National University of Singapore

School of Computing

Semester 1 (2014/2015)

CS2010 - Data Structures and Algorithms II

Written Quiz 1 (10%)

Saturday, September 20, 2014, 10.00am-11.17am (77 minutes)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this question paper until you are told to do so.
2. Written Quiz 1 is conducted at COM1-2-206/SR1.
3. This question paper contains THREE (3) sections with sub-questions.
   It comprises TEN (10) printed pages, including this page.
4. Write all your answers in this question paper, **but only in the space provided**.
   You can use either pen or pencil. Just make sure that you write **legibly**!
   Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
5. This is an **Open Book Examination**. You can check the lecture notes, tutorial files, problem set files, Steven's 'Competitive Programming 1/2/2.5/3' book, or any other printed material that you think will be useful. But remember that the more time that you spend flipping through your files implies that you have less time to actually answering the questions.
6. **Please write your Matriculation Number here:** _____ **and your Tutorial Group Number/Tutorial TA Name/day and time:** _____
   But do not write your name in order to facilitate unbiased grading.
7. All the best :).
   After Written Quiz 1, this question paper will be collected, graded manually over recess week, and likely returned to you via your Tutorial TA on Week07.

---

1 Future Question Types in VisuAlgo ($18 \times 3 = 54$ marks)

These are the set of questions involving BST/AVL Tree/Binary Heap/UFDS/~~Bitmask~~ that Steven wish can be automated in VisuAlgo Online Quiz system in the near future (Myself and my only FYP student this AY: Erin will do these soon). Note that some of these questions **cannot be randomized** which may not be truly suitable for the actual Online Quiz (test mode, the hard level) but maybe good enough for self-training mode (the easy level). Future students who reuse this Written Quiz 1 paper for practice will see that most (if not all) questions have been integrated in VisuAlgo.

Instructions: For the first 11 questions, just declare True or False (simply circle the option). If you circle the correct answer, you will get 3 marks. For the last 7 questions, you will decide True or False and also give a **short and correct** explanation before you can get the 3 marks.

1. The **smallest** element in any BST always has **no left child**? (T/F)
2. The **smallest** element in any BST always has **no right child**? (T/F)
3. The **largest** element in any BST always has **a parent**? (T/F)
4. The **smallest** element in any BST always has **a successor**? (T/F)
5. The **smallest** element in any BST always has **no predecessor**? (T/F)
6. Suppose that we have all distinct integers between 1 and 9999 inside a BST of unknown structure and we want to search for the integer 7. It is possible to have a search sequence as follows: **9999, 9998, 9997, 9996, ... (decreases by 1), ..., 10, 9, 8, 7**. (T/F)
7. The number of structurally different BSTs (not necessarily balanced) that contains $n = 3$ distinct integers is **5**? (T/F)
8. The smallest element in a Binary Max Heap that contains > 3 distinct integers is always at one of the leaves? (T/F)
9. The second largest element in a Binary Max Heap that contains > 3 distinct integers is always one of the children of the root? (T/F)
10. An array A of n distinct integers that are sorted in descending order forms a valid Binary Max Heap? You can assume that $A[0]$ is not used and the array values occupy index $[1..n]$. (T/F)
11. Given a Binary Max Heap, calling `ShiftDown(i)` $\forall i > heapsize/2$ will never change anything in the Binary Max Heap. (T/F)

12. Suppose that we have all distinct integers between 1 and 9999 inside a BST of unknown structure and we want to search for the integer 7777. It is possible to have a search sequence as follows: **7, 932, 1010, 8089, 7523, 8134, 7777**. (T/F, then explain)

13. The insert operation in BST is always **not commutative** in the sense that inserting x and then y into an existing BST (not necessarily balanced) always produce **different** BST as inserting y and then x. Note that $x \neq y$. (T/F, then explain)

14. The delete operation in BST is always **commutative** in the sense that deleting x and then y from an existing BST (not necessarily balanced) always produces **the same** BST as deleting y and then x. Note that $x \neq y$ and both x and y exists in the BST. (T/F, then explain)

15. The second smallest element in a Binary Max Heap that contains > 3 distinct integers is always at one of the leaves? (T/F, then explain)

16. The third largest element in a Binary Max Heap that contains > 3 distinct integers is always one of the children of the root? (T/F, then explain)

17. Given $n = 16$ disjoint sets initially in a UFDS, it is possible to call `unionSet(i, j)` and/or `findSet(i)` operations to get a single tree with rank (height) 4 that represents a certain disjoint set. Both path compression and union by rank heuristics are used. (T/F, then explain)

18. *** The number of structurally different BSTs (not necessarily balanced) that contains $n = 4$ distinct integers is **15**? (T/F, then explain; note that THREE STARS = very hard question)

2 Questions That are too Hard to be Automated (25 marks)

2.1 Another way to do BST remove/delete operation? (18 marks)

In Lecture 02, you have seen a way to delete a vertex v with two children from a BST: Replace the value of v with the value of v's successor and then remove that (now duplicate) successor vertex.

Is that the only way?

When presented with the same situation (delete a vertex v with two children from a BST), Professor XYZ from University ABC proposed this deletion strategy instead (in pseudo code):

```
let x be the vertex that contains the max element in the subtree rooted at v.left
make the new right child of x be v.right
make the parent of vertex v (if exists) points directly to v.left
remove v
```

The Sub-Questions

1. (4 marks) Perform Professor XYZ's strategy on deletion of vertex $v = 13$ from this BST (in Figure 1) and draw the resulting BST as your answer.

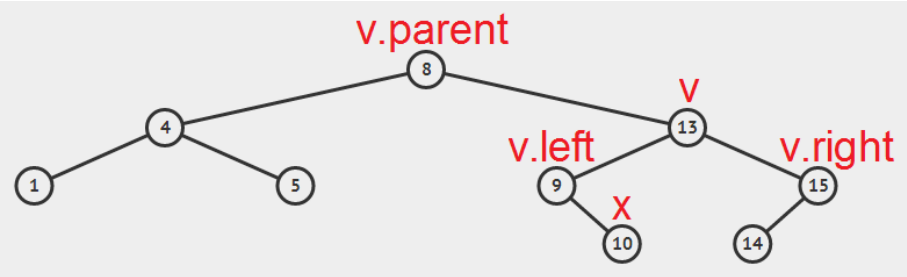

Figure 1: The Starting BST

x is the predecessor(max value in left subtree)

2. (4 marks) One more time. Delete vertex $v = 8$ from the resulting BST that you obtained from Sub-Question 1 above using Professor XYZ's strategy again and draw the resulting BST as your answer. Note that error will carry forward so please be careful.

3. (7 marks) Give a short proof that Professor XYZ's strategy is always able to delete any vertex v with two children from a BST correctly.

by choosing x as the max node in the left child of v, we are ensuring that we still have a node that has values bigger and smaller than it. hence, when v has two children, its right subtree will always be bigger than x and is retained. the nodes smaller than x in v.left will then either form the new right subtree root if v.right.node is deleted or the left subtree root if v.node is deleted and x is the new parent

4. (3 marks) What is the major drawback of Professor XYZ's strategy so that it is never used and/or mentioned in typical modern CS classes?

the node being deleted must have 2 children and may not work for an unbalanced bst or nodes with only one child

the height of the resultant tree becomes unbalanced very very fast

2.2 Is Binary Heap Still Worth Learning? (7 marks)

When attempting PS2 or in other occasions, some of you might have known that we **can** actually choose not to use Binary Heap data structure at all (that is, skip CS2010 Lecture 04) and simply use a balanced Binary Search Tree (bBST) like an AVL Tree to implement a Priority Queue.

Specifically, we will implement the operation `Enqueue(v)` of a Priority Queue as an insertion of v into a bBST and the operation `Dequeue()` of a Priority Queue as finding the maximum element of the bBST, saving it in a temporary variable, deleting that maximum element, and returning the value stored in our temporary variable. The insertion, find max, and deletion operations in a bBST runs in $O(\log n)$. Thus, we can replace Binary Heap with bBST as the underlying data structure to implement a Priority Queue without any performance penalty.

Moreover, as you have seen in PS2, the operation `Delete(v)` where v is *not* the maximum element is also much harder in a Binary (Max) Heap whereas this is simple in a bBST (just delete v). Moreover, the operation `UpdateKey(v, newv)` in a Binary Heap can be quite complicated to implement whereas this is still simple in a bBST (delete v then insert $newv$).

So, is Binary Heap still worth Learning in CS2010?

Your task in this question is interesting and this is probably the first time you see this kind of question. If you think there is a case where Binary Heap is more advantageous than bBST, argue in favor of saving Binary Heap in CS2010 syllabus. If you think all kind of operations that Binary Heap can do can be emulated by bBST, argue in favor of replacing the Data Structure discussed in Lecture 04 (Priority Queue) with bBST again instead of Binary Heap so that Steven has more time to teach other cool Data Structures and/or Algorithms to his CS2010 students. You will be given up to 7 marks based on your arguments (maximum half of this page, so do not write a long story).

3 Application (24 marks)

Disclaimer: The following question is the modified (read: simplified) version of a very recent programming (problem solving) competition: ***http://ipsc.ksp.sk/2014/real/problems/h.html****. The original problem author is Michal Forišek from Slovakia. To solve this scaled-down question, you need to remember Hash Table concepts from CS1020 and contrast it with bBST concepts from CS2010...*

There are not many data structures that are used in practice more frequently than hashsets and hashmaps (also known as associative arrays). They get a lot of praise, and deserve most of it. However, people often overestimate their capabilities. The Internet is full of bad advice such as "just use a hashset, all operations are O(1)" and "don't worry, it always works in practice". We hope that you know better. And if you don't, now you'll learn on the spot during this Written Quiz 1 :O.

Let's start by looking at two sample demo program below:

```
import java.io.*;
import java.util.*;
public class HashSetDemo {
  public static void main(String[] args) throws IOException {
    BufferedReader in = new BufferedReader( new InputStreamReader( System.in ) );
    HashSet<Long> hashset = new HashSet<Long>();
    for (String x = in.readLine(); x != null ; x = in.readLine()) {
      hashset.add( Long.parseLong(x) );
      if (!hashset.contains( Long.parseLong(x) )) // should never be false
        System.out.println("ERROR");
    }
  }
}
```

```
import java.io.*;
import java.util.*;
public class TreeSetDemo {
  public static void main(String[] args) throws IOException {
    BufferedReader in = new BufferedReader( new InputStreamReader( System.in ) );
    TreeSet<Long> treeset = new TreeSet<Long>(); // that is, change HashSet to TreeSet
    for (String x = in.readLine(); x != null ; x = in.readLine()) {
      treeset.add( Long.parseLong(x) );
      if (!treeset.contains( Long.parseLong(x) )) // should never be false
        System.out.println("ERROR");
    }
  }
}
```

Both programs do the same thing: They read a sequence of signed 64-bit values (Java Long) from the standard input, one insert the values into a HashSet while the other insert the values into a TreeSet. Both programs do a check whether those values have been properly inserted into the respective data structures and report error otherwise.

Based on their limited understanding, many people would claim that HashSetDemo program will process any sequence of n integers in $O(n)$ time while the TreeSetDemo program requires $O(n \log n)$ time to do the same. Therefore, they conclude that HashSet is always better than TreeSet that we painfully learn in Lecture 02+03 of CS2010. Are you one of those people?

The Sub-Questions:

1. (10 marks) Propose a sequence of $n = 50000$ integers that you will insert so that the program HashSetDemo—for a reason that you will explain—run very slowly. You will be graded based on the explanation of your idea. That is, you just have to discuss the general characteristics of those n integers. You do NOT have to enumerate all those 50000 integers.

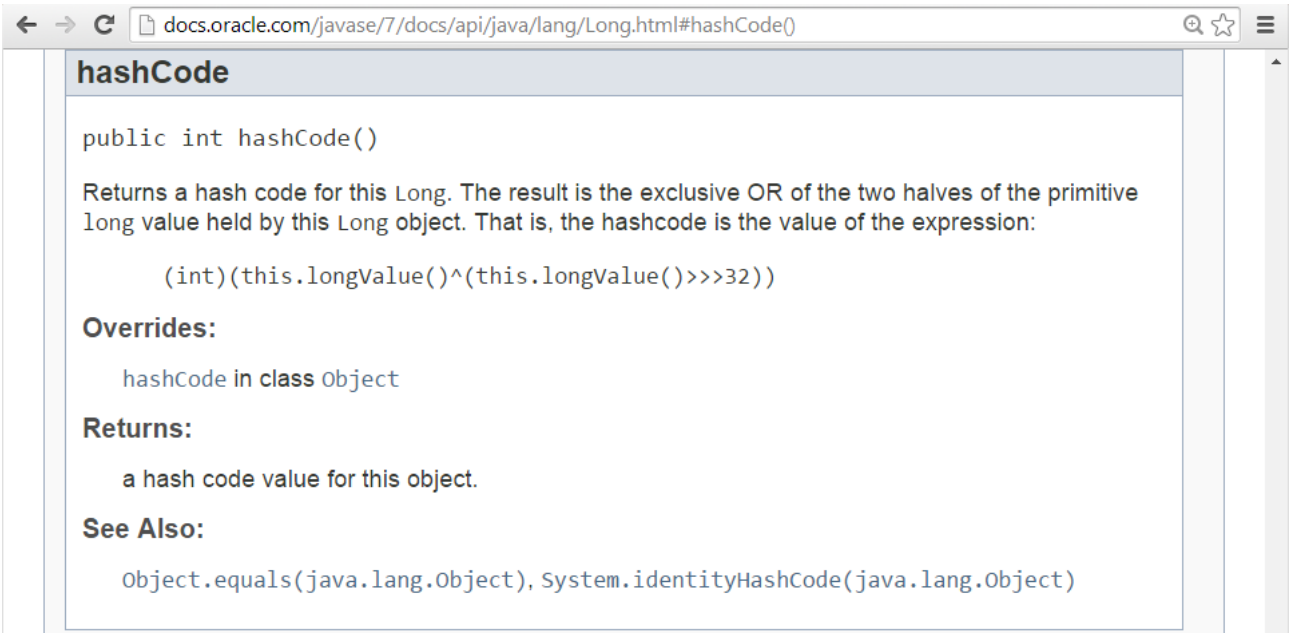

Figure 2: A very useful hint (you can assume that ">>>" is very similar to ">>")

—You can continue your answer for Sub-Question 1 here—

2. (5 marks) What happen if the same extreme case that causes program HashSetDemo to run very slowly is used on program TreeSetDemo? Explain!

3. (9 marks) Give at least three other scenarios where it is clearly better to use TreeSet (a balanced BST, CS2010 material) rather than HashSet (CS1020 material)?

when values are to be arranged in ascending order, when we want to find the smallest or max or certain kth largest values
when we want to insert a value in the correct place

.

– End of this Paper –

Candidates, please do not touch this table!

Section	Maximum Marks	Your Marks	Comments from Grader
1	51+3 bonus = 54		
2	18+7 = 25		
3	24		
Total	min(100, 100+3 bonus) = 100		