Prelim 2

CS2110 Spring 2014

April 21, 2014

	1	2	4	5	Total
Question	True False	Short Questions	Minimax & Trees	Graphs	Total
Max	16	34	25	25	100
Score					
Grader					

1. True / False (16 points)

(a). False: If a method contains an if-statement "`if (...) int x= 5; ...`", space for local variable x is allocated every time the if-condition is found true and then deallocated when the if-statement is finished.

Explanation: Execution of a method call includes: (1) Allocating a stack frame on the call stack, which contains, among other things, space for parameters and local variables, (2) assigning argument values to parameters, (3) executing the method body, and (4) popping the stack frame from the call stack and returning. Local variables are allocated in step 1, way before execution of the method body. Nothing is done for declarations during execution of the method body.

(b). False: The following function declared in class C overrides function `equals` in class Object: `public boolean equals(C ob) { return this == ob; }`

Explanation: In class `Object`, the type of parameter `ob` is `Object`, not `C`. Change the parameter type to `Object` and it does override.

(c). True: Consider the declaration "`class C extends B implements C1, C2 {...}`" and C contains no constructor. The following expression is legal and will not cause a runtime error: "`(C1)(C1)(B)(new C())`" .

Explanation: Please look at the slides for the recitation on interfaces. You will see that an object can be cast at any time to any class/interface for which it has a partition, and that includes `C`, `B`, `C1`, and `C2`.

(d). True: Suppose class C contains an inner class IC with a private field `x`. Methods declared in class C can reference field `x`.

Explanation: Look at assignment A2, on linked lists. The outer class declared an inner class with private variables, and the outer class could use them. So you wrote a program using this feature!

(e). False: Consider the declaration "`public class C extends B {...}`". Somewhere, we declare variable v using "`ArrayList<B> v= new ArrayList<B>;`" Then the call `v.add(new C());` is not syntactically legal because a C is not a B.

Explanation: You do things like this all the time. E.g. consider an `ArrayList<Animal> b`. You can do: `b.add(new Dog(...)`. What you cannot do, using the above variable v, is `v= new ArrayList<C>`;

(f). False: Suppose classes `Dog` and `Cat` extend class `Animal`. Only class Dog declares method `bark()`. Let `kitty` be an instance of `Cat`. `((Dog)(Animal) kitty).bark()` will upcast `kitty` to `Animal`, downcast to `Dog`, and then run its `bark()` method.

Explanation: You can't cast an object to something that it is not. The only thing kitty can be cast to is `Cat`, `Animal`, and `Object`.

(g). False: Suppose that for two objects `b` and `c` of class `C`, `b.equals(c)` is false. Then the specifications of `equals` and `hashCode` in the Java API package require that `b.hashCode() != c.hashCode()`.

Explanation: Its the other way around: Equal objects must hash to the same address.

(h). False: If a GUI includes a JFrame, you can add a large number of components to that JFrame with the default layout manager for JFrame.

Explanation: Only 5 objects can be added to a JFrame and appear in it: in the North South, Center, East, and West. 5 is not a large number.

(i). False: Your class `C` implements interface `ActionListener` and thus implements procedure `actionPerformed(ActionEvent e)`. That's all you have to do in order to have `actionPerformed` called when a `JButton j` on the GUI is clicked.

Explanation: An object of class `C` has to be registered as a listener for the `JButton j`.

(j). False: Let s be a String variable. The call `s.indexOf(r)` always takes time `O(s.length())` in the worst case, regardless of the length of `String r`.

Explanation: Let `length(s) = n`. Suppose `r.length()` is `n/2`. Then, in the worst case, `r` must be compared to roughly `n/2` substrings of `s`, and each comparison can take time proportional to the length of r, i.e. `n/2`. So the time is proportional to `(n/2)(n/2)`, which is `O(n*n)`.

(k). False and True: The worst case runtime complexity of Binary Search is $O(n \log n)$.

Explanation: This is a bad question, and both possible answers got credit. Binary search is $O(n)$. But, by the formal definition of $O(g(n))$, any formula that is $O(n)$ is also $O(n \log n)$, $O(n * n)$, $O(n * n * n)$, etc. So, it was sort of a trick question but wasn't meant to be.

(l). False and True: After inserting n elements into a Binary Search Tree, the BST will always have depth $\log_2 n$.

Explanation: This is a bad question, and both possible answers got credit. The answer depends on whether the insert operation is supposed to balance the tree after inserting or not balance the tree after inserting, and we did not make clear which.

(m). True: Heapsort uses only $O(1)$ space, so it is more space-efficient than mergesort.

Explanation: You can do heart in-place: First, make the array into a max-heap; then extract values one by one, placing them in the end of the array, moving backward. This requires no extract space. Mergesort uses log$O(n \log n)$ since the depth of recursion is $O(log\ n)$.

(n). True: Selection sort always makes $O(n)$ swaps and $O(n^2)$ array comparisons to sort an array of n elements.

(o). True: Let $f(x) = x^2 + 500$ and let $g(x) = 100 * x^2$. Then $g(x)$ is $O(f(x))$.

(p). True: Using $f(x)$ and $g(x)$ as defined in the previous question, $f(x)$ is $O(g(x))$.

2. Short Questions [34 points]

2.a Hashing [5 points]

Consider hashing using linear probing to implement a set, as discussed in recitation. Assume an array of 5 of elements of class Integer, as shown below. Let the hash function be the square of the value modulo the array size, e.g. hashing 5 gives 0 (which is 25 mod 5).

```
  ----------------                ---------------
0 |_null__________|            0 |___3___________|
1 |_null__________|            1 |___1___________|
2 |_null__________|            2 |___4___________|
3 |_null__________|            3 |___5___________|
4 |_null__________|            4 |___2___________|
```

(1) Try to insert the values 1, 2, 2, 3, 4, 5 into the set, in that order —write the values in the appropriate element in the table above, crossing off the value currently in that element. But do not change the size of the array, even though the technique calls for it.
Answer. Each value has to be hashed and then inserted, using linear probing, beginning with the hashed value. Note that the value 2 can be inserted only once, since a set is being implemented. If it is already in the set, it is not inserted a second time. The values 1, 2, 3, 4, and 5 hash to 1, 4, 4, 1, 0, respectively, and their insertion one by one ends up with the array shown to the right above.

(2) Now remove the value 4 from the set. Do you simply set the array element to `null`? Explain why or why not.
Answer. No, an element can't simply be set to null. Reason: Linear probing won't work. For example, suppose in the table to the right above that 4 is removed by setting location 3 to null. Then linear probing for 5, starting at element 0, would find the null in element 2 and decide that 5 is not in the set. Look at the slides for the recitation on hashing to decide how to handle the removal of values from the set.

2.b Heaps. [5 points]

Consider maintaining a heap in an array h[0..]. Write the body of the following function. [The answer is given in the function.]

```
/** Return the index of the parent of node h[k], i.e. whose index is k.
  *  Precondition:  k > 0. */
public static int parentOf(int k) {
    return (k-10/2;
}
```

2.c Game-tree and Mini-max. [6 points]

(1) In Connect 4, starting from the first move, the possible number of game states are (choose the closest option):
A. $2^{7\times6}$ B. $2^7 \times 2^6$ C. $(6 \times 7)^2$ D. 6×7.
Answer: A There are 6×7 positions, each of which can be empty, one color, or the other color. That means $3^{7\times6}$ possible states, although many of them are not legal (e.g. there can't be a null element below a color in a column). Still, answers B, C, and D are far too small to cover the possible game states.

(2) In Connect 4, with a finite horizon, i.e. only evaluating the AI for a finite depth, will an AI of a larger depth win over an AI of smaller depth?
A. Yes, always. B. Yes, most of the times. C. No
Answer: B. With finite depth, there is no complete strategy for a player.

2.d Sorting/Searching. [7 points]

(1) Below, write the body of procedure QS. You may use English in appropriate places, and you may assume that an appropriate function does the partition algorithm, as we did in lecture.
Answer: given below. It is important to know how to use abstraction (in this case, saying WHAT partition does and not saying HOW it does it. If you have stlll not grokked this idea, study all the searching/sorting methods given on the review handout for the final.

```
/** Sort b[h..k], using the quicksort algorithm. */
public static void QS(int[] b, int h, int k) {
     if (b[h..k] has < 2 elements) return;
     int j= partition(b, h, k);
     // b[h..j-1] <= b[j] <= b[j+1..k]
     QS(b, h, j-1);
     QS(b, j+1, k);
}
```

(2) How do you change Quicksort so that it takes $O(\log n)$ instead of $O(n)$ space in the worst case? You do not have to write the code; just tell us the idea; it can be explained in one sentence.
Explain briefly: This was explained at the end of the lecture on quicksort —take a look at the lecture slides. The solution is to have a loop and, after partitioning, sort the smaller segment recursively (so that the depth of recursion is *log n*) and the larger one iteratively. Some students gave an answer something like this: use a value close to the median of the array as the pivot value. That won't do, UNLESS you know how to do that in linear time without too high a constant, and you have never seen that; it is not feasible. Since we gave the answer in lecture, and it was on the lecture slides, we expected you to know it.

2.e Induction. [5 points]

Prove by induction: A complete balanced tree of depth n has 2^n leaves. (The depth is the length of the longest path from the root to a leaf; by "complete" we mean it has as many leaves as possible.) First state the theorem in terms of a property $P(n)$, as done throughout the lecture on induction.

Theorem: For all $n \geq 0$, $P(n)$ holds, where $P(n)$ is: A complete balanced tree of depth n has 2^n leaves.
Proof. Base case: $n = 0$. A tree of depth 0 has 1 node, which is a leaf. $2^0 = 1$, so the base case holds.

Assume that $P(k)$ holds, for arbitrary $k \geq 0$. We prove $P(k+1)$. A complete balanced tree of depth $k+1$ arises by adding two children to each leaf of a complete balanced tree of depth k. Since, by $P(k)$, the tree of depth k has 2^k leaves, the tree of depth $k+1$ has twice as many leaves, i.e. 2^{k+1} leaves. Q.E.D.

(Do you know what QED stands for? Some say it's some stuffy Latin phrase. We say it's for "Quit End Done".)

2.f Exception handling. [6 points]

(1) Consider the statement below, appearing in a method `m`. Does its execution result in a `RuntimeException` being thrown out to the call of `m`? Write your answer and an explanation for it to the right of the statement.

```
try {                            Yes. The division by zero in the try block causes
    return b/0;                  a RuntimeException to be thrown; it is caught by
}                                the catch clause. The catch clause throws another
catch (RuntimeException r) {     RuntimeException, which is not caught since it
    return b/0;                  is not in a try black, so it is thrown to
}                                the place of call.
```

(2) To the right below, write down what is printed by the `println` statements during execution of the call `mm(0)`, where method `mm()` is defined as follows.

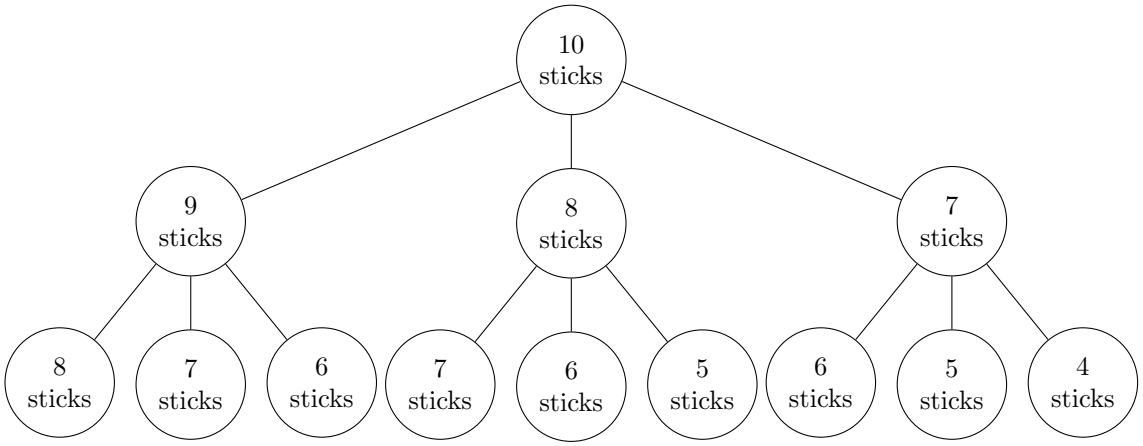
```
public static int mm(int x) {                        one
    try {                                            three
       System.out.println("one");                    four
       int b= 5/x;                                   five
       System.out.println("two");
       return b;
    } catch (RuntimeException e) {
       System.out.println("three");
       int c= 5/(x+1);
       System.out.println("four");
    }
    System.out.println("five");
    int d= 5/x;
    System.out.println("six");
    return d;
}
```

3. Minimax Game Tree [25 points]

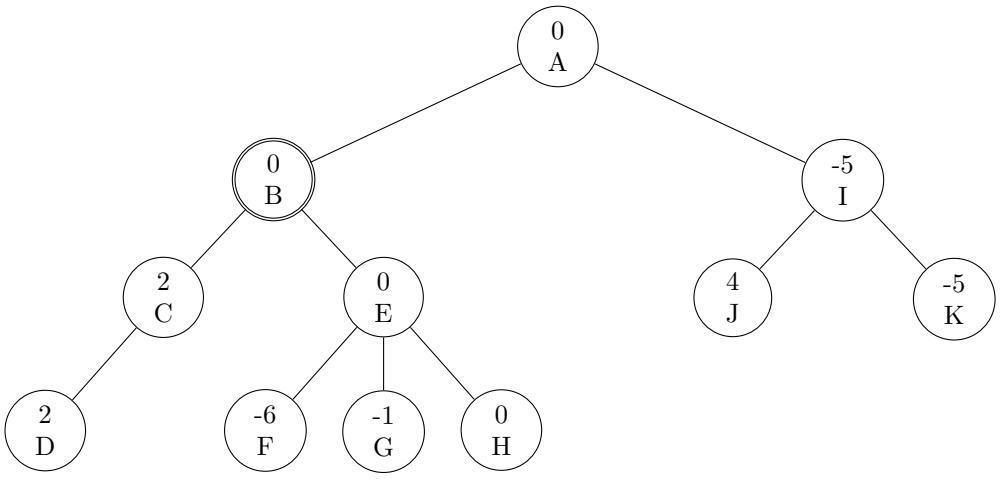
(a) [5 points] Is the Minimax algorithm, as you implemented it in Assignment 4, closer to DFS or BFS in the way it traverses the game tree? Why?

Answer: It is closer to DFS, because it recursively visits the first child of each node it encounters before continuing to process that node or its other children. As a result, it traverses all the way to a leaf along a single branch (in order to evaluate that leaf and assign it a value) before processing any other children of the parent nodes, and the right subtree of any tree will not be processed until the entire left subtree has been assigned values.

(b) [10 points] Consider the following game, which is a variant of Nim: There are 21 sticks in a pile. Two players take turns removing sticks from the pile, and the winner is the player to take the last stick. Each player can take up to 3 sticks from the pile on their turn. Show the `depth-2` game tree (i.e. a total of three levels) that an AI using the Minimax algorithm would construct for this game, if there are 10 sticks left in the pile when it is the AI's turn. Be sure to label each node of the tree with the state of the board (i.e. the pile) at that node.



(c) [10 points] The following game tree has been constructed by the AI player in a game. The details of the game are not important, and the state of the board at each node is represented by a capital letter. The table below the tree contains the values of function `evaluateBoard()` for all board states. Use the Minimax algorithm to label each node with its value, then circle the child indicating the move the AI should make to maximize its expected score.



Board state	A	B	C	D	E	F	G	H	I	J	K
`evaluateBoard()`	5	6	1	2	-2	-6	-1	0	3	4	-5

4. Graphs [25 points]

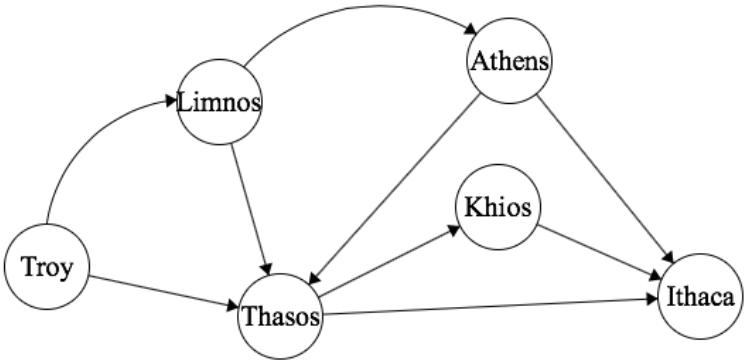


Figure 1: The possible routes from Troy to Ithaca

(1) [4 points] Write a topological sort ordering for the graph above. That is, give numbers 0 to 5 for each of the nodes above.

0: Troy 1: Limnos 2: Athens 3: Thasos 4: Khios 5: Ithaca

(2) [4 points] Odysseus, currently in the city of Troy, wants to search the cities in the above graph for his missing friend Telemachus. He knows that Telemachus is waiting for him in one of the cities. So, Odysseus, being in a rush, reasons that he will visit each city at most once. Choose an algorithm that can be used to navigate the graph efficiently to find Telemachus. Name the algorithm you chose and briefly justify your choice.

We accepted any reasonable graph traversal algorithm.

(3) [12 points] It turns out that each route has a cost associated with it. In the graph on the next page, the number on an edge from one city to another is the number of days it takes to get from the one city to the other.

Using Dijkstra's algorithm, compute the shortest path from Troy to Ithaca in terms of the time it takes to get from one city to another, writing information in the table below. The column labeled "before" shows the initial path-value for each city just before the loop of the algorithm is executed. Each column represents an iteration of the loop of the algorithm, the first one being iteration 0. For each iteration, write in its column the value for the city whose value changes at that iteration. If a value doesn't change at that iteration, don't write anything in that cell.

	before	0	1	2	3	4	5
Troy	0						
Limnos	∞	5					
Thasos	∞	3					
Athens	∞			19	~~19~~		
Khios	∞		7				
Ithaca	∞		23	20 ~~20~~	9		

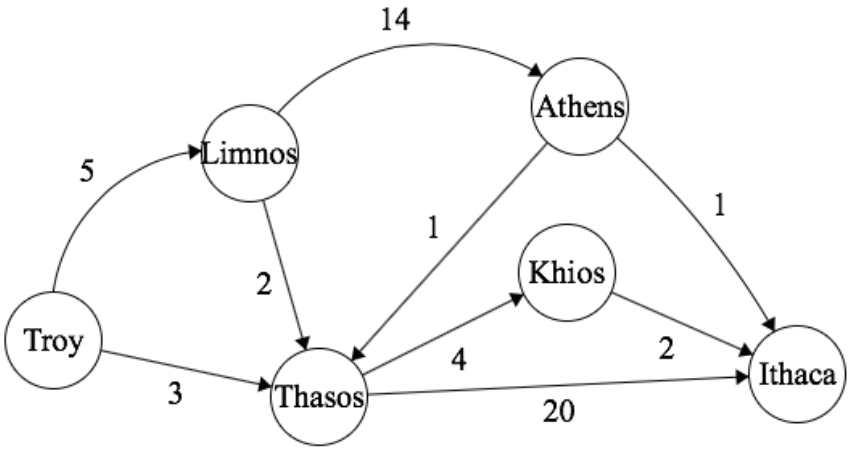
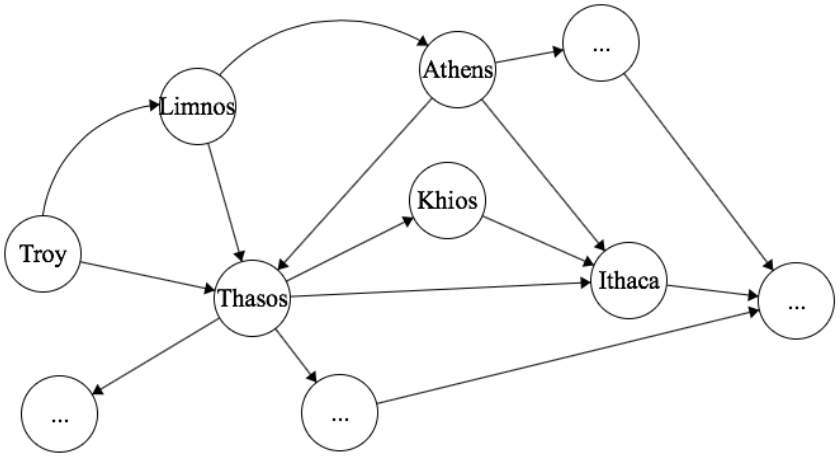


Figure 2: The possible routes from Troy to Ithaca

(4) [5 points] Suppose that the graph extends past the 6 cities as shown in the graph below. (Each node labeled "..." can also be connected to nodes that we can't see.) The world past these cities is unexplored, meaning that there could potentially be an infinite number of cities in the extended graph. Odysseus wants to find the city of Thasos. He knows that it's only a couple edges away from Troy. Does it make more sense to use Breadth-First Search or Depth-First Search to find Thasos on this potentially infinite graph? Explain your answer.



It makes more sense to use Breadth-First-Search. This is because if we use Depth-First-Search on an infinite graph, we will never encounter a node with no children and backtrack. This means that if Thasos doesn't happen to be on the path we traverse, we will never find it.