15–210: Parallel and Sequential Data Structures and Algorithms

Practice Exam I

February 2013

- There are 11 pages in this examination, comprising 6 questions worth a total of 110 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: ______________________________

Andrew ID: ________________ Section: ________________

Question	Points	Score
Recurrences	20	
Short Answers	20	
Missing Element	15	
Priority Queues	20	
Strongly Connected Component	20	
Interval Containment	15	
Total:	110	

Question 1: Recurrences (20 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) (5 points) $f(n) = 5f(n/5) + \Theta(n)$.

(b) (5 points) $f(n) = 3f(n/2) + \Theta(n^2)$.

(c) (5 points) $f(n) = f(n/2) + \Theta(\lg n)$.

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

Question 2: Short Answers (20 points)

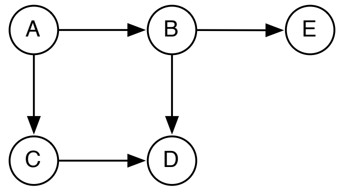
(a) (5 points) Assume you are given an associative function $f(a, b)$: `int seq` $\times$ `int seq` $\rightarrow$ `int seq` which takes two sequences of length n_1 and n_2 returning a sequence of length $n_1 + n_2$. It does $O((n_1 + n_2)^2)$ work and $O(\log(n_1 + n_2))$ span. What is the work and span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

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

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

(c) (5 points) Given a graph with integer edge weights between 1 and 5 (inclusive) you want to find the shortest weighted path between a pair of vertices. How would you reduce this problem to the shortest unweighted path problem, which can be solved with *breadth-first* search (BFS).

(d) (5 points) Recall the implementation of *depth-first* search (DFS) shown in class (on Tuesday Oct. 2) using the `enter` and `exit` functions. Circle the correct answer for each of the following questions assuming DFS starts at A:



`enter D` could be called before `enter E`:	True	False
`enter E` could be called before `enter D`:	True	False
`enter D` could be called before `enter C`:	True	False
`exit A` could be called before `exit B`:	True	False
`exit D` could be called before `enter B`:	True	False

Question 3: Missing Element (15 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) (10 points) Write an algorithm in SML that has $O(n)$ work and $O(\log^2 n)$ span.

```
(* Invariant:  |R| = |S|+1 *)
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 => ______________________________
       | n => (* recursive step *)
        let val p = ____________________________________________
            val Sleft = filter (lessThan p) S
            val Sright = filter (not o (lessThan p)) S
            val Rleft = ________________________________________
            val Rright = _______________________________________
        in  ____________________________________________________
            ____________________________________________________
            ____________________________________________________
            ____________________________________________________
        end
  end
```

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

Question 4: Priority Queues (20 points)

In this question we consider a special implementation of priority queue (pq) called **meldable priority queue** (or meldable/mergeable heap). This data structure supports the operation `meld(p, q)` of type `PQ * PQ -> PQ`, which produces a new priority queue that contains all the elements of its two input priority queues.

The cost of meld and other operations supported by this data structure are as follows:

Operation	*Work and Span*	*Description*
`empty`	$O(1)$	an empty priority queue
`insert`(Q, v)	$O(\lg(\lvert Q\rvert))$	insert v into Q and return a new pq
`deleteMin`(Q)	$O(\lg(\lvert Q\rvert))$	delete minimum element from Q and return the deleted element and a new pq
`meld`(Q_1, Q_2)	$O(\lg(\lvert Q_1\rvert + \lvert Q_2\rvert))$	meld the two priority queues

In the following questions, let S be an (unsorted) array-sequence of key-value pairs, and $|S| = n$.

(a) (6 points) Without using `meld`, how would you create a priority queue from S? (One-sentence answer is enough). What is the work and span of this operation? Give a tight bound using Θ.

(b) (8 points) Write an algorithm (in pseudocode or in SML) that uses the `meld` operation to construct a priority queue from S in $O(n)$ work and $O(\lg^2 n)$ span.

(c) (6 points) If the priority queue is represented as a sorted array-sequence could we match the bounds given in the table? Explain briefly. You can assume sorting on the keys requires $\Theta(n \lg n)$ time.

Question 5: Strongly Connected Component (20 points)

In this question you will write two functions on directed graphs. We assume that graphs are represented as

```
type graph = vertexSet vertexTable
```

with key comparisons taking constant work.

(a) (10 points) Given a directed graph $G = (V, E)$ its transpose is $G^T = (V, E')$ where

$$E' = \{(b, a) | (a, b) \in E\}.$$

Informally, it's another directed graph on the same vertices with every edge flipped.

Below is a skeleton of an SML definition for `transpose` that computes the transpose of a graph. Fill in the blanks to complete the implementation. Your implementation must have work in $O(|E| \lg |V|)$ and span in $O(\lg^2 |V|)$.

```
fun transpose (G : graph) : graph =
  let
    val S = vertexTable.toSeq(G)      (* returns (vertex*vertexSet) seq *)

    fun flip(u,nbrs) = Seq.map (______________________________) (vertexSet.toSeq nbrs)

    val ET = Seq.flatten(Seq.map flip S)

    val T = vertexTable._______________ ET
  in
    vertexTable.map __________________________________ T
  end
```

(b) (10 points) A *strongly connected component* of a directed graph $G = (V, E)$ is a subset S of V such that every vertex $u \in S$ can reach every other vertex $v \in S$ (i.e., there is a directed path from u to v), and such that no other vertex in V can be added to S without violating this condition. Every vertex belongs to exactly one strongly connected component in a graph.

Implement the function

```
scc : graph * vertex -> vertexSet
```

such that `scc(G,v)` returns the strongly connected component containing v. You may assume the existence of a function

```
reachable : graph * vertex -> vertexSet
```

such that `reachable(G,v)` returns all the vertices reachable from v in G. Not including the cost of `reachable`, your algorithm must run in $O(|E| \lg |V|)$ work and $O(\lg^2 |V|)$ span. You might find `transpose` useful and can assume the given time bounds.

```
fun scc (G : graph, v : vertex) : vertexSet =
```

Question 6: Interval Containment (15 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).

(b) (10 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.

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	$O\Big(\sum_{(k,v)\in T} W(f(v))\Big)$	$O\Big(\lg\lvert T\rvert + \max_{(k,v)\in T} S(f(v))\Big)$
`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	$O(\lg\lvert T\rvert)$	$O(\lg\lvert T\rvert)$
`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	$O(\lvert T\rvert)$	$O(\lg\lvert T\rvert)$
`collect` S `fromSeq` S	$O(\lvert S\rvert\lg\lvert S\rvert)$	$O(\lg^2\lvert S\rvert)$
`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 $n = \max(\lvert T_1\rvert, \lvert T_2\rvert)$ and $m = \min(\lvert T_1\rvert, \lvert T_2\rvert)$. 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.