CPSC 320 Midterm 2
Monday, November 15th, 2010

[16] 1. Short Answers

[4] a. Your boss asks you to design a divide-and-conquer algorithm to solve a problem whose input is an array of measurements that contain temperature data. What are the two main issues that you need to determine in order to design this algorithm?

Solution: The two issues you need to determine are:

- How to break the problem into smaller subproblems, and
- How to combine the solutions to the smaller subproblems to get the solution to the initial problem.

[4] b. In the Disjoint Sets data structure used by Kruskal's minimum spanning tree algorithm, why does the union operation insert the root with the smaller rank as a child of the root with the larger rank, rather than the other way around?

Solution: If we inserted the root with the larger rank as a child of the other root, the following sequence of n calls to `unions`:

```
T ← new Node
for i ← 1 to n do
  T ← union(T, new Node)
endfor
```

would result in a tree with height $n-1$ (and hence calls to `find` would take O(n) time).

[4] c. Using the method of your choice, give a tight bound on the function $T(n)$ described by the recurrence relation

$$T(n) = \begin{cases} 6T(n/3) + 4n^2 & \text{if } n \geq 13 \\ 17 & \text{if } n \leq 12 \end{cases}$$

Solution: We use the Master theorem. Because $n^{\log_b a} = n^{\log_3 6}$, and $\log_3 6 < 2$, the only case that may apply is case 3. Let us check the regularity condition.

$$af(n/b) = 6f(n/3) = 6 \cdot 4(n/3)^2 = 6 \cdot 4n^2/9 = (2/3)4n^2$$

and hence the regularity condition is satisfied as long as we choose $2/3 < \delta < 1$. Consequently $T(n) \in \Theta(n^2)$.

[4] d. When should we use amortized analysis?

Solution: We should use amortized analysis when we are trying to prove a tight upper bound on the worst-case running time of a *sequence* of operations.

[6] 2. Write a recurrence relation that describes the worst-case running time of the following algorithm as a function of n. Note: I do not believe that this algorithm computes anything useful, so don't waste any time trying to understand what it does.

```
Algorithm Walrus(A, first, n)

  if (n < 4) then
      return n + 1
  endif

  i ← 0
  sum ← 0
  while (i*i < n) do
      sum ← sum + A[i * i]
      i ← i + 1
  endwhile

  j ← 1
  B ← new array
  while (j < n) do
      append A[j] to B
      j ← 2 * j
  endwhile

  x ← Walrus(A, first + n/4, n/2)
  return x * sum * Walrus(B, 0, length[B])
```

Solution : The first `if` statements provides us with the base case ($n < 4$). The first `while` loop executes $1 + \lfloor \sqrt{n} \rfloor$ times. The second `while` loop executes $\lceil \log n \rceil$ times, and constructs an array with $\lceil \log n \rceil$ elements. Finally we have two recursive calls: one on an array with $\lfloor n/2 \rfloor$ elements, and one the array constructed by the second `while` loop.

Putting all of these facts together, and noting that $\log n \in O(\sqrt{n})$, we obtain

$$T(n) = \begin{cases} \Theta(1) & \text{if } n < 4 \\ T(\lfloor n/2 \rfloor) + T(\lceil \log n \rceil) + \Theta(\sqrt{n}) & \text{if } n \geq 4 \end{cases}$$

There was no need to include floors and ceilings in your solution.

[8] 3. A Computer Science researcher has designed a new data structure called a *mélèze* that supports operations `insert`, `findSmallestGap` and `removeElement`, and decides to use amortized analysis to determine the worst-case running time of a sequence of n operations on an initially empty mélèze. She defines a potential function Φ for mélèzes, such that $\Phi(T) \geq 0$ for every mélèze T, and such that $\Phi(T) = 0$ if the mélèze T is empty. After analyzing the running times of the three operations, she has learned that

- An `insert` operation on a mélèze with n elements takes time $\log n$, and increases the mélèze's potential by $\log n$.
- A `findSmallestGap` operation on a mélèze with n elements takes time $x + \log^2 n$, where x is the number of elements examined by the operation. The potential of the mélèze goes down by $x - 2$.
- A `removeElement` operation takes time $\log n$, and increases the mélèze's potential by 3.

Give as tight a bound as possible on the worst-case running time of a sequence of n operations on an initially empty mélèze.

Solution : Recall that for any given operation op_i, we have

$$cost_{am}(op_i) = cost_{real}(op_i) + \Phi(D_i) - \Phi(D_{i-1})$$

where $\Phi(D_i)$ is the potential of the mélèze after i operations have been performed[1]. We start by computing the amortized cost of each operation.

- `insert`: The amortized cost of `insert` is $\log n + \log n = 2 \log n$.
- `findSmallestGap`: The amortized cost of `findSmallestGap` is $x + \log^2 n - (x - 2) = 2 + \log^2 n$.
- `removeElement`: The amortized cost of `removeElement` is $\log n + 3$.

Hence the worst-case running time of a sequence of n operations is

$$\begin{aligned}
\sum_{i=1}^{n} cost_{real}(op_i) &\leq \sum_{i=1}^{n} cost_{am}(op_i) \\
&\leq \sum_{i=1}^{n} \max\{2 \log n, 2 + \log^2 n, 3 + \log n\} \\
&\leq \sum_{i=1}^{n} (2 + \log^2 n) \\
&\leq n(2 + \log^2 n)
\end{aligned}$$

and so it is in $O(n \log^2 n)$.

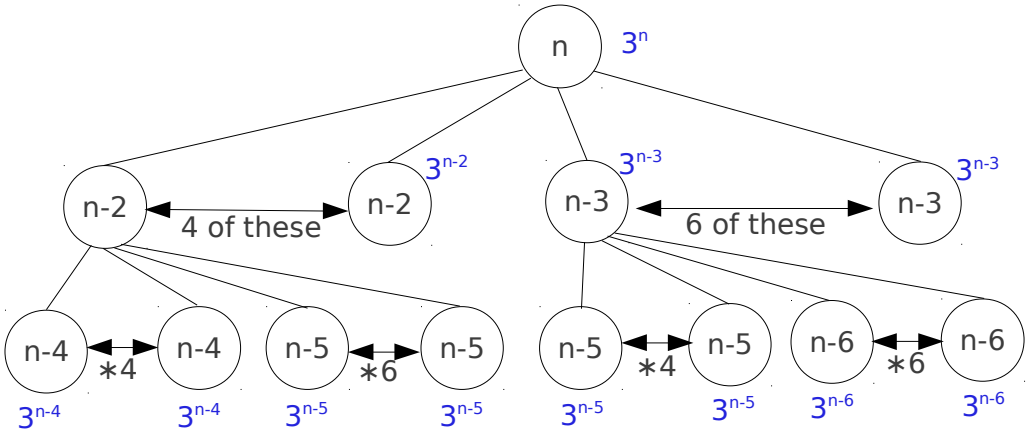
[10] 4. Using recursion trees, prove tight upper and lower bounds on the function $T(n)$ defined by

$$T(n) = \begin{cases} 4T(n-2) + 6T(n-3) + 3^n & \text{if } n \geq 3 \\ 1 & \text{if } n \leq 2 \end{cases}$$

Your grade will depend on the quality of the bounds you provide (that is, showing that $T(n) \in \Omega(1)$ and $T(n) \in O(100^n)$, while true, will not give you many marks).

[1] Mélèze is the French term for larch, a conifer. That is, it's some sort of tree (there's no such data structure as far as I know).

Solution : Here are the first three levels of the recursion tree (some of the repeated nodes have been removed since the tree would not have fit on the page otherwise).



The amount of work done on the first level of the tree (level 0) is 3^n. The amount of work done on level 1 is

$$4 * 3^{n-2} + 6 * 3^{n-3} = 4 * 3^{n-2} + 2 * 3^{n-2} = 6 * 3^{n-2} = 2 * 3^{n-1} = (2/3)3^n.$$

On level 2, we have 16 nodes that each do 3^{n-4} work, 24 nodes that each do 3^{n-5} work, another 24 nodes that each do 3^{n-5} work, and finally 36 nodes that each do 3^{n-6} work. The total amount of work done at that level is therefore

$$\begin{aligned} 16 * 3^{n-4} + 48 * 3^{n-5} + 36 * 3^{n-6} &= 16 * 3^{n-4} + 16 * 3^{n-4} + 4 * 3^{n-4} \\ &= 36 * 3^{n-4} \\ &= (2/3)^2 3^n. \end{aligned}$$

The amount of work done on level i is thus $(2/3)^i 3^n$ for the first $n/3$ levels, and then decreases until the last leaf on level $n/2$. We thus obtain the following upper bound on $T(n)$:

$$\begin{aligned} T(n) &\leq \sum_{i=0}^{n/2-1} (2/3)^i 3^n \\ &\leq \sum_{i=0}^{\infty} (2/3)^i 3^n \\ &= \frac{1}{1-2/3} 3^n \\ &= 3 * 3^n \end{aligned}$$

which means that $T(n) \in O(3^n)$. The top node of the tree perform 3^n work all by itself, and so $T(n) \in \Omega(3^n)$ as well.