**UC Berkeley – Computer Science** M25112-A
CS61B: Data Structures

Final, Spring 2015

This test has 14 questions worth a total of 60 points. The exam is closed book, except that you are allowed to use three (front-and-back) handwritten pages of notes. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write the statement out below in the blank provided and sign. You may do this before the exam begins.**

*"I have neither given nor received any assistance in the taking of this exam."*

______________________________________________

______________________________________________

| | Points | | Points |
|---|---|---|---|
| 0 | 1 | 8 | 4 |
| 1 | 6 | 9 | 4.5 |
| 2 | 3 | 10 | 7 |
| 3 | 4.5 | 11 | 0 |
| 4 | 7.5 | 12 | 8 |
| 5 | 2 | 13 | 5 |
| 6 | 2.5 | | |
| 7 | 5 | | |

Signature: _______________

Your Name: _______________

Your Student ID: _______________
Three-letter Login ID: _______
Login of Person to Left: _______
Login of Person to Right: _______
Exam Room: _______________

Tips:
- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much 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.
- Not all information provided in a problem may be useful.
- All code that we've provided on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix. The correct answer is not 'does not compile.'
- Don't panic! Recall that we shoot for around a 60% median. You should not expect to be able to do all of the problems on this exam.
- If you're feeling anxious and need to take a break, go for it. Just let a TA know, and leave your test and electronic devices behind.

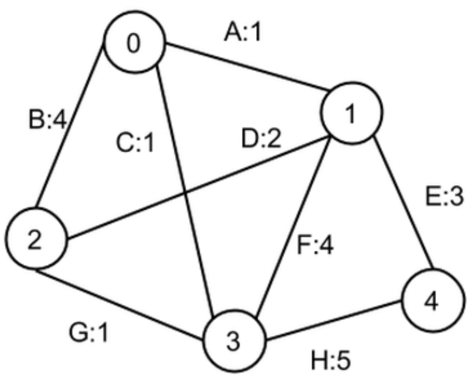
Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

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

**0. Another point. (1 points).** Write your name, login, and ID on the front page. Write your exam room. Write the IDs of your neighbors. Write the given statement and sign. Write your login in the corner of every page. ☺

**1. Basic Operations (6 Points).**

**a.** For the graph below, use Kruskal's algorithm to find the MST. The number on each edge is the weight, and the letter is a unique label you should use in your answer to specify that edge. Provide the edges **in the order they'd be found** by Kruskal's algorithm. Break any ties using the alphabetical label. Use the blanks below. You may not need all blanks. **Give your answer in terms of the alphabetical labels,** e.g. if Kruskal's starts with the edge between vertices 3 and 4, you would write H in the first blank.




____ ____ ____ ____ ____ ____

b. Repeat part a, but using Prim's algorithm, starting from vertex **#3**. **As before, give your answer in the order added and in terms of the alphabetical labels.** You may not need all of the blanks.

____ ____ ____ ____ ____ ____

c. Suppose we have the array [4, 1, 6, 2, 3, 7, 3, 4]. Give a valid partitioning of this array, using the leftmost item as a pivot. Any partitioning scheme is fine.

____ ____ ____ ____ ____ ____ ____ ____

d. Give the array that results if we use transform the array [1, 2, 4, 9, 3, 7, 5] into a max heap. Use the sinking based heapify process that we described in class (possibly helpful reminder: this process also happens to be the first step of heap sort).

____ ____ ____ ____ ____ ____ ____

2. **Basic Operations Turbo Edition (3 Points).**

a. Draw a binary tree that has a pre-order and in-order traversal of A, B, C, D, E.

b. Draw a **directed** graph whose DFS pre-order traversal is A, B, C, D, E, and whose DFS post-order traversal is C, B, D, E, A. Assume that ties are broken alphabetically.

3. **Best and Worst (4.5 Points).** Give bests and worsts as requested below.

a. Give an input that results in worst case runtime for insertion sort. Give your answer as an array of 5 integers, written in the blanks below.

____ ____ ____ ____ ____

b. Given an initially empty BST, give an insertion order for A, B, C, D, E, F, G that minimizes height. Assume we are not performing any balancing operations.

____ ____ ____ ____ ____ ____ ____

c. Suppose we want to build an array based stack that supports push and pop. In class, we learned how appropriate resizing rules can empower data structures with excellent amortized runtime. Suppose our stack has the following two rules:

- Before each push, if the array **is full, double** the size before pushing.
- After each pop, if the array is less than **half full, halve** the size of the array.

What is the worst-case amortized runtime for a sequence of pushes and pops? Give your answer in $\Theta(\cdot)$ notation in terms of N, the maximum size of the array that is reached during the stack's lifetime.

**4. Comparative Data Structures and Algorithms (7.5 points).**

a. What is the size of the largest binary min heap that is also a valid BST? Draw an example assuming all keys are letters from the English alphabet. **Assume the heap has no duplicate elements.**

b. What is the size of the largest binary max heap that is also a valid BST? Draw an example assuming all keys are letters from the English alphabet. **Assume the heap has no duplicate elements.**

c. What is the size of the largest binary min heap in which the fifth largest element is a child of the root? You do not need to draw an example. **Assume the heap has no duplicate elements.**

d. Give an example of when you'd use Quicksort over Mergesort.

e. Give an example of when you'd use a Trie-based map instead of a HashMap.

f. Why did we bother introducing heaps for implementing priority queues instead of just using left-leaning red-black binary search trees?

---

Hier ist eine keine Tausendfüßler Zone.

**5. Syntax Mastery (2 points).** Give the output of `main`. You may not need all lines provided.

______________________________________

______________________________________

______________________________________

```
public class Sklarp implements Iterable<Character>, Iterator<Character> {
    public char[] contents;
    public char magicCharacter;
    public int k;

    public Sklarp(char[] s, Character c) {
        contents = s;
        magicCharacter = c;
        k = 0;
    }

    public Iterator<Character> iterator() {
        return this;
    }

    public boolean hasNext() {
        return k < contents.length;
    }

    public Character next() {
        if (k % 3 == 0) {
            contents[k] = magicCharacter;
        }
        char returnChar = contents[k];
        k += 1;
        return returnChar;
    }

    public void remove() { throw new UnsupportedOperationException(); }

    public static void main(String[] args) {
        Sklarp g = new Sklarp("Zeoidei".toCharArray(), 'r');
        for (Character c : g) {
            System.out.print(c);
        }
        System.out.println();
        for (Character c : g) {
            System.out.print(c);
        }
        System.out.println();   }   }
```

**6. Runtime Analysis (2.5 points).** For each of the pieces of code below, give the **worst case** runtime in $\Theta(\cdot)$ notation as a function of N. Your answer should be as simple as possible (i.e. avoid unnecessary constants, lower order terms, etc.). If the worst case is an infinite loop, write an infinity symbol in the blank. Assume there is no limit to the size of an int (otherwise technically they're all constant).

________

```
public static void f1(int N) {
      int sum = 0;
      for (int i = N; i > 0; i -= 1) {
          for (int j = 0; j < i; j += 1) {
              sum += 1;
          }
      }
}
```

________

```
public static void f2(int N) {
      int sum = 0;
      for (int i = 1; i <= N; i = i*2) {
          for (int j = 0; j < i; j += 1) {
              sum += 1;
          }
      }
}
```

________

```
public static void f3(int[] a) {
      if (a.length == 0) { return; }
      int N = a.length;
      int[] newA = new int[N-1];
      for (int i = 0; i < newA.length; i += 1) {
            newA[i] = a[i];
      }
      f3(newA);
}
```

________

```
public static void f4(int N) {
      int x = N;
      while (x > 0) {
            x = x >>> 1;
      }
}
```

________

```
public static void f5(int N) {
      f1(N);           // If you left one or more of the answers
      f2(N);           // above blank, give your answer to f5 in
      f3(new int[N]);  // terms of your non-blank answers.
      f4(N);            }
```

7. **Sorting Mechanics (5 points).**

a. Below, the leftmost column is an array of strings to be sorted. The column to the far right gives the strings in sorted order. Each of the remaining columns gives the contents of the array during some intermediate step of one of the algorithms listed below. Match each column with its corresponding algorithm. Use every answer exactly once. Write your answers in the blanks provided.

| | | | | | | |
|---|---|---|---|---|---|---|
| 4873 | 1876 | 1874 | 1626 | 9573 | 2212 | 1626 |
| 1874 | 1874 | 1626 | 1874 | 7121 | 8917 | 1874 |
| 8917 | 2212 | 1876 | 1876 | 9132 | 7121 | 1876 |
| 1626 | 1626 | 1897 | 4873 | 6973 | 1626 | 1897 |
| 4982 | 3492 | 2212 | 4982 | 4982 | 9132 | 2212 |
| 9132 | 1897 | 3492 | 8917 | 8917 | 6152 | 3492 |
| 9573 | 4873 | 4873 | 9132 | 6152 | 4873 | 4873 |
| 1876 | 9573 | 4982 | 9573 | 1876 | 9573 | 4982 |
| 6973 | 6973 | 6973 | 1897 | 1626 | 6973 | 6152 |
| 1897 | 9132 | 6152 | 3492 | 1897 | 1874 | 6973 |
| 9587 | 9587 | 7121 | 6973 | 1874 | 1876 | 7121 |
| 3492 | 4982 | 8917 | 9587 | 3492 | 9877 | 8917 |
| 9877 | 9877 | 9132 | 2212 | 4873 | 4982 | 9132 |
| 2212 | 8917 | 9573 | 6152 | 2212 | 9587 | 9573 |
| 6152 | 6152 | 9587 | 7121 | 9587 | 3492 | 9587 |
| 7121 | 7121 | 9877 | 9877 | 9877 | 1897 | 9877 |
| ---- | ---- | ---- | ---- | ---- | ---- | ---- |
| __1_ | _____ | _____ | _____ | _____ | _____ | __7_ |

1: Unsorted 2: Merge 3: Quick 4: Heap 5: LSD 6: MSD 7: Sorted

Notes:

- Quicksort: Non-random and uses topmost item as pivot. We have deliberately omitted information about the partitioning strategy.
- Mergesort: Recursive implementation (a.k.a. top-down for textbook readers).
- Heapsort, LSD Radix Sort, and MSD Radix Sort: As described in class.

8. **Choosing a Sort (4 Points).**

You're an imperial engineer doing some QA on a recent batch of droids that were produced. You have a supposedly sorted array of n Droid objects that implement Comparable. However, when looking through your array, you realize these aren't the droids you're looking for! Your supervisor tells you that the machine malfunctioned, and it's made at most k mistakes: There are no more than k inversions, where **we define an inversion as a pair of droids that is not in the right order.**[1]

a. State the most efficient sorting algorithm and the big O runtime (giving the tightest bound with no unnecessary constants or lower order terms) to sort the droids if:

```
k = O(n)
Algorithm: ____________________________

Runtime:   ____________________________

k = O(n^2)
Algorithm: ____________________________

Runtime:   ____________________________

k = log(n)
Algorithm: ____________________________

Runtime:   ____________________________
```

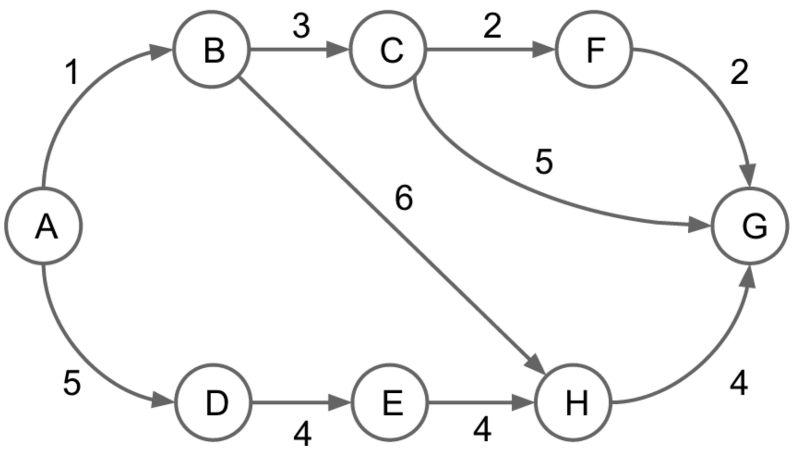
b. Two weeks later, you're given another batch of Droids that are supposed to be sorted on a 32-bit int ID, an instance variable of Droid. The sorting machine hasn't been fixed yet and again has made at most k mistakes. State the most efficient sorting algorithm and the runtime (giving the tightest bound with no unnecessary constants or lower order terms) to sort the droids by int ID if:

```
k = O(n^2)
Algorithm: ____________________________

Runtime:   ____________________________

k <= 5
Algorithm: ____________________________

Runtime:   ____________________________
```

[1] In case you've forgotten the term "inversion", here's an example that may help: Suppose we have the array [0 1 1 2 3 4 8 6 9 5 7]. This array has 6 inversions: 8-6, 8-5, 8-7, 6-5, 9-5, 9-7.

9. **Shortest Paths Algorithms (4.5 Points).**

a. For the graph below, give the order in which Dijkstra's Algorithm would visit each vertex, starting from vertex A.




__A_ ____ ____ ____ ____ ____ ____ ____

b. Change one of the weights in the graph so that the shortest paths tree returned by Dijkstra's is not correct. Hint: we showed in class that Dijkstra's returns the correct SPT as long as all edges are non-negative.

Set the weight of the edge connecting vertex ____ and vertex ____ to ____.

c. Suppose we use the following heuristic:

```
h(A) = 2
h(B) = 2
h(C) = 20
h(D) = 2
h(E) = 6
h(F) = 2
h(G) = 0
h(H) = 2
```

Recall that A* is just Dijkstra's algorithm, except that vertices are given a priority equal to the sum of their Dijkstra priority (distance from the source) plus the heuristic distance, and also that we quit when the target is visited.

**Give the path** (<u>not order visited</u>) that A* returns from A to G, you may not need all of the blanks:

__A_ ____ ____ ____

10. **Hashing and Doubly Linked Lists (7 points).** Mu Gulshan, a 61B student, was working on a DLList class. Unfortunately, Mu was transformed into an owl by a jealous Zeus before completing the implementation, so you'll need to pick up where Mu left off. For space reasons, the code for `insertFront` is hidden, but you can assume it works properly. **You may not need all lines.**

```
public class DLList {
    public int size = 0;
    public DNode sentinel = new DNode(null, null, null, null);

    /** Creates a new node and inserts it at the front of the DLL. */
    public void addToFront(String k, String v) {
        insertFront(new DNode(k, v, null, null));
    }

    /** Inserts the given node into the front of the DLL. */
    public void insertFront(DNode toInsert) { ... }

    /** Returns node containing k, or null if node doesn't exist. */
    public DNode getNode(String k) {
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________
    }

    /** Unlinks the given node from its position in the list. */
    public void detachNode(DNode toDetach) {
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________
          _____________________________________________________
    }

    public static class DNode {
        public String key, value;
        public DNode prev, next;
        public DNode(String k, String v, DNode p, DNode n) {
            key = k;
            value = v;
            prev = p;
            next = n;
        }
    }
}
```

Mu built the strange DLL class listed on the previous page as a support class for an optimized HashMap. The main idea is that recently used items are kept at the front of each list. In Mu's scheme, the HashMap (which maps Strings to Strings), behaves as follows:

1. Key/value pairs are stored in nodes in an array of buckets, where each bucket is a doubly linked list.
2. When a key/value pair is inserted into a bucket, we check to see if the key is already in the bucket.
   - If the key/value pair does not exist in the bucket, we create a new node (with that pair) at the front of the bucket.
   - If the key/value pair exists in a node in the bucket, we update that node's value, and move that node to the front.

Mu's last commit to the DLList class was the following method:

```
/** Returns node associated with string k or null if doesn't exist.
  * Moves item to the front if it does exist. */
public DNode getNodeAndPromote(String k) {
    DNode p = getNode(k);
    if (p == null) {  return null;  }
    detachNode(p);
    insertFront(p);
    return p;
}
```

Using this method (and others), fill in the put method below. **You may not need all lines.**

```
public class DLLHashMap {
     public DLList[] data;
     public DLLHashMap(int numBuckets) {
          data = new DLList[numBuckets];
          for (int i = 0; i < numBuckets; i += 1)
               data[i] = new DLList();
     }

     /** If k exists, update value and move to front. Otherwise add new
       * node (containing key and value) to front. */
     public void put(String k, String v) {
          ______________________________________________
          ______________________________________________
          ______________________________________________
          ______________________________________________
          ______________________________________________
          ______________________________________________
          ______________________________________________
          ______________________________________________
          ______________________________________________
}    }
```

**11. PNH (0 points).** This remote surfing spot was "discovered" via Google Earth in the 2000s, and is known as one of the most magnificently long barreling waves in the world. Either of its common names (as given by its discoverers, or by the people already surfing it before it was discovered) is OK.

**12. Radix Sorting and Bits (8 Points).**

a. In LSD radix sorting, we sort our inputs digit by digit, starting from the least significant digit and working our way leftwards. For each digit, we used counting sort. Conceptually, we can imagine that we used a subroutine `countSort(Something[] a, int k),` which sorts the array `a` of something based on the `k`th digit of each object in `a`.

   Suppose that we replaced counting sort with a special version of insertion sort `insertionSort(Something[] a, int k)` that insertion sorts based on only the kth digit of each `Something`. Would LSD radix sort still work? What would be the worst-case runtime in terms of N and W, where N is the number of keys and W is the width of the keys in digits? Justification is optional but may be considered for partial credit.

   `Correct  (yes/no):` ____________________

   `Runtime, big O:` ____________________

b. Same question as a, but using `heapSort(Something[] a, int k)` to sort on each digit instead of counting sort: Would this version of LSD radix sort yield the correct result? What would be the worst case runtime in terms of N and W? Justification is optional but may be considered for partial credit.

   `Correct  (yes/no):` ____________________

   `Runtime, big O:` ____________________

c. The absolute value function in Java can be implemented as:

```
public static int abs(int x) {
      if (x < 0) { return (~x + 1); }
      return x;
}
```

This function works correctly for every integer value in Java except one. Which value is this, and why? Recall that the `~x` operation flips all the bits of x.

d. Suppose we are given an LSD radix sorting function described below.

```
public class LSDRadixSorter {
      /** Sorts input digit-by-digit. */
      public static void LSDRadixSort(Digitible[] a) { ... }
}
```

LSDSorter requires that the inputs implement the Digitable interface, described below:

```
public interface Digitible {
      /** Returns the number of digits. */
      public int numDigits();
      /** Returns the kth digit. */
      public int kthDigit(int k);
}
```

Fill in `kthDigit` so that the result of our LSD radix sort is the same as if we used a regular comparison based sort. You may not need all lines given.

```
public class KetchupFriend implements Comparable<KetchupFriend>, Digitable {
      public int ketchupLevel;      // assume non-negative
      public int rednessQuotient;   // assume non-negative
      public int compareTo(KetchupFriend other) {
            if (ketchupLevel > other.ketchupLevel) { return 1; }
            else if (ketchupLevel < other.ketchupLevel) { return -1; }
            if (rednessQuotient > other.rednessQuotient) { return 1; }
            else if (rednessQuotient < other.rednessQuotient) { return -1; }
            return 0;
      }

      public int numDigits() {
            return 64;
      }

      /** Returns kth character where k = 0 is the least
        * significant digit. */
      public int kthDigit(int k) {
            _________________________________________________
            _________________________________________________
            _________________________________________________
            _________________________________________________
            _________________________________________________
      }
}
```

13. **Shortest Paths Algorithm Design (5 points).**

Warning: This problem is particularly challenging. **Do not start until you feel like you've done everything else you can.** We will be award very little partial credit for this problem. Solutions which are correct but do not meet our time and space requirements (below) will be not be awarded credit.

a. Design an efficient algorithm for the following problem: Given a weighted, directed graph G where the weights of every edge in G are all integers between 1 and 10, and a starting vertex s in G, find the distance from s to every other vertex in the graph (where the distance between two vertices is defined as the weight of the shortest path connecting them, or infinity if no such path exists).

   Your algorithm must run faster than Dijkstra's to receive credit.

b. Give the running time of your algorithm in terms of V and E (the number of vertices in the graph and the number of edges in the graph,