_{j+1} - l_{j+1} + 1 &= k - 1 - l_j + 1 \\
&= \lfloor (l_j + r_j) / 2 \rfloor - l_j \\
&\leq (l_j + r_j) / 2 - l_j \\
&\leq (r_j - l_j + 1) / 2 \\
&= 2^{1-j+i\log n} / 2 \\
&= 2^{1-(j+1)+i\log n}.
\end{align*}
\]
This completes the induction.

It is immediate from the structure of the algorithm that each execution of line 3 sees \(r - l\) decrease by at least one from the previous execution of line 3. Thus, when \(r - l + 1 \leq 1\) the algorithm terminates in at most two executions of line 3. By the claim, \(r_j - l_j + 1 \leq 1\) holds for all \(j \geq 1 + \log n\). Thus, \(M \leq 3 + \log n\) and hence \(M = O(\log n)\)