Exercises are from “Algorithm Design and Applications, Goodrich and Tamassia.”

1. Rank the following functions by their asymptotic growth rate in ascending order. In your solution, group those functions that are big-Theta of one another (all log functions are base 2):

\[
\begin{array}{cccccc}
6 \cdot n \log n & 2^{100} & \log \log n & \log^2 n & 2^{\log n} \\
2^{2n} & \lfloor \sqrt{n} \rfloor & n^{0.01} & 1/n & 4n^{3/2} \\
3n^{0.5} & 5n & \lfloor 2 \cdot n \log^2 n \rfloor & 2^n & n^2 \\
4^n & n^3 & n^2 \log n & 4^{\log n} & \sqrt{\log n}
\end{array}
\]

2. Graph the functions \(12n, 6n \log n, n^2, n^3,\) and \(2^n\) using a logarithmic scale on the \(y\) axis; that is, if the function value \(f(n)\) is \(y\), plot this as a point with \(x\)-coordinate at \(n\) and \(y\)-coordinate at \(\log y\).

3. For each function \(f(n)\) and time \(t\) in the following table, determine the largest size \(n\) of a problem that can be solved in time \(t\) assuming that the algorithm to solve the problem takes \(f(n)\) microseconds.

<table>
<thead>
<tr>
<th>(f(n))</th>
<th>1 Second</th>
<th>1 Hour</th>
<th>1 Month</th>
<th>1 Century</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\sqrt{n})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n \log n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n^2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n^3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2^n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n!)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Algorithm \texttt{algo1} uses \(10n \log n\) operations, while algorithm \texttt{algo2} uses \(n^2\) operations. What is the value of \(n_0\), such that \texttt{algo1} is better than \texttt{algo2} for \(n \geq n_0\).

5. Show that \((n + 1)^5\) is \(O(n^5)\).

6. Show that \(2^{n+1}\) is \(O(2^n)\).

7. Show that \(n^3 \log n\) is \(\Omega(n^3)\).
8. For each of the following, give a big-Oh characterization in terms of \( n \):

(a) \[
s = 1 \\
\text{for } i = 1 \text{ to } 2n \text{ do} \\
\quad s = s \times i
\]

(b) \[
s = 0 \\
\text{for } i = 1 \text{ to } 2n \text{ do} \\
\quad \text{for } j = 1 \text{ to } i \text{ do} \\
\quad \quad s = s + i
\]

(c) \[
s = 0 \\
\text{for } i = 1 \text{ to } n \times n \text{ do} \\
\quad \text{for } j = 1 \text{ to } i \text{ do} \\
\quad \quad s = s + i
\]

9. Suppose you are given an algorithm \( A \) to find an element \( e \) in an \( n \times n \) matrix \( M \). The algorithm \( A \) iterates over the rows of \( M \) and calls another algorithm \( B \) on each one, until \( e \) is found or it has searched all rows of \( M \). What is the worst-case running time of \( A \) in terms of \( n \)? (\( B \) performs a linear search in \( O(n) \) time)

10. Suppose you run two algorithms, \( P \) and \( Q \), on many randomly generated data sets. \( P \) is an \( O(n \log n) \)-time algorithm and \( Q \) is an \( O(n^2) \)-time algorithm. After your experiments you find that if \( n < 100 \), \( Q \) actually runs faster, and only when \( n \geq 100 \), \( P \) is faster. Explain why this scenario is possible, including numerical examples.

11. An array \( A \) contains \( n - 1 \) unique integers in the range \([0, n - 1]\); that is, there is one number from this range that is not in \( A \). Describe an \( O(n) \) time algorithm for finding that number. You are allowed to use only \( O(1) \) additional memory besides the array \( A \) itself.

12. Suppose that each row of an \( n \times n \) matrix \( M \) consists of only binary digits, such that, in any row of \( M \), all the 1’s come before any 0’s in that row. Assuming \( M \) is already in memory, describe an \( O(n) \)-time algorithm for finding the row of \( M \) that contains the most 1’s.

13. Given an array \( A \) of \( n - 2 \) unique integers in the range \([1, n]\), describe an \( O(n) \)-time algorithm for finding the two integers in the range \([1, n]\) that are not in \( A \). You are allowed to use only \( O(1) \) additional memory besides the array \( A \) itself.
14. An \( n \)-degree polynomial \( p(x) \) is an equation of the form:

\[
p(x) = \sum_{i=0}^{n} a_i x^i
\]

where \( x \) is a real number and each \( a_i \) is a constant.

(a) Describe a simple \( O(n^2) \)-time method for computing \( p(x) \) for a particular value of \( x \).

(b) Consider now a rewriting of \( p(x) \) as:

\[
p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)\ldots))
\]

which is known as **Horner's method**. Using the big-Oh notation, characterize the number of multiplications and additions this method uses.

15. Give the values of the maximum suffix sums \( M_t \) for the array \([-2, -4, 3, -1, 5, 6, -7, -2, 4, -3, 2]\).

16. Describe how to modify the **MaxsubFastest** algorithm so that it uses just a single loop and, instead of computing \( n + 1 \) different \( M_t \) values, it maintains just a single variable \( M \).

17. Given an array \( A \), of \( n \) integers, describe a method to find the longest subarray of \( A \) such that all the numbers in that subarray are in sorted order. What is the running time of your algorithm?

18. Given the array \( A \) with elements \([22, 15, 36, 44, 10, 3, 9, 13, 29, 25]\), illustrate the performance of the **selection-sort** algorithm on \( A \).

19. Given the array \( A \) with elements \([22, 15, 36, 44, 10, 3, 9, 13, 29, 25]\), illustrate the performance of the **insertion-sort** algorithm on \( A \).

20. Let \( S \) be an array of \( n \) elements. An inversion in \( S \) is a pair of indices \( i \) and \( j \) such that \( i < j \) but \( S[i] > S[j] \). Give an example of a sequence of \( n \) integers with \( \Omega(n^2) \) inversions.

21. Given an array \( A \), show that insertion-sort runs in \( O(n + I) \) time, where \( I \) is the number of inversions (as defined in problem 20) in the array \( A \).

22. Given and array \( A \), of \( n \) numbers in the range from 1 to \( n \), describe an \( O(n) \) time method for finding the mode, that is, the number that occurs most frequently in \( A \).

23. Suppose we are given a sequence \( S \) of \( n \) integers in the range from 1 to \( n^3 \). Give a linear-time algorithm for determining whether there are two equal numbers in \( S \).