Selected Homework Solutions – Unit 1

CMPSC 465

Exercise 2.1-3

Here, we work with the linear search, specified as follows:

LINEAR-SEARCH(A, v)
Input: $A = <a_1, a, \ldots, a_n>$; and a value v.
Output: index i if there exists an i in $1..n$ s.t. $v = A[i]$; NIL, otherwise.

We can write pseudocode as follows:

```
LINEAR-SEARCH(A, v)
    i = 1
    while i ≤ A.length and A[i] ≠ v          // check elements of array until end or we find key
    {
        i = i + 1
    }

    if i == A.length + 1                     // case that we searched to end of array, didn't find key
        return NIL
    else                                     // case that we found key
        return i
```

Here is a loop invariant for the loop above:

At the start of the ith iteration of the **while** loop, $A[1..i\text{-}1]$ doesn't contain value v

Now we use the loop invariant to do a proof of correctness:

Initialization:
Before the first iteration of the loop, $i = 1$. The subarray $A[1..i\text{-}1]$ is empty, so the loop invariant vacuously holds.

Maintenance:
For $i \in \mathbf{Z}$ s.t. $1 \le i \le A$.length, consider iteration i. By the loop invariant, at the start of iteration i, $A[1..i\text{-}1]$ doesn't contain v. The loop body is only executed when $A[i]$ is not v and we have not exceeded $A.length$. So, when the ith iteration ends, $A[1..i]$ will not contain value v. Put differently, at the start of the $(i+1)^{st}$ iteration, $A[1..i\text{-}1]$ will once again not contain value v.

Termination:
There are two possible ways the loop terminates:

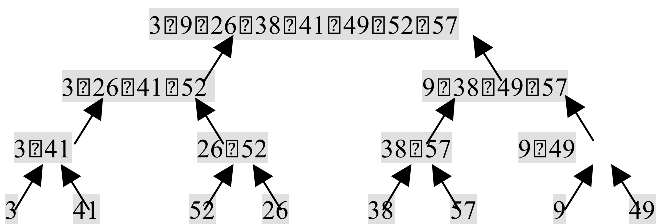
- If there exists an index i such that $A[i] == v$, then the **while** loop will terminate at the end of the ith iteration. The loop invariant says $A[1..i\text{-}1]$ doesn't contain v, which is true. And, in this case, i will not reach $A.length + 1$, so the algorithm returns i s.t. $A[i] = v$, which is correct.
- Otherwise, the loop terminates when $i = n + 1$ (where $n = A.length$), which implies $n = i - 1$. By the loop invariant, $A[1..i\text{-}1]$ is the entire array $A[1..n]$, and it doesn't contain value v, so NIL will correctly be returned by the algorithm.

Note: Remember a few things from intro programming and from Epp:

- Remember to think about which kind of loop to use for a problem. We don't know how many iterations the linear search loop will run until it's done, so we should use an indeterminate loop structure. (If we do, the proof is cleaner.)
- As noted in Epp, the only way to get out of a loop should be by having the loop test fail (or, in the **for** case, the counter reach the end). Don't return or break out of a loop; proving the maintenance step becomes very tricky if you do.

Exercise 2.3-1

The figure below illustrates the operations of the procedure bottom-up of the merge sort on the array
$A = \{3, 41, 52, 26, 38, 57, 9, 49\}$:

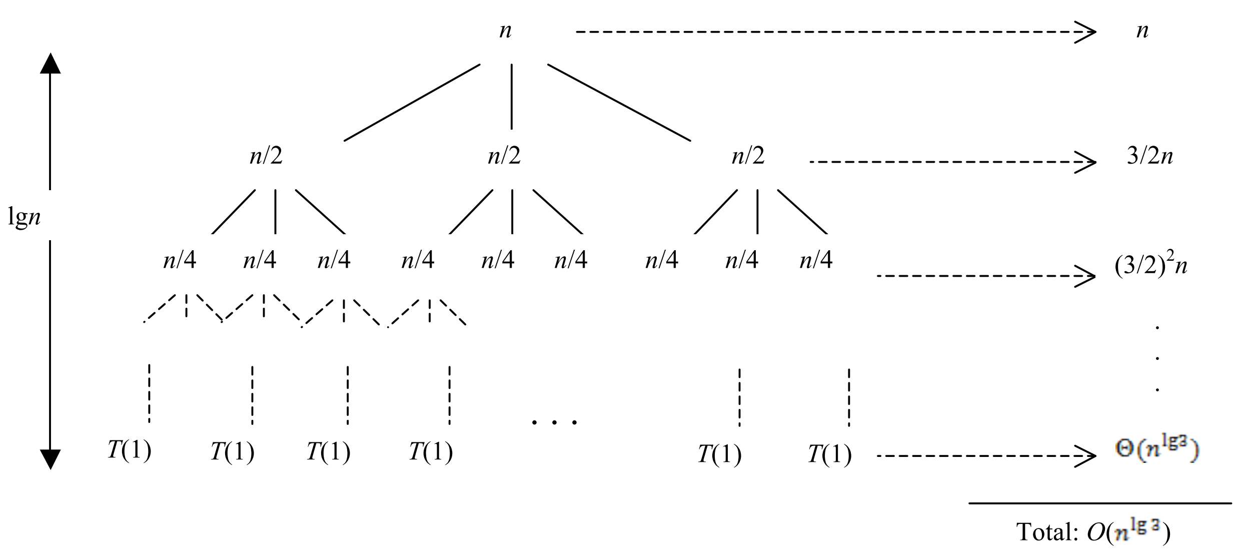


The algorithm consists of merging pairs of 1-item sequence to form sorted sequences of length 2, merging pairs of sequences of length 2 to form sorted sequences of length 4, and so on, until two sequences of length $n/2$ are merged to form the final sorted sequence of length n.

Exercise 4.4-1

The recurrence is $T(n) = 3T(\lfloor n/2 \rfloor) + n$. We use a recurrence tree to determine the asymptotic upper bound on this recurrence.

Because we know that floors and ceilings usually do not matter when solving recurrences, we create a recurrence tree for the recurrence $T(n) = 3T(n/2) + n$. For convenience, we assume that n is an exact power of 2 so that all subproblem sizes are integers.



Because subproblem sizes decrease by a factor of 2 each time when go down one level, we eventually must reach a boundary condition $T(1)$. To determine the depth of the tree, we find that the subproblem size for a node at depth i is $n/2^i$. Thus, the subproblem size hits $n = 1$ when $n/2^i = 1$ or, equivalently, when $i = \lg n$. Thus, the tree has $\lg n + 1$ levels (at depth 0, 1, 2, 3, … , $\lg n$).

Next we determine the cost at each level of the tree. Each level has 3 times more nodes than the level above, and so the number of nodes at depth i is 3^i. Because subproblem sizes reduce by a factor of 2 for each level when go down from the root, each node at depth i, for $i = 0, 1, 2, 3, \ldots, \lg n - 1$, has a cost of $n/2^i$. Multiplying, we see that the total cost over all nodes at depth i, for $i = 0, 1, 2, 3, \ldots, \lg n - 1$, is $3^i * n/2^i = (3/2)^i n$. The bottom level, at depth $\lg n$, has $3^{\lg n} = n^{\lg 3}$ nodes, each contributing cost $T(1)$, for a total cost of $n^{\lg 3} T(1)$, which is $\Theta(n^{\lg 3})$, since we assume that $T(1)$ is a constant.

Now we add up the costs over all levels to determine the cost for the entire tree:

$T(n) = n + \frac{3}{2}n + (\frac{3}{2})^2 n + (\frac{3}{2})^3 n + \cdots + (\frac{3}{2})^{\lg n - 1} n + \Theta(n^{\lg 3})$

$= \sum_{k=0}^{\lg n - 1} (\frac{3}{2})^k n + \Theta(n^{\lg 3})$ — by using Sigma to sum all the elements except the last one

$= \frac{(\frac{3}{2})^{\lg n} - 1}{\frac{3}{2} - 1} n + \Theta(n^{\lg 3})$ — by geometric series sum formula

$= 2n^{\lg 3} - 2n + \Theta(n^{\lg 3})$ — by evaluating the fraction

$= O(n^{\lg 3})$

Thus, we have derived a guess of $T(n) = O(n^{\lg 3})$ for our original recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$. Now we can use the inductive proof to verify that our guess is correct.

To prove:

For all integers n s.t. $n \geq 1$, the property $P(n)$:

The closed form $T(n) \leq dn^{\lg 3} - cn$, for some constants d and c s.t. $d > 0$ and $c > 0$, matches the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$.

Proof:

We will reason with strong induction.

Since we're only proving a bound and not an exact running time, we don't need to worry about a base case.

Inductive Step:

Let $k \in Z$ s.t. $k \geq 1$ and assume $\forall i \in Z$ s.t. $1 \leq i \leq k$, $P(i)$ is true. i.e. $T(i) \leq di^{\lg 3} - cn$ matches the recurrence. [inductive hypothesis]

Consider $T(k+1)$:

$T(k + 1) = 3T(\lfloor (k+1)/2 \rfloor) + k + 1$ — by using the recurrence definition (as $k \geq 1$ implies $k + 1 \geq 2$, so we are in the recursive case)

$\leq 3d(\frac{k+1}{2})^{\lg 3} - \frac{3}{2}c(k+1) + k + 1$ — by subs. from inductive hypothesis, $\lfloor (k+1)/2 \rfloor \leq (k + 1)/2$, and $d \geq c + 1$

$\leq \frac{3}{2^{\lg 3}} d(k+1)^{\lg 3} - \frac{3}{2}c(k+1) + k + 1$ — by laws of exp.

$\leq d(k+1)^{\lg 3} - \frac{3}{2}c(k+1) + k + 1$ — by laws of exp. and log.

$\leq d(k+1)^{\lg 3} - c(k+1)$ — as long as $c \geq 2$, $\frac{3}{2}c(k+1) - (k+1) \geq c(k+1)$

So $P(k + 1)$ is true.

So, by the principle of strong mathematical induction, $P(n)$ is true for all integers n s.t. $n \geq 1$, and constant $d = 3$, $c = 2$.

Exercise 4.5-1

a) Use the master method to give tight asymptotic bounds for the recurrence $T(n) = 2T(n/4) + 1$.

> **Solution:**
> For this recurrence, we have $a = 2$, $b = 4$, $f(n) = 1$, and thus we have that $n^{\log_b a} = n^{\log_4 2}$. Since $f(n) = 1 = O(n^{\log_4 2-\epsilon})$, where $\epsilon = 0.2$, we can apply case 1 of the master theorem and conclude that the solution is $T(n) = \Theta(n^{\log_4 2}) = \Theta(n^{\frac{1}{2}}) = \Theta(\sqrt{n})$.

b) Use the master method to give tight asymptotic bounds for the recurrence $T(n) = 2T(n/4) + \sqrt{n}$.

> **Solution:**
> For this recurrence, we have $a = 2$, $b = 4$, $f(n) = \sqrt{n}$, and thus we have that $n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}} = \sqrt{n}$. Since $f(n) = \Theta(\sqrt{n})$, we can apply case 2 of the master theorem and conclude that the solution is $T(n) = \Theta(\sqrt{n}\lg n)$.

c) Use the master method to give tight asymptotic bounds for the recurrence $T(n) = 2T(n/4) + n$.

> **Solution:**
> For this recurrence, we have $a = 2$, $b = 4$, $f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_4 2}$. Since $f(n) = \Omega(n^{\log_4 2+\epsilon})$, where $\epsilon = 0.2$, we can apply case 3 of the master theorem if we can show that the regularity condition holds for $f(n)$.
>
> To show the regularity condition, we need to prove that $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. If we can prove this, we can conclude that $T(n) = \Theta(f(n))$ by case 3 of the master theorem.
>
> **Proof of the regularity condition:**
> $af(n/b) = 2(n/4) = n/2 \leq cf(n)$ for $c = 0.7$, and $n \geq 2$.
>
> So, we can conclude that the solution is $T(n) = \Theta(f(n)) = \Theta(n)$.

d) Use the master method to give tight asymptotic bounds for the recurrence $T(n) = 2T(n/4) + n^2$.

> **Solution:**
> For this recurrence, we have $a = 2$, $b = 4$, $f(n) = n^2$, and thus we have that $n^{\log_b a} = n^{\log_4 2}$. Since $f(n) = \Omega(n^{\log_4 2+\epsilon})$, where $\epsilon = 1$, we can apply case 3 of the master theorem if we can show that the regularity condition holds for $f(n)$.
>
> **Proof of the regularity condition:**
> $af(n/b) = 2(n/4)^2 = (1/8)\, n^2 \leq c\, f(n)$ for $c = 0.5$, and $n \geq 4$.
>
> So, we can conclude that the solution is $T(n) = \Theta(n^2)$.