15–210: Parallel and Sequential Data Structures and Algorithms

Practice Exam I (Solutions)

February 2014

- There are 13 pages in this examination, comprising 8 questions worth a total of 125 points. The last 2 pages are an appendix with costs of sequence, set and table operations.
- You have 80 minutes to complete this examination.
- Please answer all questions in the space provided with the question. Clearly indicate your answers.
- You may refer to your one double-sided $8\frac{1}{2} \times 11$in sheet of paper with notes, but to no other person or source, during the examination.
- Your answers for this exam must be written in blue or black ink.

Full Name: Edsger W. Dijkstra

Andrew ID: _____ Section: _____

Question	Points	Score
Recurrences	16	
Short Answers	18	
Missing Element	12	
Interval Containment	13	
Quicksort	17	
Maximum Elements	17	
Higher Order Costs	16	
Parentheses Revisited	16	
Total:	125	

Question 1: Recurrences (16 points)

Recall that $f(n)$ is $\Theta(g(n))$ if $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$. Give a closed-form solution in terms of Θ for the following recurrences. Also, state whether the recurrence is dominated at the root, the leaves, or equally at all levels of the recurrence tree.

You do not have to show your work, but it might help you get partial credit.

(a) (4 points) $f(n) = 5f(n/5) + \Theta(n)$

Solution: $\Theta(n \lg n)$, balanced.

(b) (4 points) $f(n) = 3f(n/2) + \Theta(n^2)$

Solution: $\Theta(n^2)$, root-dominated.

(c) (4 points) $f(n) = f(n/2) + \Theta(\lg n)$

Solution: $\Theta(\lg^2 n)$, approximately balanced.

(d) (4 points) $f(n) = 5f(n/8) + \Theta(n^{2/3})$

Solution: $\Theta(n^{\lg_8 5})$ (roughly $\Theta(n^{0.77})$) leaf-dominated.

Question 2: Short Answers (18 points)

(a) (5 points) Assume you are given an associative function $f(a, b)$: `int seq` $\times$ `int seq` $\rightarrow$ `int seq` which takes two sequences of length n_1 and n_2 returning a sequence of length $n_1 + n_2$. It does $O((n_1 + n_2)^2)$ work and $O(\log(n_1 + n_2))$ span. What is the work and span of the following function?

```
fun foo(S : int seq) =
   Seq.reduce f Seq.empty (Seq.map Seq.singleton S)
```

Solution: $W = \Theta(n^2)$, $S = \Theta(\log^2 n)$

(b) (5 points) Implement `reduce` using contraction. You can assume the input length is a power of 2.

Solution:

```
fun reduce f b s =
     case length s
      of 0 => b
       | 1 => f(b, nth s 0)
       | n =>
         let
           val x = tabulate
           (fn i => case i = (n div 2) of
                    true => (nth s (2*i))
                     | _ => f(nth s (2*i), nth s (2*i + 1)))
                       (((n-1) div 2)+1)
         in reduce f b x
         end
```

(c) **Guessing Games** I am thinking of a random non-negative integer, X. Of course, I can't mean *uniformly* random, as that would mean that at least half the time I'm thinking of an infinite integer! As it turns out, the expected value of positive integers I think of is 1000.

i. (4 points) For some reason, I like to choose 15210 a lot. What's the maximum probability which which I can choose $X = 15210$ (while still obeying the condition $\mathbf{E}[X] = 100$)?

Solution: From Markov's inequality,

$$\mathbf{Pr}\left[X \geq 15210\right] \leq \frac{E[X]}{15210}$$

So, $\mathbf{Pr}\left[X = 15210\right] \leq 1000/15210$. Equivalently, you could assume that I only think of the numbers 0 and 15210, and solve from there.

ii. (4 points) I've modified my preferences in random numbers such that I generally choose numbers close to 1000; the variance of X is 20. So, it's still true that $\mathbf{E}[X] = 1000$, but now also $\mathrm{Var}(X) = \mathbf{E}[(X - \mathbf{E}[X])^2] = 20$. What's the maximum probability which which I can choose $X = 15210$ now?

Solution: From Chebyshev's inequality,

$$\mathbf{Pr}\left[|X - \mathbf{E}(X)| \geq 14210\right] \leq \frac{\mathrm{Var}(X)}{14210^2}$$

So, $\mathbf{Pr}\left[X = 15210\right] \leq \mathbf{Pr}\left[|15210 - 1000| \geq 14210\right] \leq 20/14210^2$.

Question 3: Missing Element (12 points)

For 15210, there is a roster of n **unique** Andrew ID's, each a string of at most 9 characters long (so `String.compare` costs $O(1)$).

In this problem, the roster is given as a **sorted** string sequence R of length n. Additionally, you are given another sequence S of $n-1$ **unique ID's from** R. The sequence S is **not necessarily sorted**. Your task is to design and code a divide-and-conquer algorithm to find the missing ID.

(a) (7 points) Write an algorithm in SML that has $O(n)$ work and $O(\log^2 n)$ span.

```
open ArraySequence
fun missing_elt(R: string seq, S: string seq) : string =
let fun lessThan a b = (String.compare(b, a)=LESS) (* is b<a? *)
in
   case (length R)
    of 0 => raise Fail "should not get here"
     | 1 => nth R 0
     | n =>
       let val p = nth R (n div 2)
           val Sleft = filter (lessThan p) S
           val Sright = filter (not o (lessThan p)) S
           val Rleft = take (R, n div 2)
           val Rright = drop (R, n div 2)
       in if (length Sleft < length Rleft) then
              missing_elt (Rleft, Sleft)
          else
              missing_elt (Rright, Sright)
       end
end
```

(b) (5 points) Give a brief justification of why your algorithm meets the cost bounds.

Solution: We maintain the variant that $|R| = |S| + 1$. The body of the function contains only `filter`, `take`, and `drop`, which have $\Theta(|R|)$ work and $\Theta(\log |R|)$ span. Furthermore, the algorithm makes only one recursive call on the problem of size $|R|/2$, so we have $W(n) = W(n/2) + \Theta(n)$ and $S(n) = S(n/2) + \Theta(\log n)$. These recurrences solve to $W(n) = \Theta(n)$ and $S(n) = \Theta(\log^2 n)$.

Question 4: Interval Containment (13 points)

An interval is a pair of integers (a, b). An interval (a, b) is contained in another interval (α, β) if $\alpha < a$ and $b < \beta$. In this problem, you will design an algorithm

```
count:  (int * int) seq → int
```

which takes a sequence of intervals (i.e., ordered pairs) $(a_0, b_0), (a_1, b_1), \ldots, (a_{n-1}, b_{n-1})$ and computes the number of intervals that are contained in some other interval. If an interval is contained in multiple intervals, it is counted only once.

For example, `count` $\langle(0, 6), (1, 2), (3, 5)\rangle = 2$ and `count` $\langle(1, 5), (2, 7), (3, 4)\rangle = 1$. Notice that the interval $(3, 4)$ is contained in both $(1, 5)$ and $(2, 7)$, but the count is 1.

You can assume that the input to your algorithm is sorted in increasing order of the first coordinate and that all the coordinates (the a_i's and b_i's) are distinct.

(a) (5 points) Give a brute force solution to this problem (code or prose).

Solution:

```
open ArraySequence

fun count s =
  let fun or (p,q) = p orelse q
      fun inOther (a,b) =
        reduce or false (map (fn (x,y) => (x < a) andalso (b < y)) s)
  in reduce (op+) 0 (map (fn iv => if inOther iv then 1 else 0) s)
  end
```

(b) (8 points) Design an algorithm that has $O(n)$ work and $O(\log n)$ span. Carefully explain your algorithm; you don't have to write code. Hint: The algorithm is short.

Solution:

```
open ArraySequence

fun count (s : (int*int) seq) =
  let val ends = map (fn (_,b) => b) s
      val (maxCovered, _) = scan Int.max (Option.valOf Int.minInt) ends
      fun inOther ((a,b), covered) =
          if b < covered then 1 else 0

  in reduce (op+) 0 (map2 inOther s maxCovered)
  end
```

Question 5: Quicksort (17 points)

Assume throughout that all keys are distinct.

(a) (3 points) TRUE or FALSE. In randomized quicksort, each key is involved in the same number of comparisons.

Solution: FALSE

(b) (7 points) What is the probability that in randomized quicksort, a random pivot selection on an input of n keys leads to recursive calls, both of which are no smaller than $\frac{n}{16}$? Show your work.

Solution: $\frac{7}{8}$

(c) (7 points) Consider running randomized quicksort on a permutation of $1, \ldots, n$. What is the probability that a quicksort call tree has height exactly n? Note: the height of a tree is the number of nodes on its longest path.

Solution: This happens only when we pick the maximum or the minimum element in the input repeatedly. The probability of that is:

$$\frac{2}{n} \times \frac{2}{n-1} \times \cdots \times \frac{2}{2} = \frac{2^{n-1}}{n!}$$

Question 6: Maximum Elements (17 points)

Recall the `max2` problem from lecture: this problem is to find the two largest elements in a sequence of n unique numbers. Here's the function given in the lecture notes:

```
fun max2 S =
    let
      fun replace ((m1,m2), v) =
          if v <= m2 then (m1, m2)
          else if v <= m1 then (m1, v)
          else (v, m1)
      val (s0, s1) = (nth S 0, nth S 1)
      val start = if s0 >= s1
                  then (s0, s1)
                  else (s1, s0)
    in
      iter replace start (drop (S, 2))
    end
```

(a) (5 points) Write SML or pseudocode for the function `max3`, which is like `max2`, except it finds the **three** largest elements in the sequence.

```
fun max3 S =
    let
      fun replace ((m1,m2,m3), v) =
          if v <= m3 then (m1, m2, m3)
          else if v <= m1
               then if v <= m2
                    then (m1, m2, v)
                    else (m1, v, m2)
               else (v, m1, m2)
      val (s0, s1, s2) = (nth S 0, nth S 1, nth S 2)
      val start = if s0 > s1
                  then if s1 > s2
                       then (s0, s1, s2)
                       else if s0 > s2
                            then (s0, s2, s1)
                            else (s2, s0, s1)
                  else if s0 > s2
                       then (s1, s0, s2)
                       else if s1 > s2
                            then (s1, s2, s0)
                            else (s2, s1, s0)
    in
      iter replace start (drop (S, 3))
    end
```

(b) (7 points) Assuming the input ordering is randomized (all permutations are equally likely), write an expression for the *exact* number of comparisons `max3` does *in expectation*. State your answer in terms of n, the number of elements in the sequence.

Solution: $(n-3) + (2 + \frac{2}{3}) + \sum_{i=4}^{n} \frac{3}{i} + \frac{2}{i}$

(c) (5 points) Simplify any sums in your answer to part (b) above. You may state bounds that are tight within a constant additive term.

Solution: $n + 5 \ln n$

Question 7: Higher Order Costs (16 points)

For full credit, show your work.

(a) (8 points) Give closed forms in terms of Θ for the work and span of the function f assuming the sequence s contains n sequences of m elements each and b contains m elements.

```
fun zipPlus (s1: int seq, s2: int seq) = map2 op+ s1 s2
fun f (b : int seq) (s : int seq seq) = reduce zipPlus b s
```

$W_f(n, m) =$

Solution: The work for `zipPlus` is linear in the number of elements. The work at the leaves of the reduction tree is $\frac{n}{2}\Theta(m)$. At each level the work decreases by a factor of 2. Thus, $W_f(n, m) = \Theta(nm)$. Alternatively, using a divide and conquer view of reduce $W_f(n, m) = 2W_f(n/2, m) + \Theta(m) = \Theta(nm)$, since the work is leaf dominated with $n/2$ pairs of leaves, each with $\Theta(m)$ work.

$S_f(n, m) =$

Solution: The span for `zipPlus` is $\Theta(1)$. Since the reduction tree has depth $\lg n$, $S_f(n, m) = \Theta(\lg n)$

(b) (8 points) Give closed forms in terms of Θ for the work and span of the following function g assuming the sequence s contains n sets of m elements each. Assume that all elements are unique accoss all sets and that element comparison is $O(1)$.

```
fun g (s : Set.set seq) = reduce Set.union (Set.empty ()) s
```

$W_g(n, m) =$

Solution: Since each union is with sets of the same size, the work is linear in the number of elements. The size of the union set can double in size and the number of applications of union halves at each level, the total work at each level is $\Theta(nm)$. Again, there are $\lg n$ levels. Thus $W_g(n, m) = \Theta(nm \lg n)$.

$S_g(n, m) =$

Solution: The span of union is the logarithm of the number of elements. Since the number of elements doubles each level above the leaves and for $\lg n$ levels, the total span $\sum_{i=1}^{\lg n} \lg(2^i m)$, which yields $S_g(n, m) = \Theta(\lg n \lg(nm))$.

Question 8: Parentheses Revisited (16 points)

A parenthesis expression is called *immediately paired* if it consists of a sequence of open-close parentheses — that is, of the form "()()()() ... ()".

(a) (8 points) **Longest immediately paired subsequence (LIPS) problem.** Given a (not necessarily matched) parenthesis sequence s, the longest immediately paired subsequence problem requires finding a (possibly non-contiguous) longest subsequence of s that is immediately paired. For example, the LIPS of "(((((((()()()))))()((((()(()))" is "()()()()()()" as highlighted in the original sequence.

Write a function that computes the *length* of a LIPS for a given sequence. Your function should have $O(n)$ work and $O(\lg n)$ span.

(**Hint:** Try to find a property that simplifies computing LIPS. This problem might be difficult to solve otherwise.)

```
fun findLIPS (s: paren seq) : int = (* Work = O(n), Span = O(lg n) *)
```

Solution: The algorithm simply extracts immediately paired parentheses and counts them. We prove below why this is sufficient.

```
fun findLIPS (s: paren seq) =
    let
      fun isIP i =
          case (nth s i, nth s (i+1))
            of (LPAREN, RPAREN) => 2
             | _ => 0
    in
      reduce op+ 0 (tabulate isIP (length s - 1))
    end
```

(b) (8 points) Prove succintly that your algorithm correctly computes LIPS.

Solution: Consider any parenthesis expression and let () be an immediately paired parenthesis in the result. Let i and j be the positions of the parenthesis in the original sequence. Note that $i < j$. Let k be the leftmost RPAREN and note that $i < k \leq j$ and the parenthesis at $k-1$ and k are immediately paired. In other words, there exists one immediately paired parentheses in the contiguous subsequence defined by i and j, e.g., "(....()....)", "(....()", "()...)". It thus suffices to count the immediately paired parenthesis in the input.

Appendix: Library Functions

ArraySequence	Work	Span

`empty ()`
`singleton a`
`length s`
`nth s i`
	$O(1)$	$O(1)$

`tabulate f n`
if `f i` has W_i work and S_i span
`map f s`
if `f` s_i has W_i work and S_i span, and $\|s\| = n$
`map2 f s t`
if `f` (s_i, t_i) has W_i work and S_i span, and $\|s\| = n$
	$O\left(\sum_{i=0}^{n-1} W_i\right)$	$O\left(\max_{i=0}^{n-1} S_i\right)$

`reduce f b s`
if `f` does constant work and $\|s\| = n$
`scan f b s`
if `f` does constant work and $\|s\| = n$
`filter p s`
if `p` does constant work and $\|s\| = n$
`showt s`
if $\|s\| = n$
`hidet tv`
if the combined length of the sequences is n
	$O(n)$	$O(\lg n)$

`sort cmp s` *if* `cmp` does constant work and $\|s\| = n$	$O(n \lg n)$	$O(\lg^2 n)$

`merge cmp s t`
if `cmp` does constant work, $\|s\| = n$, and $\|t\| = m$
`flatten s`
if if $s = \langle s_1, s_2, \ldots, s_k \rangle$ and $m + n = \sum_i \|s_i\|$
	$O(m + n)$	$O(\lg(m + n))$

`append (s,t)` *if* $\|s\| = n$, and $\|t\| = m$	$O(m + n)$	$O(1)$

Table/Set Operations	*Work*	*Span*
`size`(T) `singleton`(k, v)	$O(1)$	$O(1)$
`filter` f T	$O\Big(\sum_{(k,v)\in T} W(f(v))\Big)$	$O\Big(\lg\|T\| + \max_{(k,v)\in T} S(f(v))\Big)$
`map` f T	$O\Big(\sum_{(k,v)\in T} W(f(v))\Big)$	$O\Big(\max_{(k,v)\in T} S(f(v))\Big)$
`tabulate` f S	$O\Big(\sum_{k\in S} W(f(k))\Big)$	$O\Big(\max_{k\in S} S(f(k))\Big)$
`find` T k `insert` f (k, v) T `delete` k T	$O(\lg\|T\|)$	$O(\lg\|T\|)$
`extract` (T_1, T_2) `merge` f T_1 T_2 `erase` (T_1, T_2)	$O\big(m \lg(\frac{n+m}{m})\big)$	$O\big(\lg(n+m)\big)$
`domain` T `range` T `toSeq` T	$O(\|T\|)$	$O(\lg\|T\|)$
`collect` S `fromSeq` S	$O(\|S\| \lg\|S\|)$	$O(\lg^2\|S\|)$
`intersection` (S_1, S_2) `union` (S_1, S_2) `difference` (S_1, S_2)	$O\big(m \lg(\frac{n+m}{m})\big)$	$O\big(\lg(n+m)\big)$

where $n = \max(|T_1|, |T_2|)$ and $m = \min(|T_1|, |T_2|)$. For `reduce` you can assume the cost is the same as `Seq.reduce f init (range(T))`. In particular `Seq.reduce` defines a balanced tree over the sequence, and `Table.reduce` will also use a balanced tree. For `merge` and `insert` the bounds assume the merging function has constant work.