Selected Problems in CLRS

Section 2

Notifications

Problem Difficulty (count with star)

1. you can solve w/o the brain
2. you can solve if you think a bit
3. you can solve if you think carefully
4. you might solve if you push yourself
5. you can solve if you use other's brain

Exercise

2.1-3 $\star\star\star$ ($\star\star$ if familiar with loop invarient)

Consider the ***searching problem***:

Input: A sequence of n numbers $A = \langle a_1, a_2, ..., a_n \rangle$ and a value v.

Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

Write pseudocode for ***linear search***, which scans through the sequence, looking for v. Using a ***loop invariant***, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

> **Note:** correctness check using loop invariant is one of intersting parts of program verifications. You can varifiy ***simply typed lambda calculaus*** programs (similar to a calculator with loop) using this technique.

2.2-1 $\star$

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

2.2-2a $\star\star$

Consider sorting n numbers stored in array A by first finding the smalleest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the firs $n-1$ elements of A. Write pseudocode for this algorithm, which is known as ***selection sort***.

2.2-2b $\star\star\star\star$ ($\star\star\star$ if familiar with loop invarient)

What loop invariant does this algorithm maintain? Why does it need to run for only the first $n-1$ elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in Θ-notation.

2.2-3 $\star\star\star$

Consider linear search again (see Exercise 2.1-3). How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in , Θ-notation? Justify your answers.

2.3-1 $\star\star$

Using Figure 2.4 as a model(Note: we call it ***merge sort tree***), illlustrate the operation of merge sort on the array $A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.

2.3-2 $\star\star\star$

Rewrite the MERGE procedure so that it does not use sentinels, instead stopping once either array L or R has had all its elements copied back to A and then copying the remainder of the other array back into A .

2.3-3 $\star\star$

Use mathematical induction to solve closed form of $T(n)$ when n is an exact power of 2.

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

2.3-4 $\star\star\star$

In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. Wrtie a recurrence for the running time of recursive version of insertion sort.

2.3-5 $\star\star\star$

Write pseudocode, either iterative or recursive, for ***binary search***. Argue that the worst-case running time of binary search is $\Theta(\lg n)$

2.3-6 $\star\star\star$

Can we use a binary search instead of a linear search, to improve the overall worst-case running time of insertion sort to $\Theta(n \lg n)$

2.3-7 $\star\star\star\star$

Describe a $\Theta(n \lg n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exst two elements in S whose sum is exactly x.