Fall 2018, CMPSC 465
Homework Assignment #2 solutions

1

Consider the following divide-and-conquer algorithm.
A is a 1-indexed input array of size $n = 2^k$.
S is the (1-indexed) output array of size n initialized to all zeros.

```
ZigZagSum(A)
    if n = 1 then
        S[1] ← A[1];
        return;
    for i = 1 to n/2 do
        B[i] ← A[2i − 1] + A[2i];
    Recursively compute ZigZagSum(B) and store output in C;
    for i = 1 to n do
        if i is even then
            S[i] ← C[i/2];
        if i == 1 then
            S[1] ← A[1];
        if i is odd and i > 1 then
            S[i] ← C[(i−1)/2] + A[i];
```

- Show the steps of this algorithm for an array with two elements $[5, 6]$, and then show the steps for an array with four elements: $[2, 3, 1, 5]$. What does this algorithm do?
  Solution:
  It is difficult to see what this algorithm does just based on pseudocode inspection. However, if you try out a few examples, you can see that it computes for each i, the cumulative sum of the values $A[1]$ to $A[i]$, storing the result in $S[i]$. This computation is called prefix sums or the scan operation. The divide-and-conquer algorithm given above forms the basis for a parallel algorithm for this problem.
  Applying the algorithm to $[5, 6]$, we have one recursive call ZigZagSum([11]). The final output is $[5, 11]$ after performing the combine steps (the loop after the recursive call).
  Applying the algorithm to $[2, 3, 1, 5]$, we have two recursive calls ZigZagSum([5, 6]) and ZigZagSum([11]). The final output S would be $[2, 5, 6, 11]$. (For full credit, you need to show the steps for each input.)
- Give the recurrence expression for the running time of this algorithm.
  Solution:
  $T(n) = T(n/2) + \Theta(n)$ when $n > 1$ (because one subproblem of size $n/2$ is solved for a problem of size n, and the rest of the computations within each recursive call take $\Theta(n)$ time. For the base case, $T(n) = \Theta(1)$.
- Solve the recurrence equation using the Master theorem. What is the running time?
  Solution:
  Using the Master theorem gives us that $T(n) = \mathrm{O}(n)$.

- Write pseudocode for a non-recursive algorithm that would give the same output as the above recursive algorithm. Your non-recursive algorithm must have the same asymptotic bounds as the recursive approach.
  Solution:
  ZigZagSumIterative(A)

```
n ← A.size;
S[1] ← A[1];
for i = 2 to n do
    S[i] ← S[i − 1] + A[i];
```

2

Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Assuming A is already in memory, describe a method running in $O(n)$ time (not $O(n^2)$ time) for finding the row of A that contains the most 1's.
Solution:
We need to use the facts that all the (i) 1's in a row appear before the 0's, and that (ii) the matrix is already in memory, implying that accessing any memory location $A[i][j]$ is constant time. Start from the first row and access the elements in the first row sequentially, checking to see where the first 0 occurs. After at most n memory references, we will be done with row 1. Let's say the last 1 in row 1 occurs at position i, i.e., row 1 has i 1's. Now, row 1 is a likely candidate for the final answer (the row with the largest number of 1's). Instead of starting at location 1 of row 2, just check the value of $A[2][i]$ now. $A[2][i]$ can either be 0 or 1. If it is 0, then we know for sure that the second row has fewer 1's than the first row. If it is 1, then it means the second row has as many, or more, 1's than the first row. If $A[2][i]$ is a 0, we go to the third row from $A[3][i]$. If $A[2][i]$ is 1, we determine the last occurrence of a 1 in row 2, by starting to access memory locations sequentially from $A[2][i]$. Let the last location be j. Update the final solution to be row 2. Proceed to $A[3][j]$. Observe that we never *move back* if we do this process. We are guaranteed to be done with at most n horizontal moves and $n-1$ vertical moves. Thus, the overall running time is $\Theta(n)$.

3

An degree-n polynomial $p(x)$ is an equation of the form

$$p(x) = \sum_{i=0}^{n} a_i x^i,$$

where x is a real number and each a_i is a constant.
- Describe a simple $O(n^2)$-time method for computing $p(x)$ for a particular value of x.
- Consider now a rewriting of $p(x)$ as $p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)\cdots)))$. Using the Big-Oh notation, characterize the number of multiplications and additions this method of evaluation uses.

Solution:
$p(x)$, when expanded to show elementary operations (+, *), looks like the following:

$$p(x) = a_0 + a_1 * x + a_2 * x * x + a_3 * x * x * x + a_4 * x * x * x * x + \ldots + a_n * x * x * \ldots * x$$

There are n+1 terms in all, and we requires n additions to sum them up. Each term, corresponding to an a_i, requires $i-1$ multiplications. So the total number of multiplications is $0+1+2+\ldots+n = \frac{n(n-1)}{2}$. This would be the simple $\mathrm{O}(n^2)$ algorithm for evaluation.

4

Identify the mistake in the following proof.
Claim: All students in CMPSC 465 received the same grade for HW1.
Proof: We will use induction on n, the number of students, to prove this.
Base case: If there is only one student, there is only one grade.
Inductive step: Assume as induction hypothesis that within any set of $k < n$ students, there is only one grade. Now look at any set of $k = n$ students. Number them $1, 2, 3, ..., n$. Consider the sets $\{1, 2, 3, ..., n-1\}$ and $\{2, 3, 4, ..., n\}$. Each is a set of only $n-1$ students, therefore within each there is only one unique grade. But the two sets overlap, so there must be only one distinct grade for all n students.
Solution:
The flaw is in the base case. We need to show that the claim is true for $k = 2$ first, because we use a fact about the non-zero overlap size of two sets in the inductive step.

5

Sort the following functions according to their asymptotic growth. That is, give an ordering $g_1, g_2, ...$ such that $g_i = \mathrm{O}(g_{i+1})$.

$$n^{\log_2 6}, \quad 70n^2, \quad n\log n, \quad \sum_{i=1}^{n} 3^i, \quad \sqrt{n},$$

Indicate which functions are asymptotically equivalent (*i.e.*, indicate when $g_i = \Theta(g_{i+1})$) by drawing a brace ($\underbrace{\text{like this}}$) under the appropriate boxes.
Solution:

$$\sqrt{n}, \quad n\log n, \quad 70n^2, \quad n^{\log_2 6}, \quad \sum_{i=1}^{n} 3^i$$

(can you reason/prove why?)

6

Use the iterative substitution method to solve the recurrence $T(n) = T(\sqrt{n}) + \Theta(1)$.
Solution:
Let us replace $\Theta(1)$ by a constant $c > 0$ in the recurrence. We then have $T(n) = T(n^{\frac{1}{2}}) + c$. Using

this equation and iteratively replacing n by $n^{\frac{1}{2}}$, we get

$$\begin{aligned}
T(n) &= T(n^{\frac{1}{2}}) + c \\
&= \left(T((n^{\frac{1}{2}})^{\frac{1}{2}}) + c\right) + c \\
&= T(n^{\frac{1}{2^2}}) + 2c \\
&= \left(T((n^{\frac{1}{2}})^{\frac{1}{2^2}}) + c\right) + 2c \\
&= T(n^{\frac{1}{2^3}}) + 3c
\end{aligned}$$

The general form of this equation is

$$T(n) = T(n^{\frac{1}{2^k}}) + kc$$

We want to keep iterating until the base case. If we use $T(1)$ as the base case, we need $n^{\frac{1}{2^k}}$ to be 1. This however gives us that $k = \infty$. Instead, let us set $n^{\frac{1}{2^k}}$ (the base case) to a small constant $b = \mathrm{O}(1)$.

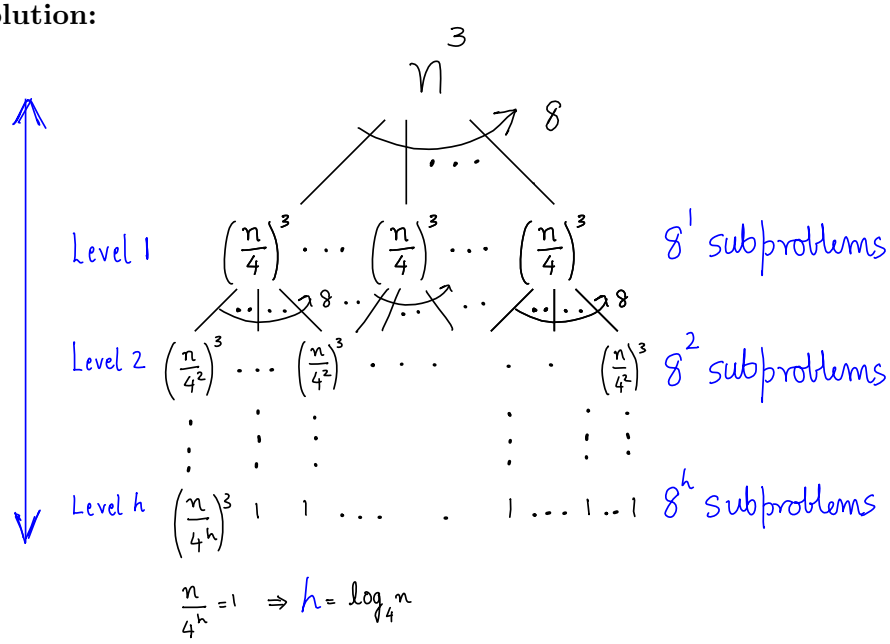
$n^{\frac{1}{2^k}} = b$. Taking log on both sides, we get $\frac{1}{2^k}\log n = \log b$, or $2^k = \frac{\log n}{\log b}$, or $k = \log\left(\frac{\log n}{\log b}\right)$. This gives us that $k = \Theta(\log\log n)$.

Substituting $k = \Theta(\log\log n)$ in the simplified equation for $T(n)$ gives us that $T(n) = \Theta(\log\log n)$.

7

Solve the recurrence equation $T(n) = 8T(n/4) + n^3$ using the recursion tree construction method. Express your answer in the Θ-notation. In the recursion tree method, you (i) draw a recursion tree, (ii) determine its height, (iii) estimate the work associated with nodes at each level, and (iv) simplify the summation to come up with a closed-form expression for the running time. Assume that $T(1) = \Theta(1)$.

Solution:



The work associated with level i is $8^i(\frac{n}{4^i})^3$.

$$T(n) = \sum_{i=0}^{\log_4 n} 8^i \left(\frac{n}{4^i}\right)^3 = n^3 \sum_{i=0}^{\log_4 n} \left(\frac{1}{8}\right)^i = \frac{8n^3}{7}(1 - (\frac{1}{8})^{\log_4 n+1}) = \frac{8n^3}{7} - \frac{n^{1.5}}{7} = \Theta(n^3)$$