

You are strongly encouraged to attempt all of the following exercises for understanding.

However, only solutions to the final three questions (A, B, C) are to be submitted for assessment.

Practice Exercises

1. Rank the following functions by order of growth:

$$n^2, \quad n!, \quad 2^{2n}, \quad \lg^2 n, \quad 2^{2^{n+1}}, \quad n \lg n, \quad n2^n, \quad n^3.$$

For each pair $f(n), g(n)$ of adjacent functions in this ranking, provide positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

2. Show that for any positive real constants a and b , $(n + a)^b = \Theta(n^b)$.
3. Give asymptotically tight upper bounds for each of the following recurrences $T(n)$. In each case, justify your solution by either naming the particular case of the (Simplified) Master Theorem which applies, or by iterating (unrolling) the recurrence.

(a) $T(n) = 16T(n/4) + n^2$.

(b) $T(n) = 2T(n/4) + \sqrt{n}$.

(c) $T(n) = 2T(7n/10) + n^2$.

4. **The Majority Problem.** For this problem, you are given an unsorted array of n elements, and asked to find the majority element, if it exists. (An element is a majority element if it appears more than $n/2$ times in the array.)

- (a) Argue that if $A[i] \neq A[j]$ then removing $A[i]$ and $A[j]$ from the array preserves the majority element, if it exists. That is, if x is a majority element of the original array, then x must still be a majority element of the shorter array with the two unequal elements removed.
- (b) Give a counter-example to show that the converse of the above fact is false. That is, give an array which has no majority element but the removal of two unequal members produces a shorter array which does have a majority element.
- (c) Using the above observations, design an algorithm for the majority problem which runs in linear time. Present your algorithm in pseudocode, and justify its correctness and running time.

5. **Chip Testing.** You have n supposedly identical computer chips that in principle are capable of testing each other. Your test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the answer of a bad chip cannot be trusted.

- (a) Show that if at least half of the chips are bad, you cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test.
- (b) Consider the problem of identifying a single good chip from a collection of n chips, more than half of which are good. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce this problem to one of roughly half the size.
- (c) Show that the problem of identifying all of the good chips in a collection of n chips, more than half of which are good, can be done using $\Theta(n)$ pairwise tests.

6. **The Skyline Problem.** Design an efficient algorithm which takes as input the locations and heights of n rectangular buildings in a 2-dimensional city, and computes the 2-dimensional skyline of these buildings, eliminating hidden lines.

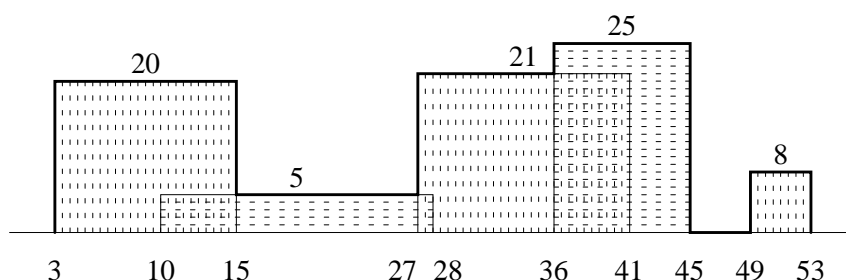
Each building is represented by a 3-tuple (L_i, H_i, R_i) , where L_i and R_i denote the left and right x -coordinates of the building, respectively, and H_i denotes the building's height. A skyline is represented as a list of x -coordinates and heights, arranged in order from left to right. As an example, the input

$$(36, 25, 45), (3, 20, 15), (49, 8, 53), (10, 5, 28), (27, 21, 41).$$

would result in the output

$$(3, 20, 15, 5, 27, 21, 36, 25, 45, 0, 49, 8, 53).$$

Pictorially, we have the following situation.



Analyze the time complexity of your algorithm.

A. Determine whether or not each of the following claims is true or false in general. Give a rigorous justification in each case. (Assume that $f(n), g(n) \geq 0$ for all $n \geq 0$.)

(a) $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

(b) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$.

B. Give asymptotically tight upper bounds for each of the following recurrences $T(n)$. In each case, justify your solution by either naming the particular case of the (Simplified) Master Theorem which applies, or by iterating (unrolling) the recurrence.

(a) $T(n) = T(n-2) + 1$.

(b) $T(n) = 9T(n/4) + n^2$.

(c) $T(n) = 3T(n/2) + n/2$.

C. Consider the following sorting algorithm.

TRISORT(A, i, j)

- 1 **if** $i+1 = j$ **and** $A[i] > A[j]$ **then** $A[i] \leftrightarrow A[j]$
- 2 **if** $i+1 < j$ **then** \triangleright *There are at least three elements*
- 3 $k \leftarrow \lfloor (j-i+1)/3 \rfloor$ \triangleright *Divide the interval into three parts*
- 4 TRISORT($A, i, j-k$) \triangleright *Sort the first two-thirds*
- 5 TRISORT($A, i+k, j$) \triangleright *Sort the last two-thirds*
- 6 TRISORT($A, i, j-k$) \triangleright *Sort the first two-thirds again*

(a) Argue that TRISORT($A, 1, \text{length}(A)$) correctly sorts the array A .

(b) Give a recurrence for the running time of this algorithm, and use this to give a tight asymptotic bound on the running time. Is this a good algorithm?