National University of Singapore

School of Computing

Semester 1 (2012/2013)

CS2010 - Data Structures and Algorithms II

Quiz 1 (15%)

Saturday, September 22, 2012, 10.00am-11.40am (100 minutes)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this question paper until you are told to do so.

2. Quiz 1 is conducted at COM1-2-206/SR1.

3. This question paper contains FIVE (5) sections with sub-questions.
   It comprises TWELVE (12) 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' 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 (or Tutor Name): _____
   But *do not* write your name in order to facilitate unbiased grading.

7. All the best :).
   After Quiz 1, this question paper will be collected, graded manually in 2-3 weeks, and likely returned to you via your Tutorial TA on Week08.

–This page is intentionally left blank. You can use it as 'rough paper'–

1 'ADT Table': (Questions → Answers) (14 marks)

The answers for this set of questions can be found in the lecture notes, tutorial files, or PS files.
Can you find them *as fast as you can*? It is $O(1)$ if you already have the answer in your memory.
Please fill in your answers on the space provided, 2 marks per question.
Grading scheme: 0 (zero correct answer), 1 (one correct answer), 2 (two correct answers).

1. Given an *unsorted* array of size n, what is the best algorithm to find the k-th largest item in the array? Write down the name of the algorithm and its worst-case time complexity:
   Name: _____
   Time Complexity: _____

2. Given a *sorted* array of size n, what is the best algorithm to find the k-th largest item in the array? Write down the name of the algorithm and its worst-case time complexity:
   Name: _____
   Time Complexity: _____

3. There are three cases for deletion in a BST:
   First, if the deleted vertex is a leaf, we simply remove that vertex.
   Second, if the deleted vertex has one child, we:

   _____

   Third, if the deleted vertex has two children, we:

   _____

4. If $n = 1024$, then $\log_2 n =$ _____ .
   If $n =$ _____ , then $\log_2 n = 20$.

5. A perfectly balanced BST with n items has height as low as _____ .
   A totally unbalanced BST with n items has height as tall as _____ .

6. List down two real-life examples where priority queue are used:
   Please show a case when the *priority* matters!
   1. _____
   _____
   2. _____
   _____

7. The time complexity of Heap Sort is _____ compared to Merge Sort.
   However, the time complexity of Heap Sort is _____ than Insertion Sort.

2 Basic (b)BST/Heap Operations (30 marks)

Grading scheme: 0 (no answer), 1 (the final answer is totally wrong), 2 (the final answer has minor mistake(s)), 5 (the final answer is correct).

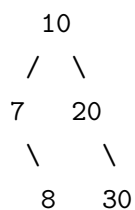### Q1. (Standard) Binary Search Tree - Insertion (5 marks)

You already have the following BST:

```
   10
  /  \
 7    20
  \     \
   8     30
```
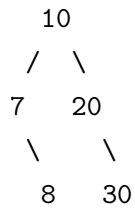
You want to insert *five more* integers: {1, 2, 3, 24, 25} into the BST above one by one, in that order. Please draw the *final* resulting BST below:

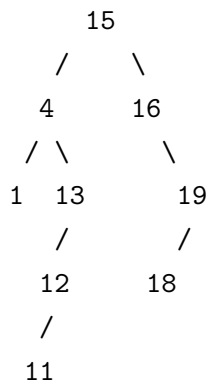### Q2. AVL Tree - Insertion (5 marks)

You already have the following AVL tree (FYI, this is similar as Q1 above):

```
   10
  /  \
 7    20
  \     \
   8     30
```

Now, insert the same *five* integers: {1, 2, 3, 24, 25} into the AVL tree above one by one, in that order. Please draw the *final* resulting AVL tree below:

Q3. (Standard) Binary Search Tree - Deletion (5 marks)

You already have the following BST:

```
     15
   /    \
  4      16
 / \       \
1  13       19
   /        /
  12       18
  /
 11
```

Now delete 18, delete 16, delete 4, delete 13, and then delete 12, in that order. Assume that if we delete a node X with two children, we will select its *successor* to replace X. Please draw the *final* resulting BST below:

Q4. Binary *Min* Heap - Extract Minimum Element (5 marks)

You already have the following Binary *Min* Heap represented in a 1-based compact array:

Index	0	1	2	3	4	5	6	7	8	9	10
Value	N/A	1	3	2	7	5	4	6	11	12	13

You extract the three smallest elements in this Binary Min Heap (obviously {1, 2, 3}) by calling `ExtractMin()` *three times*. Please update the *final* resulting Binary Min Heap compact array below:

Index	0	1	2	3	4	5	6	7
Value	N/A							

Q5. Binary *Max* Heap - Insertion/Build Heap Slower version (5 marks)

Insert *ten* integers: {-14, 34, -24, 37, 39, 57, 99, -5, -7, 10} into an initially empty Binary *Max* Heap one by one, in that order. Please draw the *final* resulting Binary *Max* Heap below:

Q6. Binary *Max* Heap - Build Heap Faster version (5 marks)

Build a Binary *Max* Heap from the same set of *ten* integers as shown in Q5 using the $O(n)$ `BuildHeap` algorithm as mentioned in Lecture 04. Please draw the *final* resulting Binary *Max* Heap below:

3 Intermediate (b)BST/Heap Operations (26 marks)

3.1 BST++ (12 marks)

You are given a BST that contains only integer values. You want to output only the vertices in the given BST that have exactly two children in *descending* order. Please look at an example below. Vertices that have exactly two children are highlighted with a star (*). For this example, we output these integers: 13, 10, 4.

```
      10*
    /     \
   4*      11
  / \        \
 1   5       15
             /
           13*
          /   \
        12    14
```
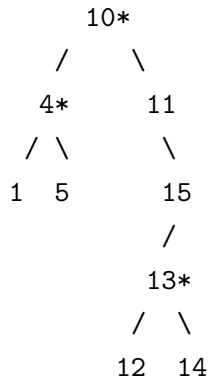
Write your (short) pseudo-code below:

if we do in order we get 4, 10, 13
in order is:

left -> root -> right
what we want is the opposite so we reverse left and right so that it is
right -> root -> left

3.2 Heap++ (14 marks)

You have just learned that Binary (Max) Heap data structure can be used as an efficient Priority Queue. Let's assume that this Priority Queue has two basic operations: `enqueue(item, priority)` that will insert `item` with certain `priority` into the priority queue and `dequeue()` that will return (and simultaneously remove) an item in the priority queue with the highest priority. Assume that if there are ≥ 1 items with the same priority, this `dequeue()` operation will return *any* of them.

Last semester, you have learned the standard First-In First-Out (FIFO) Queue and Last-In First-Out (LIFO) Stack data structures in CS1020. Now, let's combine the knowledge.

A). Show how to implement `enqueue(item)` and `dequeue()` operations of a standard FIFO Queue by making use of Priority Queue `enqueue(item, priority)` and `dequeue()` operations (6 marks)!

I will implement `enqueue(item)` by:

```
counter = 0
pq.enqueue(item, counter - -)
```

I will implement `dequeue()` by:

```
pq.dequeue()
```

B). Show how to implement `push(item)` and `pop()` operations of a standard LIFO Stack by making use of Priority Queue `enqueue(item, priority)` and `dequeue()` operations (8 marks)!

I will implement `push(item)` by:

```
counter = 0
pq.enqueue(item, count ++)
```

I will implement `pop()` by:

```
pq.dequeue()
```

4 Analysis (15 marks)

Prove (the statement is true) or disprove (the statement is false) the following statements below.

If you want to prove it, provide the proof (preferred) or at least a convincing argument.

If you want to disprove it, provide at least one counter example.

Three marks per each statement below (1 mark for saying correct/wrong, 2 marks for explanation):

Note: You are only given a small amount of space below (i.e. do **not** write too long-winded answer)!

1. Given a vertex x of a not necessarily balanced BST where x has exactly two children, we are *always* able to find a successor of x in this BST.

   true. successor is located in the right child of a node, since x has exactly two children it implies that it has a right child and the min value in the right subtree

2. The largest integer in a balanced BST (according to the +/- 1 balance criteria of AVL tree) that contains exactly three distinct integers must be located on the right subtree of the root.

   true because of bst property

3. The *third* largest integer in a Binary Max Heap with more than two integers (all integers are distinct) is always one of the children of the root.

   true. max values are in the root so the right child of root is the 2nd largest and left child of the root is 3rd largest ->
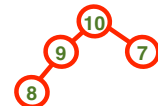
   false. the max heap property only states that the parent of i has to be the max of its two children so it is still a valid binary max heap if numbers are jumbled as long as the each root is larger than its children

4. The smallest integer of a Binary Max Heap with more than one integer (all integers are distinct) is always a leaf vertex.

   true. smallest integer doesnt have any children(it is not the max of any thing else) and hence it is always a leaf vertex

5. Instead of using Binary (Max) Heap data structure, we can implement Abstract Data Type Priority Queue with a balanced BST (e.g. AVL tree) and still maintain $O(\log n)$ performance for `enqueue(item, priority)` and `item_with_highest_priority dequeue()` operations.

   true. insert and remove operations in AVL take O(log n) time where priority is the value based on which the items are being ordered

5   Application (15 marks)

There are 20 clubs in the current English Premier League (EPL). Some notable teams are Manchester City (last year's winner), Manchester United, Chelsea, Arsenal, Liverpool, etc. You conducted a survey to n football fanatics and ask them which team will be in the top 4 this year (the order of the 4 clubs does not matter in your survey). The input given to your program is therefore n lines of 4 strings each. Each string (team name) has no more than 20 alphabet characters.

Let's define the *popularity* of a top 4 configuration to be the number of football fanatics selecting exactly the same combination of top 4 clubs. A combination of top 4 clubs is considered most popular if there is no other combination that has higher popularity.

Now you want to know the total number of football fanatics who pick some combination of top 4 clubs that is most popular.

Example 1: There are $n = 4$ respondents to your survey as shown below:

1. **ManCity - ManUtd - Chelsea - Arsenal**

2. ManUtd - Chelsea - ManCity - Liverpool

3. **ManCity - ManUtd - Chelsea - Arsenal**

4. **ManUtd - ManCity - Arsenal - Chelsea**

The combination of top 4 clubs that is the most popular in your survey is: 'Man City - Man Utd - Chelsea - Arsenal' and its 4! $= 24$ possible permutations – highlighted in **bold** above. This combination is picked by 3 football fanatics. Therefore, we answer 3.

Example 2: There are $n = 5$ respondents to your survey as shown below:

1. **ManCity - ManUtd - Chelsea - Arsenal**

2. *ManUtd - Chelsea - Arsenal - Liverpool*

3. **ManUtd - ManCity - Arsenal - Chelsea**

4. *ManUtd - Arsenal - Liverpool - Chelsea*

5. Chelsea - Swansea - WestBrom - ManCity

The combination of top 4 clubs that is the most popular in your survey are both: 'Man City - Man Utd - Chelsea - Arsenal' and its permutations – highlighted in **bold** above and 'Man Utd - Chelsea - Arsenal - Liverpool' and its permutations – highlighted in *italic* above. Each of these two popular combinations is picked by 2 football fanatics. Therefore, we answer 2+2 = 4.

There are different possible solutions for this non-original problem (the problem source will be revealed after Quiz 1). All possible solutions just require algorithmic knowledge taught so far. Your solution will be awarded different marks based on the criteria below:

Max Marks	Requirement
4	Partial marks for incomplete solution
7	An $O(n^2)$ solution, should be able to solve test cases with $1 \leq n \leq 1,000$
10	An $O(n \log n)$ solution, should be able to solve test cases with $1 \leq n \leq 100,000$
15	An $O(n)$ solution, should be able to solve test cases with $1 \leq n \leq 1,000,000$

Please come up with the *best possible* solution and analyze its time complexity.

You just need to outline your ideas (use pseudo codes) to answer this question.

Note 1: You are only given a small amount of space below (i.e. do **not** write too long-winded answer)!

Note 2: Do **not** attempt this question unless you have completed the other questions.

this is same as conformity. we first sort it in order of name and then count the frequency of each combination and then summing up the total of the top frequencies using hashtable

– End of this Paper –

Candidates, please do not touch this table!

Question	Maximum Marks	Student's Marks
1	14	
2	30	
3	26	
4	15	
5	15	
Total	100	