Theory Assignment

Algorithms and Data Structures, Spring 2020

Date: March 9, 2020

Hand-in on learnIT before March 20, 2020, 18:00

This mandatory assignment consists of theoretical exercises. Mastering them will develop your abstract reasoning skills and frustration tolerance, and deepen your understanding of algorithm design and analysis. Some exercises may require a lot of thought, but answers should be short. You must hand in the solution in groups of size 3 or 4. Keep in mind that the main purpose of the hand-in is *communication*, so put yourselves in the shoes of your reader. In several rounds of editing, polish the structure, presentation, grammar, etc. before you submit your hand-in. Obtuse writing will not be read.

Exercise 1: Theta-notation

We say that two function have the same asymptotic growth (written $\Theta(g) = \Theta(h)$) if both of the following conditions are satisfied: g is in $O(h)$ **and** h is in $O(g)$.

For each of the following functions g_i, give a function h_i that is as simple as possible and exhibits the same asymptotic growth. You *do not* need to justify your claims in writing, we only want to see the function h_i.

(a) $g_1(n) = 0.01 \cdot n^{19} - 1/n^2 + \log^{920} n$

(b) $g_2(n) = (n^3 + n + n \log n) \cdot (n^{22} + \log n) + n^n + n^{-n} + n^{1/n}$

(c) $g_3(n) = n \cdot (\cos^2(n) + \sin^2(n))$

(d) Is this true or false? $\Theta(\log_{99}(n^{21})) = \Theta(\log_2 n)$
Hint: Recall logarithm rules.

Theta (uppercase Θ, lowercase θ or ϑ) is the eighth letter of the Greek alphabet.

For example, we have $\Theta(13n^9 + 4n) = \Theta(n^9)$.

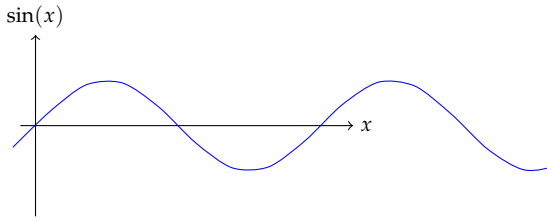

Figure 1: A plot of the sin function.

Exercise 2: Oh-notation

Argue why $n^{\log_2 n}$ is in $O(2^n)$. *Hint:* Argue mathematically that $n^{\log_2 n}/2^n$ is at most a constant, even when n grows larger and larger (to infinity). Use logarithm rules and the fact that the logarithm is monotone (that is, $\log x \leq \log y$ holds whenever $x \leq y$).

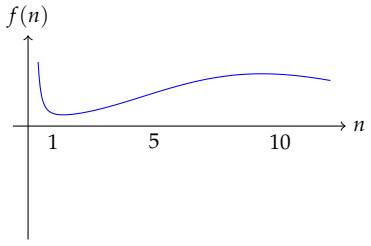

Figure 2: A plot of the function $f(n) = n^{\log_2 n}/2^n$.

Exercise 3: Algorithm design

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ be two arrays, containing integers in descending order. Design an algorithm `Find`(A, B, k) that finds the k-th largest element in the union of A and B in time $O(\log k)$.

For example, if $A = (92, 42, 31, 7)$ and $B = (91, 39, 33, 2)$, then the output of `Find`$(A, B, 3)$ is 42.

You may assume that k is a power of two, and that all $2n$ numbers are pairwise distinct. However, it is important that you argue (**using at most 3–5 sentences**) why the algorithm you design is correct and why its running time is $O(\log k)$.

Exercise 4: Analysis of an algorithm

Consider the following Algorithm B and its subroutine C. Algorithm B gets as input an array A of n integers. For convenience, we call the first position of the array $A[1]$ and the last one $A[n]$ (rather than the more familiar $A[0]$ and $A[n-1]$). Moreover, we assume that every number that occurs in A occurs exactly once. We assume that each comparison and each swap costs 1, and all other operations are free.

The subroutine C gets as input merely an index j with $1 \leq j \leq n$, and shares the array A with the main algorithm B. Then, C(j) is defined as follows:

C1 If $j = 1$ holds, then stop.
C2 If $A[j-1] > A[j]$ holds, then:
C3 Swap $A[j]$ and $A[j-1]$.
C4 Call C$(j-1)$ recursively.
C5 Else: Stop.

With this subroutine at hand, algorithm B works as follows:

B1 For i from 2 to n:
B2 Call C(i).

(a) What does algorithm B do?
(b) Analyze the running time of algorithm B in the worst case. That is, give an asymptotic upper bound for the maximal running time and justify your answer. Moreover, show that your bound is tight, that is, describe worst-case instances on which the running time of the algorithm asymptotically coincides with the upper bound. (**Use at most 5-10 sentences.**)

Exercise 5: Some graph theory

(a) A forest is a graph that does not contain any cycles. If a forest has n vertices and k connected components, then how many edges does it have? Justify your answer.
(b) A graph $G = (V, E)$ is bipartite if $V = L \cup R$ is the union of two vertex sets such that all edges of G go from a vertex in L to a vertex in R. Argue whether or not the following is true: A graph is bipartite if and only if it does not contain a cycle of odd length. Keep your answer readable and concise. Make sure every sentence

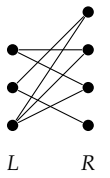

Figure 3: A bipartite graph with three vertices in L and four in R.

you write is relevant to the task. (*Hint:* Consider each implication separately: 1. "bipartite implies no odd-length cycle" and 2. "no odd-length cycle implies bipartite".)

Exercise 6: Hashing

(a) Imagine that an algorithm requires us to hash strings containing English phrases. Knowing that strings are stored as sequences of characters, Alyssa P. Hacker decides to simply use the sum of those character values (modulo the size of her hash table) as the string's hash. Will the performance of her implementation match the expected value shown in lecture? (Property L in Sedgewick–Wayne)

1. Yes, the sum operation will space strings out nicely by length.
2. Yes, the sum operation will space strings out nicely by the characters they contain.
3. No, because reordering the words in a string will not produce a different hash.
4. No, because the independence condition of the uniform hashing assumption (Assumption J in Sedgewick–Wayne) is violated.

(b) Alyssa decides to implement both collision resolution and dynamic resizing for her hash table. However, she doesn't want to do more work than necessary, so she wonders if she needs both to maintain the correctness and performance she expects. After all, if she has dynamic resizing, she can resize to avoid collisions; and if she has collision resolution, collisions don't cause correctness issues. Which statement about these two properties is true?

1. Dynamic resizing alone will preserve both properties.
2. Dynamic resizing alone will preserve correctness, but not performance.
3. Collision resolution alone will preserve performance, but not correctness.
4. Both are necessary to maintain performance and correctness.

(c) In the lecture, we discussed doubling the size of our hash table when it becomes too full. Ivy H. Crimson begins to implement this approach (that is, she lets $m' = 2m$) but stops when it occurs to her that she might be able to avoid wasting half of the memory the table occupies on empty space by letting $m' = m + k$ instead, where k is some constant. Does this work? If so, why do you think we don't do it? There is a good theoretical reason as well as several additional practical concerns; a complete answer will touch on both points.