15–210: Parallel and Sequential Data Structures and Algorithms

Practice Final

May 2016

Verify: There are 18 pages in this examination, comprising 8 questions worth a total of 152 points. The last 2 pages are an appendix with costs of sequence, set and table operations.

Time: You have 180 minutes to complete this examination.

Goes without saying: Please answer all questions in the space provided with the question. Clearly indicate your answers.

Beware: You may refer to your two double-sided 8 1/2 × 11 in sheet of paper with notes, but to no other person or source, during the examination.

Primitives: In your algorithms you can use any of the primitives that we have covered in the lecture. A reasonably comprehensive list is provided at the end.

Code: When writing your algorithms, you can use ML syntax but you don’t have to. You can use the pseudocode notation used in the notes or in class. For example you can use the syntax that you have learned in class. In fact, in the questions, we use the pseudo rather than the ML notation.

Sections

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9:30am - 10:20am</td>
<td>Edward/Angie</td>
</tr>
<tr>
<td>B</td>
<td>10:30am - 11:20am</td>
<td>Jake/Narain</td>
</tr>
<tr>
<td>C</td>
<td>12:30pm - 1:20pm</td>
<td>Sonya/Anisha</td>
</tr>
<tr>
<td>D</td>
<td>12:30pm - 1:20pm</td>
<td>Nick/Yongshan</td>
</tr>
<tr>
<td>E</td>
<td>1:30pm - 2:20pm</td>
<td>William/Bryan</td>
</tr>
<tr>
<td>F</td>
<td>1:30pm - 2:20pm</td>
<td>Sam S./Yutong</td>
</tr>
<tr>
<td>G</td>
<td>3:30pm - 4:20pm</td>
<td>Howard/Yongshan</td>
</tr>
<tr>
<td>Question</td>
<td>Points</td>
<td>Score</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>Binary Answers</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Costs</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Short Answers</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Slightly Longer Answers</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neighborhoods</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Median ADT</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Geometric Coverage</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Swap with Compare-and-Swap</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>152</strong></td>
<td></td>
</tr>
</tbody>
</table>
Question 1: Binary Answers  (30 points)

(a) (2 points) TRUE or FALSE: The expressions (Seq.reduce f I A) and (Seq.iterate f I A) always return the same result as long as f is commutative.

(b) (2 points) TRUE or FALSE: The expressions (Seq.reduce f I A) and (Seq.reduce f I (Seq.reverse A)) always return the same result if f is associative and commutative.

(c) (2 points) TRUE or FALSE: If a randomized algorithm has expected $O(n)$ work, then there exists some constant $c$ such that the work performed is guaranteed to be at most $cn$.

(d) (2 points) TRUE or FALSE: Solving recurrences with induction can be used to show both upper and lower bounds?

(e) (2 points) TRUE or FALSE: Let $p$ be an odd prime. In open address hashing with a table of size $p$ and given a hash function $h(k)$, quadratic probing uses $h(k, i) = (h(k) + i^2) \mod p$ as the $i$th probe position for key $k$. If there is an empty spot in the table quadratic hashing will always find it.

(f) (2 points) TRUE or FALSE: Bottom-Up Dynamic Programming can be parallel, whereas the Top-Down version as described in class (ie, purely functional) is always sequential.

(g) (2 points) TRUE or FALSE: The height of any treap is $O(\log n)$.

(h) (2 points) TRUE or FALSE: It is possible to write insert for treaps that uses the split operation but not the join operation.

(i) (2 points) TRUE or FALSE: Dijkstra’s algorithm always terminates even if the input graph contains negative edge weights.

(j) (2 points) TRUE or FALSE: A $\Theta(n^2)$-work algorithm always takes longer to run than a $\Theta(n \log n)$-work algorithm.

(k) (2 points) TRUE or FALSE: We can improve the work efficiency of a parallel algorithm by using granularity control.

(l) (2 points) TRUE or FALSE: We can measure the work efficiency of a parallel algorithm by measuring the running time (work) of the algorithm on a single core, divided by the running time (work) of the sequential elision of the algorithm.

(m) (2 points) TRUE or FALSE: Some atomic read-modify-write operations such as compare-and-swap suffer from the ABA problem.

(n) (2 points) TRUE or FALSE: Race conditions are just when two concurrent threads write to the same location.

(o) (2 points) TRUE or FALSE: In a greedy scheduler a processor cannot sit idle if there is work to do.
Question 2: Costs  (12 points)

(a) (6 points) Give tight asymptotic bounds ($\Theta$) for the following recurrence using the tree method. Show your work.

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

(b) (6 points) Check the appropriate column for each row in the following table:

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>root dominated</th>
<th>leaf dominated</th>
<th>balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W(n) = 2W(n/2) + n^{1.5}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$W(n) = \sqrt{n}W(\sqrt{n}) + \sqrt{n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$W(n) = 8W(n/2) + n^2$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 3: Short Answers  (26 points)
Answer each of the following questions in the spaces provided.

(a) (3 points) What simple formula defines the parallelism of an algorithm (in terms of work and span)?

(b) (3 points) Name two algorithms we covered in this course that use the greedy method.

(c) (3 points) Given a sequence of key-value pairs $A$, what does the following code do?

\[
\text{Table.map Seq.length (Table.collect A)}
\]

(d) (5 points) Consider an undirected graph $G$ with unique positive weights. Suppose it has a minimum spanning tree $T$. If we square all the edge weights and compute the MST again, do we still get the same tree structure? Explain briefly.

(e) (3 points) What asymptotically efficient parallel algorithm/technique can one use to count the number of trees in a forest (tree and forest have their graph-theoretical meaning)? (Hint: the ancient saying of “can’t see forest from the trees” may or may not be of help.) Give the work and span for your proposed algorithm.

(f) (3 points) What are the two ordering invariants of a Treap? (Describe them briefly.)

(g) (6 points) Is it the case that in a leftist heap the left subtree of a node is always larger than the right subtree. If so, argue why (briefly). If not, give an example.
Question 4: Slightly Longer Answers  (20 points)

(a) (6 points) Certain locations on a straight pathway recently built for robotics research have to be covered with a special surface, so CMU hires a contractor who can build arbitrary length segments to cover these locations (a location is covered if there is a segment covering it). The segment between a and b (inclusive) costs \((b - a)^2 + k\), where \(k\) is a non-negative constant. Let \(k \geq 0\) and \(X = \langle x_0, \ldots, x_{n-1} \rangle\), \(x_i \in \mathbb{R}_+\), be a sequence of locations that have to be covered. Give an \(O(n^2)\)-work dynamic programming solution to find the cheapest cost of covering these points (all given locations must be covered). Be sure to specify a recursive solution, identify sharing, and describe the work and span in terms of the DAG.

(b) (7 points) Here is a slightly modified version of the algorithm given in class for finding the optimal binary search tree (OBST):

\[
\text{function } \text{OBST} (A) = \\
\text{let } \\
\text{function } \text{OBST}' (S, d) = \\
\text{if } |S| = 0 \text{ then } 0 \\
\text{else } \min_{i \in \langle 1, \ldots, |S| \rangle} (\text{OBST}' (S_{1:i-1}, d + 1) + d \times p(S_i) + \text{OBST}' (S_{i+1:|S|}, d + 1)) \\
\text{in } \\
\text{OBST}' (A, 1) \end{align*}

Recall that \(S_{i:j}\) is the subsequence \(\langle S_i, S_{i+1}, \ldots, S_j \rangle\) of \(S\). For \(|A| = n\), place an asymptotic upper bound on the number of distinct arguments \(\text{OBST}'\) will have (a tighter bound will get more credit).

(c) (7 points) Given \(n\) line segments in 2 dimensions, the 3-intersection problem is to determine if any three of them intersect at the same point. Explain how to do this in \(O(n^2)\) work and \(O(\log^2 n)\) span. You can assume the lines are given with integer endpoints (i.e. you can do exact arithmetic and not worry about roundoff errors).
Question 5: Neighborhoods  (20 points)

Suppose that you are given a weighted, directed graph $G$ representing the road network in a
city. Your mission is to develop a “walking paths” algorithm that may not always return the
shortest paths but will return a path between two points of interest that is enjoyable to walk.
To this end, suppose that the graph $G$ is labeled with its neighborhood. For example, a vertex
representing the Gates building may have an “oakland” label.

In $G$, a walking path from a source in a neighborhood to another vertex in the same neighbor-
hood is defined as the shortest path that never leaves that neighborhood—all the vertices on
the shortest path are in the neighborhood.

Throughout assume that $G$ contain no negative edges. Use $n$ for the number of vertices in the
graph and $m$ for the number of edges.

(a) (5 points) Describe how to modify Dijkstra’s algorithm so that it calculates in $H$ walking
paths from a source to all the other vertices in the same neighborhood.

(b) (5 points) What is the work and span of your algorithm? Give a tight bound. Define any
extra variables that you may use, if any.

\[
\text{Work} = \\
\text{Span} =
\]
(c) For this part, assume that you live in a city that is planned to be walkable. Specifically, the city consists of a single center vertex $c$ with $k$ outgoing edges/streets each of which connects with one of $k$ neighborhoods with $n_1 \ldots n_k$ vertices and $m_1 \ldots m_k$ edges respectively. Furthermore, you can walk on a street in either direction, i.e., each edge has a corresponding reverse edge with the same weight. The graph below illustrates an example with $k = 5$, where $G_1 \ldots G_k$ represents the neighborhoods.

Give a parallel algorithm for the SSSP (single-source shortest paths) problem that given a source $s$ finds the shortest paths to all vertices in the graph. Your algorithm should take advantage of the special topology of your city.

You are not allowed to use Bellman-Ford because it will likely perform too much work and it still has a relatively large span.

i. (5 points) Describe your algorithm. Let $G_s$ denote the neighborhood for the source $s$.

ii. (5 points) What is the work and span of your algorithm

\[
Work = \quad \text{__________________}
\]

\[
Span = \quad \text{__________________}
\]
Question 6: Median ADT  (12 points)

The median of a set \( C \), denoted by \( \text{median}(C) \), is the value of the \( \lceil n/2 \rceil \)-th smallest element (counting from 1). For example,

\[
\text{median}\{1, 3, 5, 7\} = 3 \\
\text{median}\{4, 2, 9\} = 4
\]

In this problem, you will implement an abstract data type \( \text{medianT} \) that maintains a collection of integers (possibly with duplicates) and supports the following operations:

- \( \text{insert}(C, v) : \text{medianT} \times \text{int} \rightarrow \text{medianT} \): add the integer \( v \) to \( C \).
- \( \text{median}(C) : \text{medianT} \rightarrow \text{int} \): return the median value of \( C \).
- \( \text{fromSeq}(S) : \text{int Seq.t} \rightarrow \text{medianT} \): create a \( \text{medianT} \) from \( S \).

Throughout this problem, let \( n \) denote the size of the collection at the time, i.e., \( n = |C| \).

(a) (5 points) Describe how you would implement the \( \text{medianT} \) ADT using (balanced) binary search trees so that \( \text{insert} \) and \( \text{median} \) take \( O(\log n) \) work and span.

(b) (7 points) Using some other data structure, describe how to improve the work to \( O(\log n) \), \( O(1) \) and \( O(|S|) \) for the three operations respectively. The \( \text{fromSeq} S \) function needs to run in \( O(\log^2 |S|) \) expected span and the work can be expected case. (\textit{Hint: think about maintaining the median, the elements less than the median, and the elements greater than the median separately.})
Question 7: Geometric Coverage (12 points)

For points \( p_1, p_2 \in \mathbb{R}^2 \), we say that \( p_1 = (x_1, y_1) \) covers \( p_2 = (x_2, y_2) \) if \( x_1 \geq x_2 \) and \( y_1 \geq y_2 \). Given a set \( S \subseteq \mathbb{R}^2 \), the geometric cover number of a point \( q \in \mathbb{R}^2 \) is the number of points in \( S \) that \( q \) covers. Notice that by definition, every point covers itself, so its cover number must be at least 1.

In this problem, we’ll compute the geometric cover number for every point in a given sequence. More precisely:

**Input:** a sequence \( S = \langle s_1, \ldots, s_n \rangle \), where each \( s_i \in \mathbb{R}^2 \) is a 2-d point.

**Output:** a sequence of pairs each consisting of a point and its cover number. Each point must appear exactly once, but the points can be in any order.

Assume that we use the **ArraySequence** implementation for sequences.

(a) (4 points) Develop a brute-force solution \( gcnBasic \) (in pseudocode or Standard ML). Despite being a brute-force solution, your solution should not do more work than \( O(n^2) \).

(b) (4 points) In words, outline an algorithm \( gcnImproved \) that has \( O(n \log n) \) work. You may assume an implementation of **OrderedTable** in which \( \text{split}, \text{join}, \text{and insert} \) have \( O(\log n) \) cost (i.e., work and span), and \( \text{size} \) and \( \text{empty} \) have \( O(1) \) cost.
(c) (4 points) Show that the work bound cannot be further improved by giving a lower bound for the problem.
Question 8: Swap with Compare-and-Swap  (20 points)

(a) (10 points) Write a function \texttt{swap} that takes two memory locations \texttt{la} and \texttt{lb} and atomically swaps their values using compare-and-swap. Recall that compare-and-swap takes a memory location \(\ell\), an old value \(v\), and a new value \(w\) and atomically replaces the contents of \(\ell\) with \(w\) if the contents of \(\ell\) is equal to \(v\).

\[
\text{long lock} = 0;
\]

\[
\textit{function} \text{ swap\text{-}with\text{-}cas (la: long, lb: long) =}
\]
(b) (10 points) Does your algorithm suffer from the ABA problem? If so, explain how it does, and whether the problem affects the correctness of your algorithm. If so, then can you describe briefly a way to fix the problem (no pseudo-code needed)?
Appendix: Library Functions

signature SEQUENCE =

sig
  type 'a t
  type 'a seq = 'a t
  type 'a ord = 'a * 'a -> order
  datatype 'a listview = NIL | CONS of 'a * 'a seq
  datatype 'a treeview = EMPTY | ONE of 'a | PAIR of 'a seq * 'a seq

exception Range
exception Size

val nth : 'a seq -> int -> 'a
val length : 'a seq -> int
val toList : 'a seq -> 'a list
val toString : ('a -> string) -> 'a seq -> string
val equal : ('a * 'a -> bool) -> 'a seq * 'a seq -> bool

val empty : unit -> 'a seq
val singleton : 'a -> 'a seq
val tabulate : (int -> 'a) -> int -> 'a seq
val fromList : 'a list -> 'a seq

val rev : 'a seq -> 'a seq
val append : 'a seq * 'a seq -> 'a seq
val flatten : 'a seq seq -> 'a seq

val filter : ('a -> bool) -> 'a seq -> 'a seq
val map : ('a -> 'b) -> 'a seq -> 'b seq
val zip : 'a seq * 'b seq -> ('a * 'b) seq
val zipWith : ('a * 'b -> 'c) -> 'a seq * 'b seq -> 'c seq

val enum : 'a seq -> (int * 'a) seq
val filterIdx : (int * 'a -> bool) -> 'a seq -> 'a seq
val mapIdx : (int * 'a -> 'b) -> 'a seq -> 'b seq
val update : 'a seq * (int * 'a) -> 'a seq
val inject : 'a seq * (int * 'a) seq -> 'a seq

val subseq : 'a seq -> int * int -> 'a seq
val take : 'a seq -> int -> 'a seq
val drop : 'a seq -> int -> 'a seq
val splitHead : 'a seq -> 'a listview
val splitMid : 'a seq -> 'a treeview

val iterate : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b
val iteratePrefixes : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq * 'b
val iteratePrefixesIncl : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq
val reduce : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a
val scan : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq
val scanIncl : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq
val sort : 'a ord -> 'a seq -> 'a seq
val merge : 'a ord -> 'a seq * 'a seq -> 'a seq
val collect : 'a ord -> ('a * 'b) seq -> ('a * 'b seq) seq


```plaintext
val collate : 'a ord -> 'a seq ord
val argmax : 'a ord -> 'a seq -> int
val $ : 'a -> 'a seq
val % : 'a list -> 'a seq
end
```

<table>
<thead>
<tr>
<th>ArraySequence</th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty ()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>singleton a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>length s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nth s i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subseq s (i, len)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tabulate f n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if f(i) has W_i work and S_i span</td>
<td>O(n-1 \sum_{i=0} W_i)</td>
<td>O(n-1 \max_{i=0} S_i)</td>
</tr>
<tr>
<td>map f s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if f(s[i]) has W_i work and S_i span, and</td>
<td>O(n)</td>
<td>O(lg n)</td>
</tr>
<tr>
<td>zipWith f (s, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if f(s[i], t[i]) has W_i work and S_i span, and min(</td>
<td>s</td>
<td>,</td>
</tr>
<tr>
<td>reduce f b s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if f does constant work and</td>
<td>O(n)</td>
<td>O(lg n)</td>
</tr>
<tr>
<td>scan f b s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if f does constant work and</td>
<td>O(n)</td>
<td>O(lg n)</td>
</tr>
<tr>
<td>filter p s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if p does constant work and</td>
<td>O(n)</td>
<td>O(lg n)</td>
</tr>
<tr>
<td>flatten s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O(n-1 \sum_{i=0} (1 +</td>
<td>s[i]</td>
<td>)</td>
</tr>
<tr>
<td>sort cmp s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if cmp does constant work and</td>
<td>O(n lg n)</td>
<td>O(lg^2 n)</td>
</tr>
<tr>
<td>merge cmp (s, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if cmp does constant work,</td>
<td>O(m + n)</td>
<td>O(lg(m + n))</td>
</tr>
<tr>
<td>append (s,t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>O(m + n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
```
signature TABLE =
  sig
    structure Key : EQKEY
    structure Seq : SEQUENCE

    type 'a t
    type 'a table = 'a t

    structure Set : SET where Key = Key and Seq = Seq

    val size : 'a table -> int
    val domain : 'a table -> Set.t
    val range : 'a table -> 'a Seq.t
    val toString : ('a -> string) -> 'a table -> string
    val toSeq : 'a table -> (Key.t * 'a) Seq.t

    val find : 'a table -> Key.t -> 'a option
    val insert : 'a table * (Key.t * 'a) -> 'a table
    val insertWith : ('a * 'a -> 'a) -> 'a table * (Key.t * 'a) -> 'a table
    val delete : 'a table * Key.t -> 'a table

    val empty : unit -> 'a table
    val singleton : Key.t * 'a -> 'a table
    val tabulate : (Key.t -> 'a) -> Set.t -> 'a table
    val collect : (Key.t * 'a) Seq.t -> 'a Seq.t table
    val fromSeq : (Key.t * 'a) Seq.t -> 'a table

    val map : ('a -> 'b) -> 'a table -> 'b table
    val mapKey : (Key.t * 'a -> 'b) -> 'a table -> 'b table
    val filter : ('a -> bool) -> 'a table -> 'a table
    val filterKey : (Key.t * 'a -> bool) -> 'a table -> 'a table

    val reduce : ('a * 'a -> 'a) -> 'a -> 'a table -> 'a
    val iterate : ('b * 'a -> 'b) -> 'b -> 'a table -> 'b
    val iteratePrefixes : ('b * 'a -> 'b) -> 'b -> 'a table -> ('b table * 'b)

    val union : ('a * 'a -> 'a) -> ('a table * 'a table) -> 'a table
    val intersection : ('a * 'b -> 'c) -> ('a table * 'b table) -> 'c table
    val difference : 'a table * 'b table -> 'a table

    val restrict : 'a table * Set.t -> 'a table
    val subtract : 'a table * Set.t -> 'a table

    val $ : (Key.t * 'a) -> 'a table
  end
signature SET =
  sig
    structure Key : EQKEY
    structure Seq : SEQUENCE

type t
  type set = t

  val size : set -> int
  val toString : set -> string
  val toSeq : set -> Key.t Seq.t

  val empty : unit -> set
  val singleton : Key.t -> set
  val fromSeq : Key.t Seq.t -> set

  val find : set -> Key.t -> bool
  val insert : set * Key.t -> set
  val delete : set * Key.t -> set

  val filter : (Key.t -> bool) -> set -> set

  val reduceKey : (Key.t * Key.t -> Key.t) -> Key.t -> set -> Key.t
  val iterateKey : ('a * Key.t -> 'a) -> 'a -> set -> 'a

  val union : set * set -> set
  val intersection : set * set -> set
  val difference : set * set -> set

  val $ : Key.t -> set
end
<table>
<thead>
<tr>
<th>MKTreapTable</th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>size $T$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>filter $f T$</td>
<td>$\sum_{(k \mapsto v) \in T} W(f(v)) \lg</td>
<td>T</td>
</tr>
<tr>
<td>map $f T$</td>
<td>$\sum_{k \in X} W(f(k)) \max_{k \in X} S(f(k))$</td>
<td></td>
</tr>
<tr>
<td>reduce $f b T$</td>
<td>$O(</td>
<td>T</td>
</tr>
<tr>
<td>insertWith $f (T, (k, v))$</td>
<td>$O(\lg</td>
<td>T</td>
</tr>
<tr>
<td>find $T k$</td>
<td>$O(\lg</td>
<td>T</td>
</tr>
<tr>
<td>delete $(T, k)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>domain $T$</td>
<td>$O(</td>
<td>T</td>
</tr>
<tr>
<td>range $T$</td>
<td>$O(</td>
<td>T</td>
</tr>
<tr>
<td>toSeq $T$</td>
<td>$O(</td>
<td>S</td>
</tr>
<tr>
<td>collect $S$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fromSeq $S$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each argument pair $(A, B)$ below, let $n = \max(|A|, |B|)$ and $m = \min(|A|, |B|)$.

<table>
<thead>
<tr>
<th>MKTreapTable</th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>union $f (X, Y)$</td>
<td>$O(m \lg \frac{n+m}{m})$</td>
<td>$O(\lg(n + m))$</td>
</tr>
<tr>
<td>intersection $f (X, Y)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>difference $(X, Y)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restrict $(T, X)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subtract $(T, X)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>