COT5405 Analysis of Algorithms
Homework 1 Solution

Summer 2012

Due: June 1, 2012, 11:55pm

Problem 1 *5 Pts*

Prove or disprove:

$$\sum_{i=1}^{n} i^2 \in \Theta(n^2)$$

Answer
Disprove. Since $\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1) = \Theta(n^3) = \Omega(n^2)$

Problem 2 *10 Pts*

List the 16 functions below from lowest asymptotic order to highest asymptotic order. If any two (or more) are of the same asymptotic order, indicate which.

$$n \quad 2^n \quad n\lg n \quad n^3 \quad n^2 \quad \lg n \quad n - n^3 + 7n^5 \quad n^2 + \lg n$$
$$e^n \quad \sqrt{n} \quad 2^{n-1} \quad \lg\lg n \quad \ln n \quad (\lg n)^2 \quad n! \quad n^{1.5}$$

Answer
From low asymptotic order to high asymptotic order
$\lg\lg n$
$\lg n \quad \ln n$
$(\lg n)^2$
$\sqrt{n}$
n
$n\lg n$
$n^{1.5}$
$n^2 \quad n^2 + \lg n$
n^3
$n - n^3 + 7n^5$
$2^{n-1} \quad 2^n$
e^n
$n!$

Problem 3 *15 Pts*

The first n cells of the array E contain integers sorted in increasing order. The remaining cells all contain some very large integer that we may think of as infinity (call it *maxint*). The array may be arbitrarily large (you may think of it as infinite), and *you don't know n*. Give an algorithm to find the position of a given integer x ($x < maxint$) in the array in $O(\log n)$ time.

Answer
We examine selected elements in the array in increasing order until an entry larger than x is found, then do a binary search in the segment that must contain x if it is in the array at all. To keep the number of comparisons in $O(\log n)$, the distance between elements examined in the first phase is doubled at each step. That is, compare x to $E[1]$, $E[2]$, $E[4]$, $E[8]$, ..., $E[2^k]$. We will find an element larger than x (perhaps *maxint*) after at most $\lceil \lg n \rceil + 1$ probes. ($k < \lceil \lg n \rceil$)

If x is found, the search terminates. Otherwise do the Binary Search in the range $E[2^{k-1}+1], \ldots, E[2^k]$ using at most $k-1$ comparisons. Thus the number of comparisons is at most $2k = 2\lceil \lg n \rceil$

Problem 4 *15 Pts*

Suppose c is a constant and $0 < c < \infty$. Solve the following recurrence relations *without* using the master theorem (You may assume $T(1) = 1$.).

(a) $T(n) = 2T(n/2) + cn$.

(b) $T(n) = 3T(n/2) + cn^2$

(c) $T(n) = T(n/2) + c\lg n$.

Answer We assume $T(1) = 1$
(a) $T(n) = 2T(n/2) + cn$
$= 2(2T(n/4) + c(n/2)) + cn$
$= 4T(n/4) + 2cn$
$= 8T(n/8) + 3cn$
$= \ldots$
$= nT(1) + (\lg n)cn$
$= n + cn\lg n$
$\in \Theta(n\lg n)$.

(b) $T(n) = 3T(n/2) + cn^2$
$= 3(3T(n/4) + c(n/2)^2) + cn^2$
$= 3^2T(n/4) + cn^2(1 + 3/4)$
$= 3^3T(n/8) + cn^2(1 + 3/4 + 9/16)$
$= \ldots$
$= 3^{\lg n}T(1) + cn^2(1 + 3/4 + 9/16 + \ldots)$

$= n^{\lg 3} + cn^2(1 + 3/4 + 9/16 + \dots)$
Since the geometric series $1, \frac{3}{4}, \frac{9}{16}, \dots$ has a finite sum, and $\lg 3 < 2$, so $T(n) \in \Theta(n^2)$

(c) $T(n) = T(n/2) + c\lg n$
$= T(n/4) + c\lg(n/2) + c\lg n$
$= T(n/8) + c\lg(n/4) + c\lg(n/2) + c\lg n$
$= \dots$
$= T(1)\lg n + c[\lg n + \lg(n/2) + \lg(n/4) + \cdots + \lg(n/n))$
$= \lg n + c(\lg n + (\lg n - 1) + (\lg n - 2) + \cdots + (\lg n - \lg n)]$
$= \lg n + c[(\lg n)^2 - (1 + 2 + \cdots + \lg n)]$
$= \lg n + c[(\lg n)^2 - \lg n(1 + \lg n)/2]$
$= \lg n + \frac{c}{2}((\lg n)^2 - \lg n)$
$\in \Theta((\lg n)^2)$

Problem 5 *15 Pts*

How many key comparisons are done by (a) Insertion Sort (b) Quick Sort (c) Merge Sort when all the keys are already in order when the sort begins?

Answer
(a) *Insertion Sort*. In each loop the current key just compares to its left neighbour. Since it is larger than the left neighbour so this loop terminates. Only 1 comparison is done in each loop so a total of $n - 1$ comparisons are done.
(b) *Quick Sort*. In each partition we just leave out 1 key (the *pivot*), so the total number of comparisons are $(n-1) + (n-2) + (n-3) + \cdots + 1 = n(n-1)/2$.
(c) *Merge Sort*. If the keys are already sorted, merging the two haves will need only $n/2$ comparisons. So the recurrence relation is:

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \lceil n/2 \rceil, \text{ and } T(1) = 0$$

For simplicity assume n is a power of 2. Then it is easy to see $T(n) = 2T(n/2) + n/2 = (n \lg n)/2$.

Problem 6 *20 Pts*

Each of n elements in an array may have one of the key values *red*, *white* or *blue*. Give a linear time $O(n)$ algorithm for rearranging the elements so that all the *red*s come before all the *white*s, and all the *white*s come before all the *blue*s. (It may happen that there are no elements of one or two of the colors.) The only operations permitted on the elements are examination of a key to find out what color it is, and swap, or interchange, of two elements (specified by their indexes).

Answer
We assume the elements are stored in the range $1, 2, \dots, n$. At an intermediate step the elements are divided into four regions: the first contains only *reds*, the second

contains only *whites*, the third contains elements of unknown color, and the fourth contains only *blues*. There are three indexes, described in the comments below.

```
int r; // index of last red
int u; // index of first unknown
int b; // index of first blue

r = 0; u = 1; b = n + 1;
while (u < b)
if (E[u] == red)
Interchange E[r+1] and E[u].
r ++;
u ++;
else if (E[u] == while)
u ++;
else //E[u] == blue
Interchange E[b-1] and E[u].
b --;
```

While each iteration of *white* loop either u is incremented or b is decremented, so there are n iterations.

Problem 7 *20 Pts*

Suppose we have an unsorted array A of n elements and we want to know if the array contains any duplicate elements.

(a) Outline an efficient method for solving this problem.

(b) What is the asymptotic order of running time of your method in the worst case?

(c) Suppose we know the n elements are integers from the range $1,\ldots,2n$, so other operations besides comparing keys may be done. Give an algorithm for the same problem that is specified to use this information. Tell the asymptotic order of the worst-case running time for this solution. It should be of lower order than your solution for part (a).

Answer
(a) It could be done by first sorting the array A, then doing a post-scan of the sorted array to check for equal adjacent elements.
(b) The time complexity for sorting is $O(n \lg n)$, and the time complexity for scanning the sorted array is $O(n)$. So the overall time complexity is $O(n \lg n)$.
(c) Since the elements are all integers, we could use a count table T with size $2n$. First we initialize the table by setting all of its values to 0. Then we scan the array A. For each elements in A we increase $T[A[i]]$ by 1. Once we meet with an element in count table T with value 2 we can assert that duplicates exist in A. The time complexity is $O(n)$.