1 3.1-1 as stated in book - we did this one in class for practice

Problem

Let \( f(n) \) and \( g(n) \) be asymptotically nonnegative functions. Using the basic definition of \( \Theta \)-notation, prove that \( \max(f(n), g(n)) = \Theta(f(n) + g(n)) \).

Solution

The functions \( f(n) \) and \( g(n) \) are asymptotically nonnegative, there exists \( n_0 \) such that \( f(n) \geq 0 \) and \( g(n) \geq 0 \) for all \( n \geq n_0 \). Thus, we have that for all \( n \geq n_0 \), \( f(n) + g(n) \geq f(n) \geq 0 \) and \( f(n) + g(n) \geq g(n) \geq 0 \). Adding both inequalities (since the functions are nonnegative), we get \( f(n) + g(n) \geq \max(f(n), g(n)) \) for all \( n \geq n_0 \). This proves that \( \max(f(n), g(n)) \leq c(f(n) + g(n)) \) for all \( n \geq n_0 \) with \( c = 1 \), in other words, \( \max(f(n), g(n)) = O(f(n) + g(n)) \).

Similarly, we can see that \( \max(f(n), g(n)) \geq f(n) \) and \( \max(f(n), g(n)) \geq g(n) \) for all \( n \geq n_0 \). Adding these two inequalities, we can see that

\[
2\max(f(n), g(n)) \geq (g(n) + f(n))
\]

, or

\[
\max(f(n), g(n)) \geq \frac{1}{2}(g(n) + f(n))
\]

for all \( n \geq n_0 \). Thus \( \max(f(n), g(n)) = \Omega(g(n) + f(n)) \) with constant \( c = \frac{1}{2} \).

New version of 3.1-1 that you were to solve in this homework

Problem

Let \( f(n) \) and \( g(n) \) be asymptotically nonnegative functions. Using the basic definition of \( \Theta \)-notation, in class we showed that \( \max(f(n), g(n)) = \Theta(f(n) + g(n)) \).
Θ(f(n) + g(n)). Try to prove the same when max is replaced by min. One part is still possible to prove, but the other part is not. Give a clear reason why.

Solution

The functions f(n) and g(n) are asymptotically non negative, there exists n₀ such that f(n) ≥ 0 and g(n) ≥ 0 for all n ≥ n₀. Thus, we have that for all n ≥ n₀, f(n) + g(n) ≥ f(n) ≥ 0 and f(n) + g(n) ≥ g(n) ≥ 0. Adding both inequalities (since the functions are nonnegative), we get f(n) + g(n) ≥ min(f(n), g(n)) for all n ≥ n₀. This proves that min(f(n), g(n)) ≤ c(f(n) + g(n)) for all n ≥ n₀ with c = 1, in other words, min(f(n), g(n)) = O(f(n) + g(n)).

However, it is not possible to show the other way. An easy way to see this is with a counter example. Let f(n) = n and g(n) = n², i.e. min(f(n), g(n)) = n. It is easy to see that for any n₀, c > 0 there always is an n s.t. n < c(n + n²). Thus min(f(n), g(n)) = Ω(f(n) + g(n)) is false.

2 3.1-2

Problem

Show that for any real constants a and b, where b > 0, (n + a)ᵇ = Θ(nᵇ).

Solution

By the definition of Θ(·), we need find the constants c₁, c₂, n₀ such that 0 ≤ c₁nᵇ ≤ (n + a)ᵇ ≤ c₂nᵇ for all n ≥ n₀.

Note that for large values of n, n ≥ |a| we have

n + a ≤ n + |a| ≤ 2n

and for further large values of n, n ≥ 2|a|, (i.e., |a| ≤ ½n)

n + a ≥ n − |a| ≥ ½n

. Thus, when n ≥ 2|a|, we have

0 ≤ ½n ≤ n + a ≤ 2n

. Since b is a positive constant, we can raise the quantities to the bᵗʰ power with out affecting the inequality. thus

0 ≤ (½n)ᵇ ≤ (n + a)ᵇ ≤ (2n)ᵇ

0 ≤ (½)ᵇnᵇ ≤ (n + a)ᵇ ≤ (2)ᵇ(n)ᵇ


Thus, with $c_1 = (\frac{1}{2})^b$, $c_2 = 2^b$, and $n_0 = 2|a|$ we satisfy the definition.

3 3.1-4

Problem
Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

Solution
$2^{n+1} = O(2^n)$, but $2^{2n} \neq O(2^n)$. To show that $2^{n+1} = O(2^n)$, we must find constants $c, n_0 > 0$ such that $0 \leq 2^{n+1}c2^n$ for all $n \geq n_0$. Since $2^{n+1} = 22^n$ for all $n$, we can satisfy the definition with $c = 2$ and $n_0 = 1$.

To show that $2^{2n} \neq O(2^n)$, assume there exist constants $c, n_0 > 0$ such that $0 \leq 2^{2n} \leq c2^n$ for all $n \geq n_0$. Then $2^{2n} = 2^n \times 2^n \leq c2^n \implies 2^n \leq c$. But no constant is greater than $2^n$ for all $n$, and so the assumption leads to a contradiction.

4 3.2

Problem
Indicate, for each pair of expressions (A, B) in the table below, whether A is $O, o, \Omega, \omega, \Theta$ of B. Assume that $k \geq 1, \epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with “yes” or “no” written in each box.

Note that we treat $\lg n$ as a natural logarithm. Dealing with logarithms with different base is simple as they just add a constant factor, for example, $\log_b n = \frac{\log n}{\log b}$.

Solution

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$O$</th>
<th>$o$</th>
<th>$\Omega$</th>
<th>$\omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. $\lg^k n$</td>
<td>$n^\epsilon$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>b. $n^k$</td>
<td>$c^n$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>c. $\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>d. $2^n$</td>
<td>$2^{n/2}$</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>e. $n^{1+\epsilon}$</td>
<td>$c^{\lg n}$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>f. $\lg(n!)$</td>
<td>$\lg(n^n)$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

(a) Apply L'Hospital’s rule repeatedly to see that $\lim_{n \to \infty} \frac{(\lg n)^k}{n^{\epsilon}} = 0$ to con-
clude that \((\log n)^k = o(n^c)\).

\[
\lim_{n \to \infty} \frac{(\log n)^k}{n^c} = \lim_{n \to \infty} \frac{k(\log n)^{k-1} \frac{1}{n}}{c n^{c-1}} \\
= \lim_{n \to \infty} \frac{k(\log n)^{k-1}}{c n^{c-1}} \\
= \lim_{n \to \infty} \frac{k d(\log n)^{k-1}}{c \frac{dn}{dn}} \\
= \lim_{n \to \infty} \frac{k(k-1)(\log n)^{k-2} \frac{1}{n}}{c^2 n^{c-1}}
\]

After \(k\) applications of the rule, we get

\[
\lim_{n \to \infty} \frac{k(k-1)(k-2)\ldots1}{c^k n^c} = 0
\]

(b) Apply L'Hospital’s rule repeatedly to see that \(\lim_{n \to \infty} \frac{n^k}{c^n} = 0\) to conclude that \(n^k = o(c^n)\).

(c) You can visually inspect the plots to see that \(n^{\sin n}\) is an oscillating function. \(\sin n\) oscillates between 1 and \(-1\). When at its maximum value, \(n^{\sin n} > c \sqrt{n}\) and thus \(n^{\sin n} \neq O(\sqrt{n})\). When \(\sin n\) is at its minimum, \(n^{\sin n} < c \sqrt{n}\) and thus \(n^{\sin n} \neq \Omega(\sqrt{n})\).

(d) \(\lim_{n \to \infty} 2^n = \infty\) and therefore \(2^n = \omega(2^{n/2})\).

(e) Recall that \(n^{\lg c} = c^{\lg n}\).

(f) Note \(\lg(n^n) = n \lg(n)\), and using Stirling’s formula it is shown in the text that \(\lg(n!) = \Theta(n \lg(n))\).

5

Problem

Argue that the solution to the recurrence \(T(n) = 1\) if \(n = 1\) and \(8T(n/4) + n^2\) otherwise, where \(c\) is a constant, is \(O(n^3)\).

Solution

We will use the substitution method to prove the problem statement. \(n=1\) works as base case. We make a strong induction assumption, which is for all \(k > 1\)
and $k < n$, $T(n) \leq cn^3$.

$$T(n) = 8T(n/4) + n^2$$
$$\leq 8 * c(n/4)^3 + n^2$$
$$\leq cn^3/8 + n^2$$

We get stuck here, but since the solution is correct, we adjust our guess a little by subtracting a lower order term. i.e. we modify our guess as $T(n) = cn^3 - bn^2$.

$$T(n) = 8T(n/4) + n^2$$
$$\leq 8 * c(n/4)^3 - b(n/4)^2 + n^2$$
$$\leq cn^3/8 - b/2n^2 + n^2$$
$$\leq n^3$$

when we choose $b = 2$. Since the function $cn^3 - bn^2 \in O(n^3)$, we can conclude that $T(n) = O(n^3)$.

6 4.4-6

Problem

Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + cn$, where $c$ is a constant, is $\Omega(n\log n)$ by appealing to a recursion tree.

Solution

We are trying to prove a lower bound to the recurrence. Consider the smallest path of the recursion tree, $n \rightarrow 1/3n \rightarrow (1/3)^2n... \rightarrow 1$. This recursion bottoms out at level $k$ at which $n/3^k = 1$, i.e., $k = \log_3n$. Since each node has two children and each level contributes $cn$, overall contribution from the internal nodes is at least $n\log_3n = \Omega(n\log n)$.

7 Algorithm

Problem

You are given an $n \times n \times n$ array $A(i, j, k)$ of numbers. After $\Theta(n^3)$ preprocessing, show how to compute queries of the following form in $O(1)$ time:

Input: $1 \leq i_1 \leq i_2 \leq n$, $1 \leq j_1 \leq j_2 \leq n$, $1 \leq k_1 \leq k_2 \leq n$
Output: $\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} \sum_{k=k_1}^{k_2} A(i, j, k)$

Hint: We discussed similar problem for 1 and 2 dimensions in class
Solution

For the one dimensional case, the preprocessing involves computing the following sums, \( S(1) = A(1), S(2) = \sum_{i=1}^{2} A(i), \ldots, S(n) = \sum_{i=1}^{n} A(i) \). This can be computed in linear time by observing that \( S(i+1) = S(i) + A(i+1) \). Any query of from \( \sum_{i=i_1}^{i_2} A(i) \) can be computed as \( S(i_2) - S(i_1) \) in constant time.

In case of 2-d case, we compute the sums \( S(r, s) = \sum_{i=1}^{r} \sum_{j=1}^{s} A(i, j) \) in \( O(n^2) \) by noting the recurrence \( S(i+1, j+1) = A(i, j) + S(i+1, j) + S(i, j+1) - S(i, j) \). The computation can be done in \( O(n^2) \) as we spend constant time on each element by making use of previously computed results and there are \( n^2 \) elements to compute. The query of form \( \sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} A(i, j) \) can be computed as \( S(i_2, j_2) - S(i_1 - 1, j_2) - S(i_2, j_1 - 1) + S(i_1 - 1, j_1 - 1) \).

For the 3 dimensional case, compute the sums

\[
S(r, s, t) = \sum_{i=1}^{r} \sum_{j=1}^{s} \sum_{k=1}^{t} A(i, j, k)
\]

which can be done in \( O(n^3) \). This computation can be performed using the following recursion,

\[
S(i+1, j+1, k+1) = A(i+1, j+1, k+1) + S(i, j+1, k+1) + S(i+1, j, k+1) + S(i, j, k+1) - S(i+1, j, k) - S(i, j+1, k) - S(i, j, k+1) + S(i, j, k)
\]

note that when applying this recursion, we observe the following initial conditions. \( S(i, j, k) = 0 \) when atleast one of \( i, j, k \) is zero. For example,

\[
S(1, 1, 1) = A(1, 1, 1) + S(0, 1, 1) + S(1, 0, 1) + S(1, 1, 0) - S(1, 0, 0) - S(0, 1, 0) - S(0, 0, 1) + S(0, 0, 0) = A(1, 1, 1) + 0 + 0 + 0 - 0 - 0 + 0 + 0
\]

Queries of the following form can now be computed in \( O(1) \) time

\[
\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} \sum_{k=k_1}^{k_2} A(i, j, k) = S(i_2, j_2, k_2) - S(i_1 - 1, j_2, k_2) - S(i_2, j_1 - 1, k_2) - S(i_1 - 1, j_2, k_1 - 1) + S(i_1 - 1, j_1 - 1, k_2) + S(i_1 - 1, j_2, k_1 - 1) + S(i_2, j_1 - 1, k_1 - 1) - S(i_1 - 1, j_1 - 1, k_1 - 1).
\]
As an example of how these expressions are derived, consider the 2d case.

\[
\sum_{i=1}^{i_2} \sum_{j=1}^{j_2} A(i, j) = \sum_{i=1}^{i_1-1} \sum_{j=1}^{j_2} A(i, j) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_2} A(i, j)
\]

\[
S(i_2, j_2) = S(i_1-1, j_2) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_2} A(i, j)
\]

\[
= S(i_1-1, j_2) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_1-1} A(i, j) + \sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} A(i, j) \quad \text{last term is the quantity we need}
\]

\[
= S(i_1-1, j_2) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_1-1} A(i, j) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_1-1} A(i, j) + \sum_{i=i_1}^{i_1-1} \sum_{j=1}^{j_1-1} A(i, j) - \sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} A(i, j)
\]

\[
= S(i_1-1, j_2) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_1-1} A(i, j) + \sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} A(i, j) - \sum_{i=i_1}^{i_1-1} \sum_{j=1}^{j_1-1} A(i, j)
\]

\[
= S(i_1-1, j_2) + \sum_{i=i_1}^{i_2} \sum_{j=1}^{j_1-1} A(i, j) + S(i_2, j_1-1) - S(i_1-1, j_1-1)
\]

Rearranging terms we get the desired expression.

\[
\sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} A(i, j) = S(i_2, j_2) - S(i_1-1, j_2) - S(i_2, j_1-1) + S(i_1-1, j_1-1)
\]

The expression for the 3-d problem can be obtained in a similar fashion.