CSE 5311 Homework 1 Solution

Problem 2.2-1

Express the function $n^3/1000 - 100n^2 - 100n + 3$ *in terms of* Θ*-notation*

Answer

$\Theta(n^3)$.

Problem 2.3-3

Use mathematical induction to show that when n *is an exact power of 2, the solution of the recurrence*

$$T(n) = \begin{cases} 2 & \text{if } n = 2, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{for } k > 1. \end{cases}$$

is $T(n) = n \lg n$

Answer

First, establish a function for induction:

$$F(k) = T(2^k) \tag{1}$$

We want to prove that:

$$F(k) = 2^k \lg 2^k \tag{2}$$

Base case:

$$F(1) = T(2) = 2 = 2 \lg 2 = 2^1 \lg 2^1 \tag{3}$$

Prove for $k + 1$:

$$\begin{aligned} F(k+1) = T(2^{k+1}) &= 2T(\frac{2^{k+1}}{2}) + 2^{k+1} && (4) \\ &= 2T(2^k) + 2^{k+1} = 2 \cdot 2^k \lg 2^k + 2^{k+1} && (5) \\ &= 2^{k+1} \lg 2^{k+1} && (6) \end{aligned}$$

Problem 2-1 Insertion sort on small arrays in merge sort

Although merge sort runs in $\Theta(\lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

1. Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.
2. Show how to merge the sublists in $\Theta(n \lg(n/k))$ worst-case time.
3. Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?
4. How should we choose k in practice?

Answer

1. Sorting sublists

Note that sorting each list takes $ak^2 + bk + c$ for some constants a, b and c. We have n/k of those, thus:

$$\frac{n}{k}(ak^2 + bk + c) = ank + bn + \frac{cn}{k} = \Theta(nk)$$

2. Merging sublists

Sorting a sublists of length k each takes:

$$T(a) = \begin{cases} 0 & \text{if } a = 1, \\ 2T(a/2) + ak & \text{if } a = 2^p, \text{if } p > 0. \end{cases}$$

This makes sense, since merging one sublist is trivial and merging a sublists means splitting dividing them in two groups of $a/2$ lists, merging each group recursively and then combining the results in ak steps, since have two arrays, each of length $\frac{a}{2}k$.

Proof by induction:

Base. Simple as ever:

$$T(1) = 1k \lg 1 = k \cdot 0 = 0$$

Step. Assume that $T(a) = ak \lg a$ and we calculate $T(2a)$:

$$T(2a) = 2T(a) + 2ak = 2(T(a) + ak) = 2(ak \lg a + ak) = \tag{7}$$
$$= 2ak(\lg a + 1) = 2ak(\lg a + \lg 2) = 2ak \lg(2a) \tag{8}$$

This proves it. Now if we substitue the number of sublists n/k for a:

$$T(n/k) = \frac{n}{k} k \lg \frac{n}{k} = n \lg(n/k)$$

While this is exact only when n/k is a power of 2, it tells us that the overall time complexity of the merge is $\Theta(n \lg(n/k))$.

3. The largest value of k

The largest value is $k = \lg n$. If we substitute, we get:

$$\Theta(n \lg n + n \lg \frac{n}{\lg n}) = \Theta(n \lg n)$$

If $k = f(n) > \lg n$, the complexity will be $\Theta(nf(n))$, which is larger running time than merge sort.

4. The value of k in practice

In practice, k should be the largest list length on which insertion sort is faster than merge sort.

Problem 3-1 Asymptotic behavior of polynomials

Let

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

where $a_d > 0$, be a degree-d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties.

1. If $k \geq d$, then $p(n) = O(n^k)$.
2. If $k \leq d$, then $p(n) = \Omega(n^k)$.
3. If $k = d$, then $p(n) = \Theta(n^k)$.
4. If $k > d$, then $p(n) = o(n^k)$.
5. If $k < d$, then $p(n) = \omega(n^k)$.

Answer

Let's see that $p(n) = O(n^d)$. We need do pick $c = a_d + b$, such that:

$$\sum_{i=0}^{d} = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_1 n + a_0 \leq c n^d$$

When we divide by n^d, we get:

$$c = a_d + b \geq a_d + \frac{a_{d-1}}{n} + \frac{a_{d-2}}{n^2} + \ldots + \frac{a_0}{n^d}$$

Or:

$$b \geq \frac{a_{d-1}}{n} + \frac{a_{d-2}}{n^2} + \ldots + \frac{a_0}{n^d}$$

If we choose $b = 1$, then we can choose n_0 to be:

$$n_0 = \max(d a_{d-1}, d\sqrt{a_{d-2}}, \ldots, d\sqrt[d]{a_0})$$

Now we have n_0 and c, such that:

$$p(n) \leq c n^d \quad \text{for } n \geq n_0$$

Which is the definition of $O(n^d)$. By chosing $b = -1$ we can prove the $\Omega(n^d)$ inequality and thus the $\Theta(n^d)$ inequality.

It's very similar to prove the other inequalities.

Problem 3-2 Relative asymptotic growths

Indicate for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ 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.

Answer

A	**B**	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ	yes	yes	no	no	no
n^k	c^n	yes	yes	no	no	no
$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
2^n	$2^{n/2}$	no	no	yes	yes	no
$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

Problem 3-6 Iterated functions

We can apply the iteration operator * used in the $\lg^*$ function to any monotonically increasing function $f(n)$ over the reals. For a given constant $c \in \mathbb{R}$, we define the iterated function f_c^* by

$$f_c^*(n) = min\{i \geq 0 : f^{(i)}(n) \leq c\}$$

which need not be well defined in all cases. In other words, the quantity $f_c^*(n)$ is the number of iterated applications of the function f required to reduce its argument down to c or less.

For each of the following functions $f(n)$ and constants c, give as tight a bound as possible on $f_c^*(n)$.

Answer

As shown in the table:

$f(n)$	c	$f_c^*(n)$
$n-1$	0	$\Theta(n)$
$\lg n$	1	$\Theta(\lg^* n)$
$n/2$	1	$\Theta(\lg n)$
$n/2$	2	$\Theta(\lg n)$
$\sqrt{n}$	2	$\Theta(\lg \lg n)$
$\sqrt{n}$	1	does not converge
$n^{1/3}$	2	$\Theta(\log_3 \lg n)$
$n/\lg n$	2	$\omega(\lg \lg n), o(\lg n)$

Problem 4.5-1

Use the master method to give tight asymptotic bounds for the following recurrences:

1. $T(n) = 2T(n/4) + 1$
2. $T(n) = 2T(n/4) + \sqrt{n}$
3. $T(n) = 2T(n/4) + n$
4. $T(n) = 2T(n/4) + n^2$

Answer

1. $\Theta(n^{\log_4 2}) = \Theta(\sqrt{n})$
2. $\Theta(n^{\log_4 2} \lg n) = \Theta(\sqrt{n} \lg n)$
3. $\Theta(n)$
4. $\Theta(n^2)$

Problem 4.5-4

Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Why or why not? Give an asymptotic upper bound for this recurrence.

Answer

With $a = 4, b = 2$, we have $f(n) = n^2 \lg n \neq \mathcal{O}(n^{2-\epsilon}) \neq \Omega(n^{2-\epsilon})$, so no - we cannot apply the master method.

Let's guess $\Theta(n^2 \lg^2 n)$:

$$
\begin{aligned}
T(n) &\leq 4T(n/2) + n^2 \lg n && (9)\\
&\leq 4c(n/2)^2 \lg^2(n/2) + n^2 \lg n && (10)\\
&\leq cn^2 \lg(n/2) \lg n - cn^2 \lg(n/2) \lg 2 + n^2 \lg n && (11)\\
&\leq cn^2 \lg^2 n - cn^2 \lg n \lg 2 - cn^2 \lg(n/2) + n^2 \lg n && (12)\\
&\leq cn^2 \lg^2 n + (1-c)n^2 \lg n - cn^2 \lg(n/2) \qquad (c > 1) && (13)\\
&\leq cn^2 \lg^2 n - cn^2 \lg(n/2) && (14)\\
&\leq cn^2 \lg^2 n && (15)
\end{aligned}
$$

Exercise 4.6-2 is the general case for this.

Problem 4-1 Recurrence examples

Give asymptotic upper and lower bound for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

1. $T(n) = 2T(n/2) + n^4$
2. $T(n) = T(7n/10) + n$
3. $T(n) = 16T(n/4) + n^2$
4. $T(n) = 7T(n/3) + n^2$
5. $T(n) = 7T(n/2) + n^2$
6. $T(n) = 2T(n/4) + \sqrt{n}$
7. $T(n) = T(n-2) + n^2$

Answer

1. $\Theta(n^4)$ (master method)
2. $\Theta(n)$ (master method, $\log_{10/7} 1 = 0$)
3. $\Theta(n^2 \lg n)$ (master method)
4. $\Theta(n^2)$ (master method)

5. $\Theta(n^{\log_2 7})$ (master method)

6. $\Theta(\sqrt{n}\lg_n)$ (master method)

7. $T(n) = n^2 + T(n-2) = n^2 + (n-2)^2 + T(n-4) = \sum_{i=0}^{n/2}(n-2i)^2 = \Theta(n^3)$

Problem 30-3 Multidimensional fast Fourier transform

We can generalize the 1-dimensional discrete Fourier transform defined by equa- tion (30.8) to d dimensions. The input is a d -dimensional array $A = (a_{j_1,j_2,\cdots,j_d})$ *whose dimensions are* $n_1, n_2, \cdots, n_d$ *, where* $n_1 n_2 \cdots n_d = n$*. We define the d-dimensional discrete Fourier transform by the equation*

$$y_{k_1,k_2,\cdots,k_d} = \sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}\cdots\sum_{j_d=0}^{n_d-1} a_{j_1,j_2,\ldots,j_d}\omega_{n_1}^{j_1k_1}\omega_{n_2}^{j_2k_2}\cdots\omega_{n_d}^{j_dk_d} \tag{16}$$

for $0 \le k_1 < n_1, 0 \le k_2 < n_2, \ldots, 0 \le k_d < n_d$.

a. Show that we can compute a d-dimensional DFT by computing 1-dimensional DFTs on each dimension in turn. That is, we first compute n/n_1 separate 1-dimensional DFTs along dimension 1. Then, using the result of the DFTs along dimension 1 as the input, we compute n/n_2 separate 1-dimensional DFTs along dimension 2. Using this result as the input, we compute n/n_3 separate 1-dimensional DFTs along dimension 3, and so on, through dimension d.

b. Show that the ordering of dimensions does not matter, so that we can compute a d-dimensional DFT by computing the 1-dimensional DFTs in any order of the d dimensions.

d. Show that if we compute each 1-dimensional DFT by computing the fast Fourier transform, the total time to compute a d-dimensional DFT is $O(n \lg n)$,independent of d.

Answer

a.

Reorder the summation as follows

$$y_{k_1,k_2,\cdots,k_d} = \sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}\cdots\sum_{j_d=0}^{n_d-1} a_{j_1,j_2,\ldots,j_d}\omega_{n_1}^{j_1k_1}\omega_{n_2}^{j_2k_2}\cdots\omega_{n_d}^{j_dk_d} \tag{17}$$

$$= \sum_{j_d=0}^{n_d-1}\left(\omega_{n_d}^{j_dk_d}\sum_{j_{d-1}=0}^{n_{d-1}-1}\left(\omega_{n_{d-1}}^{j_{d-1}k_{d-1}}\cdots\sum_{j_2=0}^{n_2-1}\left(\omega_{n_2}^{j_2k_2}\sum_{j_1=0}^{n_1-1}\left(a_{j_1,j_2,\ldots,j_d}\omega_{n_1}^{j_1k_1}\right)\right)\right)\right). \tag{18}$$

Now each step of summation is single 1-dimensional DFT.

b.

With the result of **a.** and use the properties of summation and multiplication, it can be proved that any two indexes s and t ($1 \leq s, t \leq d$) can be switch in Equation 18. Hence, prove the hypothesis.

c.

The complexity of FFT on each dimension is

$$T(n, n_1) = (n/n_1) * \Theta(n_1 \lg n_1) = \Theta(n \lg n_1) \tag{19}$$
$$T(n, n_2) = (n/n_2) * \Theta(n_2 \lg n_2) = \Theta(n \lg n_2) \tag{20}$$
$$\cdots \tag{21}$$
$$T(n, n_d) = (n/n_d) * \Theta(n_d \lg n_d) = \Theta(n \lg n_d) \tag{22}$$

The overall complexity is

$$T(n) = \sum_{i=1}^{d} T(n/n_i) \tag{23}$$
$$= \sum_{i=1}^{d} \Theta(n \lg n_i) \tag{24}$$
$$= \Theta(n \lg n_1, n_2, \cdots, n_d) \tag{25}$$
$$= \Theta(n \lg n) \tag{26}$$