CS255 (section 2)
Design and Analysis of Algorithms
San Jose State University

Midterm 2, 09/Apr/2014, duration: 1h15m

Read all questions before starting the midterm. When you're asked to write an algorithm, you should can pseudocode, C, or Java.

(20 points) Question 1.

Consider the following pseudocode for computing the value of a function H(n) which takes as input a non-negative integer n.

```
H($n$)
    if $n == 0$
        return 0
    else
        return $1 + \mathrm{H}(n-1) + \mathrm{H}(\lfloor n/2 \rfloor)$
```

Write new pseudocode for computing H(n) that is asymptotically faster than the pseudocode shown above. Justify your answer.

Answer

The pseudocode given recomputes the same thing over and over again. We can use dynamic programming or memoization, and compute each H(i) once. Here's a dynamic programming solution,

```
BOTTOM-UP-H($n$)
    Let $h[0..n]$ be a new array
    $h[0] = 0$
    for $i = 1$ to $n$
        $h[i] = 1 + h[n-1] + h[\lfloor n/2 \rfloor]$
    return $h[n]$
```

The running time of the new algorithm is $\Theta(n)$ because there's only one for loop that executes n times and each iteration takes constant time. It's not easy to compute the running time of the pseudocode given initially as we didn't cover that type of recursion in class, so I accept as justification if you simply say that the code recomputes the same subproblems H(i) over and over again. (Even though you could use the substitution method to show that the original code is worse than linear, i.e., for any constant $c > 0$, there is an $n_0 > 0$ such that $T(n) > cn$ for all $n \geq n_0$)

(30 points) Question 2.

The number of combinations $\binom{n}{k}$ gives the number of ways of picking k unordered outcomes from n possibilities. This is known as the binomial coefficient and is often read as "n choose k", and can be defined recursively as

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } n = k \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \end{cases}$$

Write a dynamic programming algorithm (or a memoized algorithm) that takes as input two non-negative integers n and k, with $n >= k$, and outputs $\binom{n}{k}$. What's the running time of your algorithm? Justify your answer.

Answer

Here's a memorized version. We create an array $C[0\,..\,n, 0\,..\,k]$ to store the results of subproblems. $C[i,j]$ will be used to store $\binom{i}{j}$.

```
COMB-MEMOIZED($n, k$)
    Let $C[0..n, 0..k]$ be a new array
    **for** $i$ = 0 **to** $n$
        **for** $j$ = 0 **to** $\min(i, k)$
            $C[i,j]$ = UNKNOWN
    **return** COMB($n, k, C$)
```

```
COMB($i, j, C$)
    **if** $C[i,j]$ == UNKNOWN
        **if** $j == 0$ or $j == i$
            $C[i,j] = 1$
        **else**
            $C[i,j]$ = COMB($i-1, j-1, C$) + COMB($i-1, j, C$)
    **return** $C[i,j]$
```

The running time is $\Theta(n * k)$ because there are $\Theta(n * k)$ subproblems and with memoization they are only solved once. Moreover for each subproblem we take constant time.

(20 points) Question 3.

Consider the greedy algorithm for activity selection discussed in class. Recall that in this problem we are given n activities, each with a start time and finish time, and the goal is to find a subset of activities that do not overlap with each other in time, and whose size is as large as possible.

Suppose there is an activity x that does not overlap in time with any other activity. Will activity x always be selected by the greedy algorithm? Justify your answer.

Answer

Yes, it will always be selected (no matter the greedy strategy used).

We can prove it by contradiction. Suppose S is the optimal subset of solutions and x is not in S. Since x doesn't overlap with any other activity, it is never eliminated and therefore it would still be available to be included by the algorithm. Thus, we could get a new subset $S' = S \cup x$ which would have size $|S| + 1$, contradicting the fact that S was optimal.

(30 points) Question 4.

Consider the dynamic table example presented in class when we studied amortized analysis. As opposed to doubling the size of the table whenever it's full, suppose we instead increase the table size by 100 (i.e., if the table has size k, the new size after expanding it will be $k + 100$). All other things regarding the problem are exactly as discusses in class.

With this modification, what would be the worst case running time for a sequence on n insert operations? Justify your answer.

Answer

Let's assume the initial size of the table is 1, like in the example seen in class. We will need to expand the table when we do INSERT(2), INSERT(102), INSERT(202), ..., INSERT($100j+2$). In all those cases we need to copy the elements from the old table into the new table, before inserting the element itself. Let c_i be the cost of i-th INSERT,

$$\begin{aligned} c_i &= \begin{cases} i & \text{, if } (i-2) \text{ is a multiple of 100} \\ 1 & \text{, otherwise} \end{cases} \\ &= \begin{cases} 1+i-1 & \text{, if } (i-2) \text{ is a multiple of 100} \\ 1 & \text{, otherwise} \end{cases} \end{aligned}$$

The cost of a sequence of n INSERT operations is,

$$\begin{aligned} \sum_{i=1}^{n} c_i &= n + \sum_{j=0}^{\lfloor \frac{(n-2)}{100} \rfloor} (100j+1) \\ &= n + 100 \sum_{j=0}^{\lfloor \frac{(n-2)}{100} \rfloor} j + \sum_{j=0}^{\lfloor \frac{(n-2)}{100} \rfloor} 1 \\ &= n + \Theta(n^2) + \Theta(n) \qquad \text{, } \Theta(n^2) \text{ comes from arithmetic progression} \\ &= \Theta(n^2) \end{aligned}$$