Princeton University – Computer Science
COS226: Data Structures and Algorithms

Midterm, Fall 2013

This test has 8 questions worth a total of 57 points. The exam is closed book, except that you are allowed to use a one page written cheat sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. Write and sign the Honor Code pledge before turning in the test.

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

	Score		Score
0		5	
1		6	
2		7	
3		8	
4			
Sub 1		Sub 2	

Total	/57

Name:

Login ID:

Exam Room: Friend 101
Friend 109
CS 105

P01	Guna	F 9	P03A	Debbie	F 11
P02	Guna	F 10	P04	Debbie	F 1230
P02A	Tengyu	F 10	P04A	Ruth	F 1230
P03	Bob	F 11			

Tips:

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we will deduct points if your answers are more complicated than necessary.
- There are a lot of problems on this exam. Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.
- On all design problems, you may assume the uniform hashing assumption unless otherwise stated.
- Not all information provided in a problem may be useful.

Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

Before exam: [☹_____☺].
After exam: [☹_____☺].

0. So it begins. (1 point). Write your name and Princeton NetID on the front page. Circle the exam room. Circle your precept. Enjoy your free point.

1. **Union Find (6 points).**

(a) Which of the following could be the result of weighted quick union on a set of 10 items? For arrays that could possibly occur, write "P" for possible in the blank provided. For arrays that could not occur, write "I" for impossible.

```
        i:   0 1 2 3 4 5 6 7 8 9
      -------------------------
------- id[i]: 0 1 2 1 1 8 6 7 8 9

------- id[i]: 4 4 1 0 8 0 0 4 6 4

------- id[i]: 9 9 3 0 0 2 8 6 8 9

------- id[i]: 5 5 5 9 5 9 8 2 9 9
```

(b) Suppose we have a union find object of size N that utilizes weighted quick union with path compression. Assuming this object is initially empty, suppose we next perform N^2 random union operations on this union find object. After all of these union operation are complete, we perform connected queries for all $\sim N^2/2$ pairs of items. Give the tree height of the resulting data structure in big theta notation (same as order of growth).

2. **Analysis of Algorithms (5 points).**

Consider the Java code below, which identifies all unique words in an input file.

```
In in = new In(args[0]);
String[] words = in.readAllStrings();
int N = words.length;

Queue<String> allWords = new Queue<String>();
Arrays.sort(words);
int i = 0;

while (i < N) {
      String currentWord = words[i];
      allWords.enqueue(currentWord);
      int j = i + 1;
      while (j < N) {
            if (!words[j].equals(currentWord))
                  break;
            j++;
      }

      i = j;
}
```

(a) What is the worst case order of growth of the call to Arrays.sort in terms of N?

(b) What is the worst case order of growth for the entire code fragment in terms of N? Consider working out a small example if you're uncertain how to proceed.

(c) Very briefly describe a solution to this problem that requires expected linear time or better. If you use any algs4 or java.util data types, give your answer in terms of those data types. You do not need to write code, though you can if you find that's the most concise way.

3. Quicksort (6 points).

(a) Show the results after 2-way partitioning. Use the I at the far left as your pivot (and yes, 2-way partitioning is the standard version from lecture).

I M I W R F D T T O S D E E P

(b) What is the order of growth of quicksort for each of the situations below? You may use answers more than once.

----- Worst case if all keys are equal for 2-way quicksort.

----- Worst case if all keys are equal for **3-way** quicksort.

----- Best case if all keys are equal for 2-way quicksort modified so that we do not stop on equal keys.

----- Best case if all keys are unique for 2-way quicksort modified so that we do not stop on equal keys.

A. $\Theta(N)$

B. $\Theta(N \lg N)$

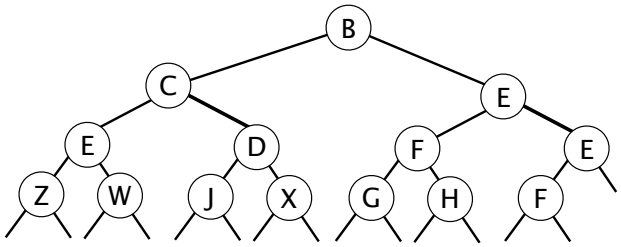
C. $\Theta(N^2)$

D. $\Theta(N^3)$

E. None of these.

4. **Heaps and Priority Queues (5 points).**

Consider the **min heap** below.



(a) Delete the **minimum** item, and give the array representation of the resulting heap.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-														

(b) Consider the 8puzzle assignment. If S was the number of search nodes in your MinPQ at the time of insertion, what was the **best case** time required to insert an additional search node?

$\Theta(1)$ $\Theta(\log S)$ $\Theta(S)$ $\Theta(S \log S)$ $\Theta(S^2)$ $\Theta(S^2 log S^2)$ $\Theta(S^4)$

(c) What was the **worst case** time required to insert an additional search node? Keep in mind that MinPQ is implemented with an array, and the size of a MinPQ is unbounded.

$\Theta(1)$ $\Theta(\log S)$ $\Theta(S)$ $\Theta(S \log S)$ $\Theta(S^2)$ $\Theta(S^2 log S^2)$ $\Theta(S^4)$

5. **Left Leaning Red Black BSTs (7 points).**

(a) We use red links when inserting in order to emulate the addition of a key to an existing node in a 2-3 tree, forming a 3 node or a 4 node. We then use elementary LLRB operations (color-flipping, left rotation, right rotation) to emulate the process of splitting a 4-node. In the worst case, how many elementary LLRB operations must we complete to emulate a single 4-node split? Do not consider cascading splits in your answer.

(b) Consider the integer keys 1, 2, and 3. Give an order of insertion for these three keys into an initially empty tree that causes the worst case given in part a.

(c) For each of the situations below in a **valid** LLRB tree, list whether the node is red or black. Recall that a node's color is the same as the link connected to its parent.

----- The largest key in a tree with more than one node.

----- The smallest key in a tree with more than one node.

----- A node whose parent is red.

----- A node whose children are the same color.

----- A freshly inserted node after the insertion operation is completed.

A. Red

B. Black

C. Either red or black

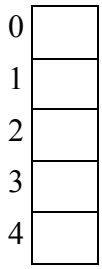
6. Hash Tables (9 points).

Consider a symbol table that uses strings containing only the digits 0-9 as keys, and uses single characters as values. Suppose that the hashCode() of these strings is given simply by the sum of their digits, e.g. the hashCode() of "342" is 3+4+2=9.

(a) Given a hash table of initial size 5, convert each hashCode() into an index using the modulus operator. **Fill in the index column of the table to the right.** The first two indices have been filled in for you.

Key	**Value**	**hashCode()**	**Index**
"13"	A	4	4
"15"	B	6	1
"2"	C	2	
"34"	D	7	
"16"	E	7	
"100"	F	1	

(b) Draw the table that results if the six keys above are inserted into the symbol table and we use separate chaining to resolve collisions. You may assume that the table does NOT resize during these insertions.



(c) Suppose we now insert the key-value pair ("81", G), and that this insert results in resizing the table to size 10. What is the size of the longest linked list after insertion is complete?

(d) Give a set of 4 String keys which, when inserted into an empty tree, result in only a single linked list of 4 nodes using the hash function described on the previous page.

(e) Assuming the uniform hashing assumption is true and that we resize such that the number of items is never more than 5 times the number of bins, would we expect to see a noticeable performance improvement if we used LLRBs instead of linked lists to store items with the same hash value? Why or why not?

(f) What is the order of growth of the worst case runtime for a **single** insertion if we're using a separate chaining hash table with resizing? Give your answer in terms of N, the number of items in the hash table, and circle your answer from the list below. You may not make the uniform hashing assumption.

$\Theta(1)$ $\Theta(\log N)$ $\Theta\left(\frac{logN}{loglogN}\right)$ $\Theta(N)$ $\Theta(N\log N)$ $\Theta\left(N\frac{logN}{loglogN}\right)$ $\Theta(N^2)$

7. Sorting (9 points).

(a) If we start with the array `6 1 14 10 5 12 11 9`, which sorting algorithms will encounter `1 5 6 10 14 12 11 9` during the sorting process?

If the sorting algorithm will encounter the second array at any point in the sorting process, write "Yes" in the blank provided. Otherwise, write "No".

----- Insertion sort
----- Selection sort
----- 2-way quicksort (no shuffling, using leftmost item as pivot)
----- Mergesort (top down)
----- Heapsort

For formatting reasons, there's all this extra space. Perhaps you can draw something down here in this area if you finish early?

(b) Awaking drenched in sweat one night, you clearly see your path to fame and fortune. You will build a robotic rhinoceros and tour the country singing songs about nature to children, who will be allowed to play and interact with the rhinoceros. While a real rhinoceros would be too dangerous, you believe a robotic rhinoceros can be kept in check. In each of the situations below, which sort would you use? In all cases, assume memory is not an issue, and that the goal is to minimize run time so that the rhino can react as quickly as possible to any potential trouble. Answers may be used many times.

A. Quicksort

B. Mergesort

C. Insertion sort

D. Selection sort

E. Knuth shuffle

----- The rhinoceros is outfitted with a large number of sensors, each of which generates objects of type Observation. Observations include many instance variables, including importance, timestamp, pressure, temperature, light intensity, etc. These are placed in an unsorted array, and every time 1000000 Observations are generated, they are delivered to a central processing unit that sorts the Observations by the `importance` field, which is of type double. What sort should you use to minimize the run time required to sort all Observations by `importance`?

----- Due to some close calls, you're going to refactor the sorting process to deal with a rare but dangerous situation where some Observations are generated with an incorrect `importance` value. For engineering reasons not described here, you can detect these by sorting by the `timestamp` and `importance` of each Observation.

Instead of `importance`, you first want to sort by the `timestamp` of each Observation. The `timestamp` is of a comparable type called DateTime. What sort should you use to minimize the run time required to sort all 1000000 Observations by `timestamp`?

----- After sorting by `timestamp`, you want to sort by `importance` such that all the objects of the same `timestamp` stay clustered. What sort should you use to minimize the run time while maintaining this clustering?

----- You iterate through the array, update the `importance` of the very rare bad Observations with a new value, and sort once more. What sort do you use to put items in order of `importance` while minimizing run time?

8. Extrinsic Max PQ (10 points)

An ExtrinsicMaxPQ is a priority queue that allows the programmer to specify the priority of an object independent of the intrinsic properties of that object. This is unlike the MaxPQ from class, which assumed the objects were comparable and used the compare method to establish priority. You may assume the Items are comparable.

```
public class ExtrinsicMaxPQ<Item extends Comparable<Item>> {
      ExtinsicMaxPQ()                  //do not implement
      void put(Item x, int priority)
      Item delMax()
}
```

If an Item already exists in the priority queue, then its priority is changed instead of adding another item. All operations should complete in **amortized logarithmic time in the worst case**. Your ExtrinsicMaxPQ should use **memory proportional to the number of items**. **For a <u>small</u> amount of partial credit, you may assume that no Item's priority is ever changed (i.e. no item is inserted twice).**

Example:

```
put("cat", 12)      // cat is inserted with priority 12
put("dog", 10)
put("swimp is a raccoon who enjoys fries and does not like to eat dirt", 11)
put("dog", 15)      // dog's priority is changed to 15
delMax()            // deletes dog, which has priority 15, cat is now max
put("fish", 20)     // fish is inserted, and is now max
put("fish", 11)     // fish's priority is reduced to 11, cat is again max
delMax()            // removes cat (priority 12), either swimp or fish now max
delMax()            // removing either swimp or fish is OK, both priority 11
```

On the **next page**, follow the prompts to give a crisp and concise English description of your data structure and how the put() and delMax() operations are implemented. Your answer will be graded on correctness, efficiency, and clarity. Feel free to use the space below to work, but be aware that it will not be considered for grading.

- Concisely describe the data structure(s) you use to implement ExtrinsicMaxPQ. If you use a common implementation of some useful data type, specify precisely what you're using (e.g. linear probing hash set of integers, resizing array based queue of doubles, separate chaining based symbol table that maps strings to queues of doubles, etc.). If you create a helper class (e.g. SearchNodes from 8puzzle), list its instance variables. **Do not describe how the data structures change when calls are made to the put() and delMax() methods** (such descriptions come later on this page).

  Also draw a diagram of your data structure(s) after the following calls:

```
put("dog", 12)
put("cat", 15)
put("laser", 7)
put("jello", 9)
put("cat", 8)        //ignore if doing partial credit
put("jello", 20)     //ignore if doing partial credit
delMax()
```

- Describe precisely how your data structure(s) change when put() is called! If there are different cases, make sure to list exactly what happens in each case. You should give your answer in terms of operations discussed in class (i.e. do not describe how things like resizing, sinking, swimming, hash table insertion, etc. work).

- Describe precisely how your data structure(s) change when delMax() is called.