CS 581 Homework 1 Solution

January 17th, 2018

Problem 1

Answer true or false:

1. Any problem has a general algorithm that can solve it.

   Answer:

   False. Not all problems can be solved, such the Halting problem or other problems that reduce to it.

2. An (correct) algorithm must terminate.

   Answer:

   True. (Usually when people talk about an algorithm, they should be talking about an "correct algorithm") Correct algorithms must halt with correct output for each input. An algorithm can be defined as a finite list of well-defined instructions for calculating a function. It's finite in the sense that it will terminate, and the total number of instructions executed is finite.

Problem 2

Use induction to prove that for all $n \in \mathbb{N}$,

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Specify your base case, induction hypothesis, induction step, and how you tie them all together.

Proof:
Given proposition:

$$P(n) : \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Base case:

$$P(1) : \sum_{i=1}^{1} i = \frac{1(1+1)}{2}, 1 = \frac{1 \times 2}{2}, 1 = 1$$

So, P(1) is true.

Induction hypothesis:
Assume that $n = k$ for some $k \in \mathbb{N}$, and P(k) is true. Then

$$P(k) : \sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

Induction step:
To show the truth of P(k+1), we have to show that

$$\sum_{i=1}^{k+1} i = \frac{(k+1)((k+1)+1)}{2}$$

Using the induction hypothesis P(k), we get

$$\sum_{i=1}^{k+1} i = \sum_{i=1}^{k} i + (k+1) = \frac{k(k+1)}{2} + (k+1) = \frac{k(k+1)}{2} + \frac{2(k+1)}{2} = \frac{(k+1)(k+2)}{2} = \frac{(k+1)((k+1)+1)}{2}.$$

So, P(k+1) is also true, and we establish that by the Principle of Mathematical Induction, P(n) is true for all $n \in \mathbb{N}$.■

Problem 3

Other than speed, name two other measures of efficiency might one use in a real-world setting.

Answer:
Other two important measures are **space** and **energy**. Space matters because it's limited and it's not free. Energy is also expensive, and for more CPU cycles that programs runs, more energy is used. Other possible answers include network bandwidth usage etc.

Problem 4

Define what it means for a sorting algorithm to be "in-place" and "stable" respectively.

Answer:

In-place sorting algorithm re-arranges the numbers within the array with at most a const number of them stored outside of the array (other const amount of storage) at any time.

Stable sorting algorithm insures that two elements with the same values will remain in the same order (like before sorting) after array is sorted.

Problem 5

Sort array {4, 2, 6, 1, 8, 3, 5, 7} using insertion sort. Show each step.

Answer:

```
{4, 2, 6, 1, 8, 3, 5, 7}
{2, 4, 6, 1, 8, 3, 5, 7} // insert 2
{2, 4, 6, 1, 8, 3, 5, 7} // leave 6 where it is
{1, 2, 4, 6, 8, 3, 5, 7} // insert 1
{1, 2, 4, 6, 8, 3, 5, 7} // leave 8 where it is
{1, 2, 3, 4, 6, 8, 5, 7} // insert 3
{1, 2, 3, 4, 5, 6, 8, 7} // insert 5
{1, 2, 3, 4, 5, 6, 7, 8} // insert 7
```

Problem 6

Sort array {4, 2, 6, 1, 8, 3, 5, 7} using merge sort. Show each step.

Answer:

I assume here that we don't have to show steps when we just divide array into subarrays. I'm showing only merging steps. I also assume that algorithm isn't parallel.

```
{4, 2, 6, 1, 8, 3, 5, 7}
{2, 4, 6, 1, 8, 3, 5, 7} // swap first pair {2, 4}
{2, 4, 1, 6, 8, 3, 5, 7} // swap second pair {6, 1}
{1, 2, 4, 6, 8, 3, 5, 7} // merge first two pairs
{1, 2, 4, 6, 3, 8, 5, 7} // swap third pair {8, 3}
{1, 2, 4, 6, 3, 8, 5, 7} // swap forth pair {5, 7}
{1, 2, 4, 6, 3, 5, 7, 8} // merge second two pairs
{1, 2, 3, 4, 5, 6, 7, 8} // merge two sorted halves
```

Problem 7

We can express insertion sort as a recursive procedure as follows. In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. Write a recurrence for the running time of this recursive version of insertion sort.

Answer: The recurrence equation is the following:

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad \text{n} \leq \text{c}, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases} \tag{1}$$

Divide step:
In our case, time to divide problem to sub problems is constant, each sub problem is 1 element less than previous. So, $D(n) = \Theta(1)$.

Conquer step:
We recursively solve **one** sub problem of size $n-1$, which means $a = 1$ and $n/b = n-1$, and we get $T(n-1)$ in addition to recurrence.

Combine step:
When we return from recursion, first $n-1$ elements are sorted, so we need to insert n^{th} element to sorted part, for that we have to iterate through sorted part until we find a spot, shifting element by element. And it may take up to n elements to do that. So generally speaking, it takes $\Theta(n)$ in worst case. So, $C(n) = \Theta(n)$. (In the average case, $C(n) = \Theta(n/2)$. In the best case, $C(n) = \Theta(1)$. **I assume** that problem is asking about the worst case since it's the most relevant.)

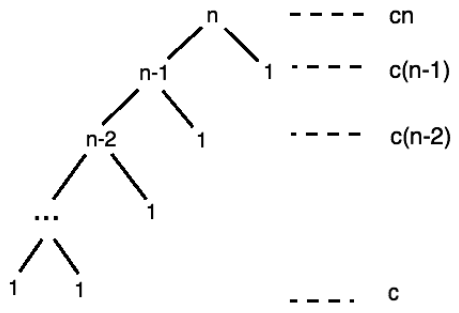
We get that

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad \text{n} = 1, \\ T(n-1) + \Theta(n) & \text{if} \quad \text{n} > 1. \end{cases} \tag{2}$$

And if c is a const time that takes to solve problem with 1 element as well as const time per array element of the divide and combine steps, then we get:

$$T(n) = \begin{cases} c & \text{if} \quad \text{n} = 1, \\ T(n-1) + cn & \text{if} \quad \text{n} > 1. \end{cases} \tag{3}$$

Our recursion tree will have n levels. At the root, when problem just begins, it will take cn to process all elements. At the next level, it will take $c(n-1)$, and etc. Very last level of the recursion tree will have only 1 element. So, it will take $cn + c(n-1) + c(n-2) + ... + c$ to process an array.



From here we get that

$$cn + c(n-1) + c(n-2) + ... + c = c(n + (n-1) + (n-2) + ... + 1) = c\frac{n(n+1)}{2} = c\frac{n^2 + 2n + 1}{2}.$$

Ignoring constants and lower order terms, we get $T(n) = n^2$. In the recursion tree above, leaves (the 1s) should only be at the nth (lowest) level. Leaves correspond to the base recursive case.

Acknowledgement

The solution is contributed by Ksenia Burova. I only made some minor modification.