**COS 226** **Algorithms and Data Structures** **Fall 2014**

# Midterm

This test has 9 questions worth a total of 55 points. You have 80 minutes. The exam is closed book, except that you are allowed to use a one page cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write out and sign the Honor Code pledge just before turning in the test.**

*"I pledge my honor that I have not violated the Honor Code during this examination."*

| Problem | Score |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| Sub 1 | |

| Problem | Score |
|---|---|
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| | |
| Sub 2 | |

| | |
|---|---|
| Total | |

**Name:**

**netID:**

**Room:**

**Precept:**

| | | |
|---|---|---|
| P01 | F 9 | Andy Guna |
| P02 | F 10 | Jérémie Lumbroso |
| P03 | F 11 | Josh Wetzel |
| P03A | F 11 | Jérémie Lumbroso |
| P04 | F 12:30 | Robert MacDavid |
| P04A | F 13:30 | Shivam Agarwal |

0. **Initialization. (2 points)**

   In the space provided on the front of the exam, write your name and Princeton netID; circle your precept number; write the name of the room in which you are taking the exam; and write and sign the honor code.

1. **Memory and data structures. (5 points)**

   Suppose that you implement a left-leaning red-black BST using the following representation:

```
public class RedBlackBST<Key extends Comparable<Key>, Value> {
   private Node root;         // root of BST
   private int N;             // number of key-value pairs

   private class Node {
      private Key key;        // symbol table key
      private Value value;    // symbol table value
      private Node left;      // left child
      private Node right;     // right child
      private boolean color;  // color of link from parent
      private int count;      // number of nodes in subtree rooted at this node
   }

   ...
}
```

   Using the 64-bit memory cost model from lecture and the textbook, how much memory (in bytes) does a `RedBlackBST` object use as a function of the number of key-value pairs $N$? Use tilde notation to simplify your answer.

   *Include all memory except for the* `Key` *and* `Value` *objects themselves (because you do not know their types).*

   $\sim$ ☐ bytes

2. **Seven sorting algorithms and a shuffling algorithm. (8 points)**

   The column on the left contains an input array of 24 strings to be sorted or shuffled; the column on the right contains the strings in sorted order. Each of the other 8 columns contain the contents at some intermediate step during one of the 8 algorithms listed below.

   Match up each algorithm by writing its number under the corresponding column. Use each number exactly once.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | left | hash | left | flow | byte | byte | lifo | byte | byte | byte |
| 1 | hash | flip | left | hash | find | exch | miss | exch | ceil | ceil |
| 2 | flip | heap | left | flip | flip | find | flow | find | edge | edge |
| 3 | heap | byte | heap | heap | hash | flip | sink | flip | exch | exch |
| 4 | byte | find | hash | byte | heap | flow | left | hash | find | find |
| 5 | find | edge | find | find | left | hash | heap | heap | flip | flip |
| 6 | sort | ceil | flow | edge | sink | heap | left | left | flow | flow |
| 7 | sink | exch | edge | left | sort | left | hash | lifo | hash | hash |
| 8 | miss | flow | byte | ceil | exch | left | prim | miss | heap | heap |
| 9 | lifo | left | flip | left | flow | lifo | trie | sink | left | left |
| 10 | exch | left | ceil | exch | left | miss | find | size | left | left |
| 11 | size | left | exch | left | lifo | prim | flip | sort | left | left |
| 12 | prim | prim | lifo | prim | miss | sink | sort | ceil | lifo | lifo |
| 13 | flow | size | load | size | prim | size | exch | edge | load | load |
| 14 | left | trie | loop | lifo | size | sort | size | flow | loop | loop |
| 15 | trie | push | miss | trie | trie | trie | byte | left | miss | miss |
| 16 | push | lifo | path | push | ceil | push | push | left | push | path |
| 17 | ceil | miss | prim | miss | edge | ceil | ceil | load | sink | prim |
| 18 | left | rank | push | sink | left | left | left | loop | trie | push |
| 19 | rank | load | rank | rank | load | rank | rank | path | rank | rank |
| 20 | load | loop | sink | load | loop | load | load | prim | sort | sink |
| 21 | loop | path | size | loop | path | loop | loop | push | prim | size |
| 22 | path | sink | sort | path | push | path | path | rank | path | sort |
| 23 | edge | sort | trie | sort | rank | edge | edge | trie | size | trie |
| | ---- | ---- | ---- | ---- | ---- | ---- | ---- | ---- | ---- | ---- |
| | 0 | | | | | | | | | 9 |

(0) Original input

(1) Selection sort

(2) Insertion sort

(3) Mergesort (*top-down*)

(4) Mergesort (*bottom-up*)

(5) Quicksort (*standard, no shuffle*)

(6) Quicksort (*Dijkstra 3-way, no shuffle*)

(7) Heapsort

(8) Knuth shuffle

(9) Sorted

3. **Analysis of algorithms. (6 points)**

Suppose that you have an array of length $N$ consisting of replications of the string $BBBA$. For example, below is the array for $N = 16$: four replications of $BBBA$.

`B B B A B B B A B B B A B B B A`

(a) How many compares does *selection sort* make to sort the array as a function of $N$? Use tilde notation to simplify your answer.

$\sim$ compares

(b) How many compares does *insertion sort* make to sort the array as a function of $N$? Use tilde notation to simplify your answer.

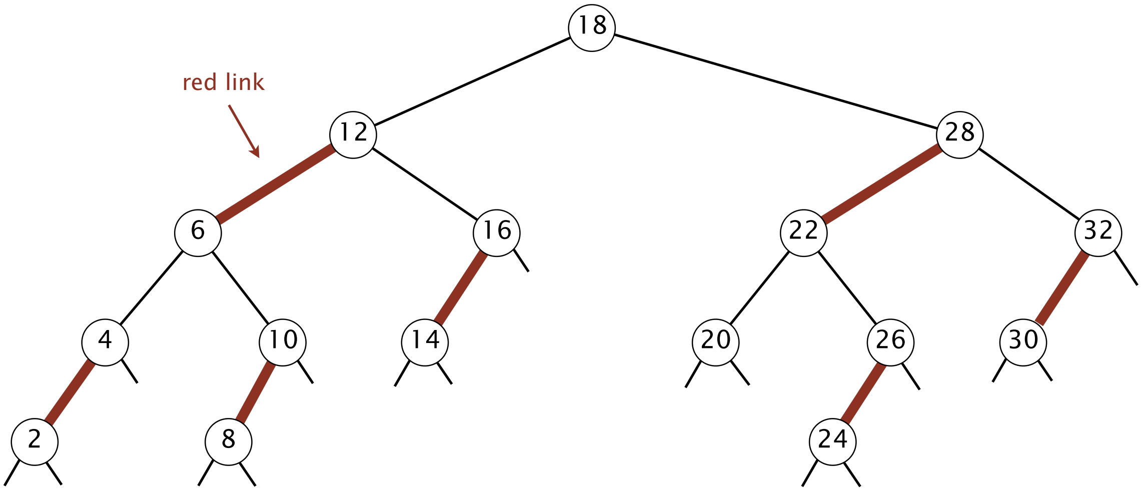
$\sim$ compares

(c) How many compares does *mergesort* make to sort the array as a function of $N$? You may assume $N$ is a power of 4. Use tilde notation to simplify your answer.

$\sim$ compares

4. **Balanced search trees. (6 points)**

Consider the following left-leaning red-black BST.



Suppose that you insert the given keys below into the LLRB above. For each insertion, give the number of color flips, the number of (left or right) rotations, and the key that appears in the root node immediately after the insertion.

*The insertions are not cumulative—you are inserting each key into the LLRB above.*

| *insertion key* | *number of color flips* | *number of rotations* | *key in root after insertion* |
|---|---|---|---|
| 17 | | | |
| 1 | | | |
| 31 | | | |
| 19 | 0 | 0 | 18 |

5. **Hash tables. (5 points)**

Suppose that the following keys are inserted into an initially empty linear-probing hash table, but not necessarily in the order given,

| *key* | *hash* |
|---|---|
| `A` | 1 |
| `D` | 5 |
| `L` | 6 |
| `M` | 0 |
| `N` | 1 |
| `S` | 6 |
| `X` | 4 |

and it result in the following hash table:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| `S` | `M` | `N` | `A` | `X` | `D` | `L` |

Assuming that the initial size of the hash table was 7 and that it did not grow or shrink, circle all possible keys that could have been the *last key inserted*.

`A` `D` `L` `M` `N` `S` `X`

6. **Problem identification. (7 points)**

You are applying for a job at a new software technology company. Your interviewer asks you to identify the following tasks as either *possible* (with algorithms and data structures introduced in this course), *impossible*, or an *open research problem.* You may use each letter once, more than once, or not at all.

I. Impossible

P. Possible

O. Open

_____ Determine whether there are any intersections among a set of $N$ axis-aligned rectangles in $N \log N$ time in the worst case.

_____ Stably sort a singly *linked list* of $N$ comparable keys using only a constant amount of extra memory and $\sim N \log_2 N$ compares.

_____ Given a binary heap of $N$ distinct comparable keys, create a binary search tree on the same set of $N$ keys, using at most $2N$ compares.

_____ Uniformly shuffle an array in linear time using only constant memory (other than the input array), assuming access to a random number generator.

_____ Find the $k$th smallest key in a left-leaning red-black BST in logarithmic time in the worst case.

_____ Implement a FIFO queue using a resizing array, in constant amortized time per operation.

_____ Given an array `a[]` of $N \geq 2$ distinct comparable keys (not necessarily in sorted order) with `a[0] < a[N-1]`, find an an index `i` such that `a[i] < a[i+1]` in logarithmic time.

7. **Multiway merge. (8 points)**

   Given $k$ sorted arrays containing a total of $N$ comparable keys, print the $N$ keys in sorted order.

   (a) Describe your algorithm in the box below. Your answer will be graded on correctness, efficiency, and clarity. For full credit, your algorithm should run in time proportional to $N \log k$ in the worst case and use extra space proportional to at most $k$.

   (b) What is the order of growth of the worst-case running time of your algorithm as a function of $N$ and $k$? Circle your answer.

   $N$ $k \log N$ $N \log k$ $N \log N$ $Nk$ $Nk \lg N$

   (c) What is the order of growth of the extra space that your algorithm uses (beyond the $k$ input arrays) as a function of $N$ and $k$? Circle your answer.

   $1$ $\log k$ $\log N$ $k$ $N$ $Nk$

8. **Move-to-front. (8 points)**

A *move-to-front* data type is a data type that stores a sequence of items. It supports inserting an item at the front of the sequence (*add*); accessing the item at index $i$ in the sequence (*item-at-index*); and moving the item at index $i$ to the front of the sequence (*move-to-front*), as documented in the following API:

`public class MoveToFront<Item>`

| | | |
|---|---|---|
| | `MoveToFront()` | *create an empty move-to-front data structure* |
| `void` | `add(Item item)` | *add the item at the front (index 0) of the sequence (thereby increasing the index of every other item)* |
| `Item` | `itemAtIndex(int i)` | *the item at index i* |
| `void` | `mtf(int i)` | *move the item at index i to index 0 (thereby increasing the index of items 0 through i – 1)* |

*All operations should take time proportional to* $\log N$ *in the worst case, where* $N$ *is the number of items in the data structure.*

Here is an example,

```
MoveToFront<String> mtf = new MoveToFront<String>();
mtf.add("A");         // A             [ add A ]
mtf.add("B");         // B A           [ add B ]
mtf.add("C");         // C B A         [ add C ]
mtf.add("D");         // D C B A       [ add D ]
mtf.add("E");         // E D C B A     [ add E ]
mtf.itemAtIndex(1);   // E D C B A     [ return D ]
mtf.mtf(1);           // D E C B A     [ move-to-front D ]
mtf.itemAtIndex(3);   // D E C B A     [ return B ]
mtf.mtf(3);           // B D E C A     [ move-to-front B ]
```

*Give a crisp and concise English description of your data structure. Your answer will be graded on correctness, efficiency, and clarity.*

(a) Declare the instance variables for your `MoveToFront` data type in the box below.

```
public class MoveToFront {








}
```

(b) Brief describe how to implement each of the operations, using either prose or code.

- `void add(Item item):`

- `Item itemAtIndex(int i)`

- `void mtf(int i):`