UC Berkeley – Computer Science
CS61B: Data Structures

Midterm #2, Spring 2017

This test has 8 questions worth a total of 120 points, and is to be completed in 110 minutes. The exam is closed book, except that you are allowed to use two double sided written cheat sheets (front and back). 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."

Signature: ___________________________

#	Points	#	Points
0	0.55	5	5.25
1	11.5	6	12.5
2	15.2	7	18
3	12	8	25
4	20		
		TOTAL	120

Name: ___________________________

SID: ___________________________

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.
- Unless otherwise stated, all given code 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. Unless we specifically give you the option, the correct answer is not 'does not compile.'
- ○ indicates that only one circle should be filled in.
- ☐ indicates that more than one box may be filled in.
- For answers which involve filling in a ○ or ☐, **please fill in the shape completely.**
- If the exam says "write only one statement per line", a for loop counts as one statement.

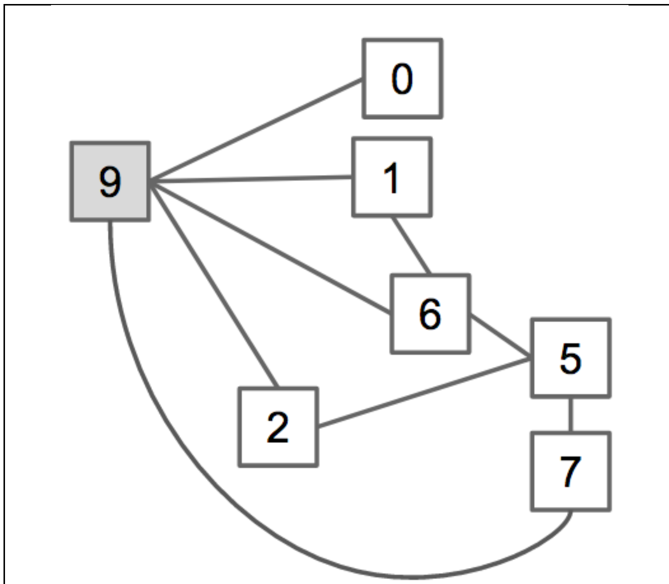
Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

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

0. So it begins (0.55 points). Write your name and ID on the front page. Write the exam room. Write the IDs of your neighbors. Write the given statement. Sign when you're done with the exam. Write your login in the corner of every page. Enjoy your free 0.55 points ☺.

1. Traversin (11.5 points).

a) (6 pts) For the graph below, give the DFS preorder, postorder, and BFS order traversals starting from **<u>vertex 9</u>**. The BFS order is the order in which vertices are enqueued. Assume ties are broken in numerical order (i.e. the edge 15 → 16 would be considered before 15 → 17).



DFS Preorder: ___________________

DFS Postorder: ____________________

BFS Order: ____________________

b) (4 pts) Suppose we have a min heap of 7 **<u>unique</u>** items and we want to print the values of the heap **<u>in increasing order</u>**. For which of our standard tree traversals will we get the values of the heap in increasing order if we print when we visit a node? **Fill in the bubbles <u>completely</u>.**

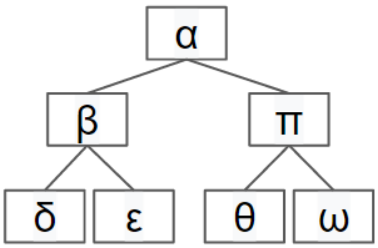
Preorder	○ Never	○ Sometimes	○ Always
Inorder	○ Never	○ Sometimes	○ Always
Postorder	○ Never	○ Sometimes	○ Always
Level order	○ Never	○ Sometimes	○ Always

c) (1.5 pts) Draw a tree with 5 nodes for which the preorder traversal is the reverse of the inorder traversal, and for which all values are unique. Or if this is not possible, simply write "Impossible".

2. An Operational Understanding (15.2 points).

a. Consider the tree on the left where greek letters represent numerical values. In the boxes to the right, shade all values that might match the text. **Assume all values are unique.** For BSTs, assume left items are less than. When treating the tree like a graph, assume nothing about the order of adjacency lists.

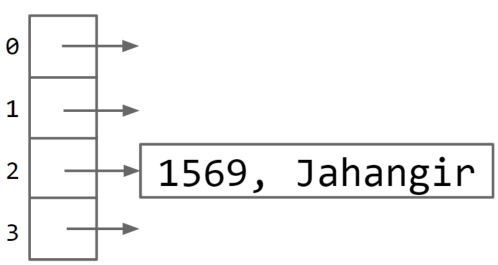
Fill in the boxes completely.



MinHeap, largest item	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
MinHeap, smallest item	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
BST, largest item	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
BST, smallest item	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
MinHeap, median item	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
BST, median item	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
MinHeap, new root after deleteMin	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
BST, new root after Hibbard[1] deletion of α	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	
MinHeap, root after inserting new item φ	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	☐φ
BST, root after inserting new item φ	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	☐φ
Last item dequeued running BFS from ω	☐α	☐β	☐π	☐δ	☐ε	☐θ	☐ω	

b. Suppose we have an initially empty hash map (as discussed in lecture) that maps a year to a famous person born that year. Suppose that the hash code of the year is given by the sum of the first and last digits, e.g. hashCode(1569) would be 10. Draw the hash table after calling **put** with the following key/value pairs. Assume that each bucket is a list, and that there is no resizing. The first one has been completed for you. Assume new items are placed at the end of the list.

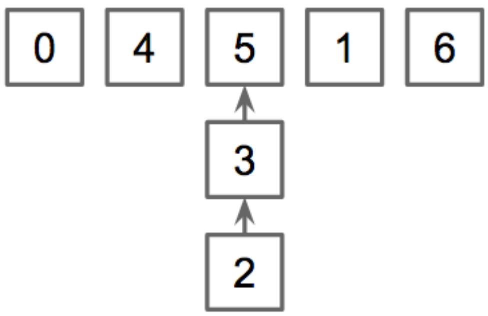
K/V pairs: [1569 / Jahangir], [155 / Cao], [1107 / Dandolo], [1737 / Paine], [713 / Stephen], [1048 / Khayyam], [1737 / Hancock]

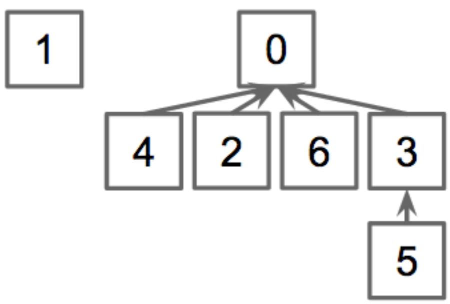


[1] Hibbard deletion is the deletion technique from lecture where we arbitrarily take one of two values to be the new root.

3. Weighted Quick Union (12 points)

a. (2 pts) Suppose we have a **<u>weighted quick union</u>** object. What calls to `connect(a, b)` produce the following trees? Assume that each **WQU** starts with all items disconnected. Or fill in the "Impossible" option if the given tree is impossible. Assume that in case of a tie, the root of the left argument is placed below the root of the right argument.

	`connect(_______, ________)` `connect(_______, ________)` ○ Impossible

	`connect( 2     , ________)`
	`connect( 4     , ________)`
	`connect(_______, ________)`
	`connect(_______, ________)`
	`connect(_______, ________)`
	
	○ Impossible

b. (4 pts) Suppose we add a new operation `undo(a, b)` that undoes an earlier Disjoint Sets `connect` operation. If `connect(a, b)` has never been called, then this method has no effect. For each of the implementations of Disjoint Sets, **mark** the corresponding box **if it is impossible** to add the `undo` operation without adding additional data structures (i.e. instance variables) to that implementation.

☐Quick Union ☐Quick Find ☐Weighted Quick Union(WQU) ☐WQU with Path Compression

c. (6 pts) Suppose we use a **graph** instead of a quick union tree to solve the disjoint sets problem. Assume we implement the `connect(a, b)` and `isConnected(a, b)` operations using a graph represented as an adjacency list with no duplicates allowed. Assume that we implement `connect` using `addEdge`, and `isConnected` using DFS with a marked array, where DFS terminates early if a connection is detected. Let N be the number of nodes in the graph, and let M be the total number of calls to either the `connect` or `isConnected` methods. For example, if we call `connect` 37 times, then `isConnected` 13 times, then `connect` 20 times, then M=70.

What will be the **worst case** runtime of <u>any single call</u> to `connect(a, b)`?

○Θ(1)	○Θ(log N)	○Θ(log M)	○Θ(N)	○Θ(M)	○Θ(N + M)	○Θ(NM)	○Θ(N + M log* M)

What will be the **worst case** runtime of <u>any single call</u> to `isConnected(a, b)`?

○Θ(1)	○Θ(log N)	○Θ(log M)	○Θ(N)	○Θ(M)	○Θ(N + M)	○Θ(NM)	○Θ(N + M log* M)

4. Asymptotics (20 points).

a) (7 pts) For each code block below, fill in the blank(s) so that the function has the desired runtime. Do not use any commas. If the answer is impossible, just write "impossible" in the blank.

```
public static void f1(int N) {      // desired runtime: Θ(N)
  for (int i = 1; i < N; __i += 1__) { System.out.println("hi"); }
}

public static void f2(int N) {      // desired runtime Θ(log N)
  for (int i = 1; i < N; __i *= 2__) { System.out.println("hi"); }
}

public static void f3(int N) {      // desired runtime Θ(1)
  for (int i = 1; i < N; __i += N__) { System.out.println("hi"); }
}
```

b) (8 pts) Give the runtime of the following functions in Θ or O notation as requested. Your answer should be as simple as possible with no unnecessary leading constants or lower order terms. For f5, your bound should be as tight as possible (so don't just put $O(N^{NM!})$ or similar for the second answer). **Don't spend too much time on these**!

```
                                         (N) = 1 + N^2
                                       /   |     |   \
                                   (N/2)    (N/2)  (N/2)  (N/2) = 4 + 4 x N^2
_Θ_____   public static void f4(int N) {   / \\\       /\\\    /\\\    /\\\
(2^N x N^2)    if (N == 0) { return; }   (N/4) (N/4) (N/4) (N/4) = 16
               f4(N / 2);                                         = 64, 256
               f4(N / 2);                1 + 4 + 16 + 64 + … + 4^(log2 N)
               f4(N / 2);                = 1 + a1(1-q^n)/(1-q)
               f4(N / 2);                = 1 + 4(1-4^n)/(1-4) = 1 + 4(4^n-1)/(3)
               g(N); // runs in Θ(N^2) time  = 1 + 4/3 x (4^(log2 N) - 1)
          }                              ~= 2^(2^log2 N) = 2^N
                                         ~= 2^N x N^2
                                         (op. of 4 calls @ each row can be omitted when
                                         compared to N^2)

_ O_(N)_   public static void f5(int N, int M) {
               if (N < 10) { return; }
               for (int i = 0; i <= N % 10; i++) {
                   f5(N / 10, M / 10);          # calls to f5(N, M)
                   System.out.println(M);       worst case, # of N%10 is 10
               }                                f(N) =1
          }                                     f(N/10) x (# of N%10)
                                                f(N/100) x (# of N%10)
     1 + 2 + 4 + 8 + … + 2^N
     = 1 + (2^N-1)/(2-1)                        1 + 10 + 100 + 10000 + … + 10^(log10 N)
     = 1 + 2^N                                  = 1 x (10^(log10 N) - 1)/(10 - 1)
     1 + 2 + 4 + 8 + … + logN                   ~ N
     = 1 + (2^logN - 1)/(2-1) = N
```

c) (0 pts) This mostly subterranean building, designed by I.M. Pei, cost more than $250,000,000 to construct, and was built by a small religious group that believes that by "building architectural masterpieces in remote locations, they are restoring the Earth's balance".

d) (5 pts) Suppose we write a method to assign careers to a list of puppies, defined below:

```
public IntTree assignPupJobs(List<Puppy> puppies, List<Job> jobs)
```

Suppose we perform timing experiments and collect the table of runtimes shown below where N is the length of the puppies list, M is the length of the jobs list, and R(N, M) is the rounded runtime for the `assignPupJobs` to complete for the given values of N and M.

N	M	R(N, M)
100	10	0.05 sec
500	10	0.08 sec
1000	10	0.13 sec
100	50	3.1 sec
500	50	15.5 sec
1000	50	30.5 sec
100	100	24.9 sec
500	100	125.1 sec
1000	100	251.9 sec

Estimate the runtime in terms of N and M, assuming that the runtime is of the form $\sim aN^{b1}M^{b2}$ in tilde notation, similar to the ungraded part of HW2. a is given for you.

$a \cong 251.9 / (1000^{b1} \times 100^{b2})$, b1 ≅ ___________________, and b2 ≅ _________________, where ≅ means "approximately equals".

You should round your exponents, giving integer values for b1 and b2.

5. Regex (5.25 points).

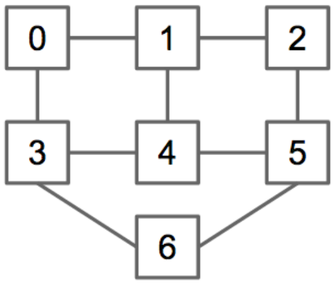
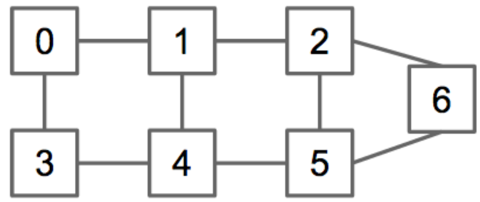
Suppose we have the regular expressions in the left most column of the table below. For each string (in the right 7 columns), check the box if the regular expression for that row matches that string.

For example, if the string `ulg` matches the regular expression `(ulg)*ometh`, you'd shade the bottom left box. **Fill in the boxes completely** (as you should be with all other boxes/circles on the exam). If none of the strings match a given regular expression, leave that row blank.

	`ulg`	`ulgometh`	`grigometh`	`ulgo`	`grigo`	`meth`	`ometh`
`ulg`	☐	☐	☐	☐	☐	☐	☐
`(ulgo|grigo)meth`	☐	☐	☐	☐	☐	☐	☐
`ulgo|grigometh`	☐	☐	☐	☐	☐	☐	☐
`((ulgo)|(grigo))meth`	☐	☐	☐	☐	☐	☐	☐
`(ulgo|grigo)*meth`	☐	☐	☐	☐	☐	☐	☐
`(u|l|g)*ometh`	☐	☐	☐	☐	☐	☐	☐
`(ulg)*ometh`	☐	☐	☐	☐	☐	☐	☐

6. Bipartite Graphs (12.5 points).

a) (3 pts) Suppose we want to color every vertex of a graph either blue or green such that no vertex touches anther vertex of the same color. This is possible for some graphs but not others. A graph where a valid coloring exists is called "bipartite". Which of the graphs below are bipartite?

	
○ Bipartite ○ Not Bipartite	○ Bipartite ○ Not Bipartite

b) (9.5 pts) Suppose we are using the undirected Graph API from the lecture / optional textbook, shown below.

```
public class Graph {
  public Graph(int V):                Create empty graph with v vertices
  public void addEdge(int v, int w): add an edge v-w
  Iterable<Integer> adj(int v):       vertices adjacent to v
  int V():                            number of vertices
  int E():                            number of edges
  ...
}
```

Fill in the method `twocolor` below such that a correct assignment to the blue vertices is printed out when the code runs, or if no such assignment is possible, an exception is thrown. Write only one statement per line (note that the for loop counts as one statement by the rules of the exam on page 1).

```
HashSet<Integer> blue = new HashSet<Integer>();
HashSet<Integer> green = new HashSet<Integer>();
twocolor(G, 0, blue, green);
System.out.println("Blue vertices are: " + blue.toString());

public static void twocolor(Graph G, int v, Set<Integer> a, Set<Integer> b){

    _________________________________

    for (__________________________________________) {
        if (__________________________________________) {
             throw new IllegalArgumentException("graph is not bipartite"); }
        if (_______________________________________) {
             _________________________________
        }
    }
}
```

7. Trees and Hashing (18 points).

a. (4.5 pts) Suppose we implement a `LLRBBucketHashSet` where the hash table buckets are stored as left leaning red black binary search trees. Assume we resize by doubling the number of buckets whenever the load factor L exceeds 2, and that we never decrease the number of buckets. Assume that hashCode computation is constant time. **Do not assume** that the hash code nicely spreads out items! If there are currently N items and M buckets in the hash table, fill in the runtimes for each operation for a **single call** in the table below (i.e. the first box is the best case for **one `put` call**). Give your answer in Θ notation in terms of N and M. You may not need both N and M. Not all of these facts may be relevant.

`put`		`containsKey`		`remove`	
Best case	Worst case	Best case	Worst case	Best case	Worst case

b. (4.5 pts) Fill in the runtimes below for a **single operation** for an `LLRBBucketNoResizeHashSet`, which is the same as in part a, except that the number of buckets is never increased. Give your answer in terms of N and M. You may not need to use both of these parameters. Not all facts may be relevant.

`put`		`containsKey`		`remove`	
Best case	Worst case	Best case	Worst case	Best case	Worst case

c. (2 pts) If we have an `LLRBBucketHashSet` that is **initially empty** and we perform Q insertions, what is the amortized (i.e. average) runtime for a **single call** to `put` **assuming that our hash code spreads items nicely across the buckets**? Give your answer in the blank below in terms of Q and M. You may not need to use both of these parameters. Not all facts may be relevant.

Amortized time per `put` call after Q calls: ________________________

d. (2 pts) Same question as part c, but for an `LLRBBucketNoResizeHashSet`.

Amortized time per `put` call after Q calls: ________________________

e. (3 pts) Is an `LLRBBucketHashSet` significantly worse, about the same, or significantly better than a standard HashSet that uses a linked list for buckets? Explain your answer.

LLRBBucketHashSets are: ○ Significantly Worse ○ About the same ○ Significantly Better

Explanation: ____

f. (2 pts) Suppose we wanted to implement `LLRBBucketHashSet` by using the `RedBlackBST.java` file provided in our optional textbook. Would it be more appropriate to use an extension based approach (i.e. extending `RedBlackBST`), a delegation approach (having instance variables that include a `RedBlackBST`), or either one?

It'd be better to use: ○ Extension ○ Delegation ○ Either is appropriate

8. Xelha (25 points).

Consider the method defined below which generates a `XelhaTree` from a list of numbers.

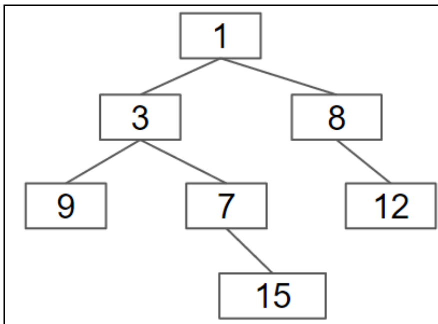
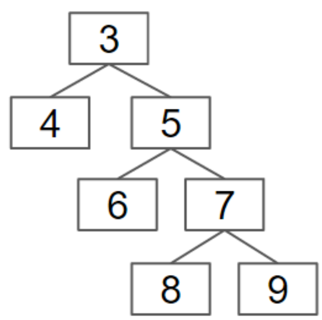
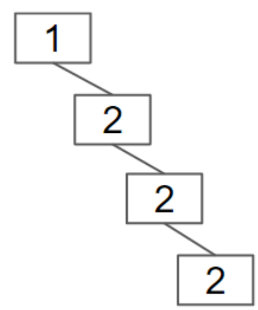
```
public IntTree generateXelhaTree(List<Integer> X)
```

Given a list of numbers X, a `XelhaTree` for that list obeys the following:

1. The `XelhaTree` has the min-heap property (i.e. every value is less than or equal to its children).
2. An inorder traversal of the `XelhaTree` visits the nodes in the same order as the list.

For example, given the list [9, 3, 7, 15, 1, 8, 12], the corresponding XelhaTree is as shown below in part a. This tree has the min-heap property, and an in-order traversal of this tree visits the vertices in the order 9, 3, 7, 15, 1, 8, 12. A XelhaTree does not need to be complete. XelhaTrees allow duplicate items.

a. (2 pts) Which of the following are valid XelhaTrees for the given sequences? The first is done for you.

		
[9, 3, 7, 15, 1, 8, 12] Valid: ●, Invalid: ○	[4, 3, 6, 5, 8, 7, 9] Valid: ○, Invalid: ○	[1, 2, 2, 2] Valid: ○, Invalid: ○

b. (3 pts) Draw a valid XelhaTree corresponding to the sequence [8, 3, 9, 1].

c. (5 pts) Draw a valid XelhaTree corresponding to the sequence [13, 7, 2, 1, 5, 16, 8, 9]. **<u>Don't spend too much time on this if you're stuck!</u>** Go back to an earlier problem and come back later.

d. (15 pts) Describe an algorithm in English for building a XelhaTree (i.e. `createXelhaTree`). **Your answer will be graded on correctness, efficiency, and clarity.** To keep things organized, you might consider using a numbered list of steps as all or part of your answer. If you didn't figure out c on the previous page, there's probably no point in working on this one.

For **full credit**, your algorithm must take **less than $\Theta(N^2)$** time in the worst case, though **partial credit** will be given for algorithms that complete in **$\Theta(N^2)$ time** in the worst case.

Ungraded: To double check your answer, give the results of running your algorithm on the input [9, 3, 6, 2, 10, 14] below.

End of Exam Decompression Space