COMP 340 **Analysis of Algorithms**

Problem Set 2

Assigned: February 2, 2018 *Due: February 16, 2018*

1. (Modified ADPS Problem 4-1) Find the closed form for the following recurrence relations by using the recursion tree method. In other words, draw the recursion tree, determine the height of the tree, compute the cost at each level, and sum the cost across the levels. Assume $O(1)$ work and span at the base case, i.e. at the leaves.

   (a) $W(n) = 3W(n/2) + n$

   (b) $W(n) = 2W(\sqrt{n}) + \log n$

   (c) $S(n) = \max(S(n/3), S(n/4)) + \log n$

   (d) $W(n, m) = W(n/2, m/4) + n^2 m$
   Though we have not looked at such recurrences in class, it is not all that different to solve this by drawing a recursion tree. There are many algorithms that have two different inputs with two different sizes n and m. These algorithms come up often in string matching, which is itself essential to bioinformatics.

2. (Modified ADPS Problem 4-1) Find the closed form for the following recurrence relations using the Master theorem.

   (a) $W(n) = 3W(n/2) + n$

   (b) $S(n) = 2S(n/4) + n$

   (c) $W(n) = 2W(n/4) + n^2$

   (d) $W(n) = 2W(n/2) + n \log n$

3. (ADPS Problem 4-5) In class I briefly mentioned a design technique called **reduction**[1], in which you solve a problem by seeing that it is actually some other (solved) problem expressed in a different way. You use the algorithm for the solved problem, and perhaps a few other simple steps, to get the solution to the original problem.

   In the order statistics problem you are given a sequence of n (unsorted) values and an integer $k \in [1, n]$ and must return the kth-smallest value. Give an algorithm for the order statistics problem by reduction to sorting.

4. TADM Exercise 4-1.

5. TADM Exercise 4-5. For the purposes of this problem set, you only need to provide a sequential algorithm. However, I suggest to you think about how to parallelize it in order to study for the next quiz.

6. TADM Exercise 4-8.

7. Knapsack variant, attempt #2. Consider the following knapsack variants:

   (a) Given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$ and a positive integer k, return a subset whose sum is *at least* k, or the empty set if no such set exists.

[1] Not to be confused with the `reduce` algorithm.

(b) Given a set of integer $S = \{s_1, s_2, \ldots, s_n\}$ and a positive integer k, return a subset that sums to the highest possible value $\leq k$.

You are given an algorithm A that solves the first variant in time $T_A(n)$. Use this algorithm as a black box[2] to solve the second variant in time $O(T_A(n) \cdot \log k)$. Analyze its running time and briefly explain (a formal proof is not necessary) its correctness.

8. In class we analyzed parallel mergesort with a sequential merge step. Now let's improve that.

   (a) (ADPS Problem 8-1) Give a parallel algorithm for merging two sorted sequences L and R. If the sequences have length m and n, respectively, then your algorithm should perform $O(n+m)$ work in $O(\log(n+m))$ span.
   Hint: Assume $m \geq n$. Once you find the midpoint mid_L of L, find a way to separate R into elements that are smaller than mid_L and those that are larger. Then divide and conquer. The analysis is a bit harder: show that the size of the first subproblem is k where $\frac{n+m}{4} \leq k \leq \frac{3(n+m)}{4}$.

   (b) Replace the sequential merge step in our mergesort with your algorithm. If you could not come up with a parallel merge algorithm, assume you have one with work $O(n)$ and span $O(\log^2 n)$. Analyze the new mergesort's work and span.

9. The following questions involve the "Stock market" problem, which comes from a parallel algorithms course at WUSTL[3]:

   > The problem with the stock market is that, while it is possible to make a great deal of money buying and selling stocks, it's easy to lose even more. The long-standing—if somewhat unhelpful—maxim to make more money than you lose is "buy low, sell high."
   >
   > The stock market problem is finding the best opportunity to follow this advice: for any sequence of integer prices, where the index in the sequence represents time, find maximum jump from an earlier price to a later price. For example, if the sequence of prices was
   >
   > $$\langle 40, 20, 0, 0, 0, 1, 3, 3, 0, 0, 9, 21 \rangle$$
   >
   > then the maximum jump is 21, which happens between the price at time 2 and time 11. More formally, the stock market problem is to compute
   >
   > $$\max\{s_j - s_i \mid 0 \leq i \leq j < |s|\}$$
   >
   > Note that this maximum is only well defined if there is at least one element in s.

   (a) Give a (sequential) brute force algorithm for the Stock market problem. Analyze the runtime.

   (b) Give a parallel divide and conquer algorithm for this problem.

   (c) Analyze the runtime of your divide and conquer algorithm.

   (d) Prove your divide and conquer algorithm is correct.

   (e) (Bonus) Implement both algorithms in Cilk Plus and compare the running times. If you have finished the rest of the problems (and ONLY then) and are serious about this, let me know and I can provide you with some starter code and inputs on which to test .

[2] In other words, you may call a procedure/function $A(S, \ell)$ that runs algorithm A on S with $k = \ell$. You may call A as many times as you wish (provided you satisfy the time bound) and choose ℓ each time.

[3] Washington University in St. Louis