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: Selection algorithm (find k-th element) with quick-sort's partitioning
   Time Complexity: Expected O(n), discussed in Lecture01/Tutorial01

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: Simply report A[k-1]
   Time Complexity: Obviously O(1)

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:
   Ask the parent of this vertex to take over the only child of this vertex.
   Third, if the deleted vertex has two children, we:
   Replace this vertex with its successor and delete the original successor.

4. If $n = 1024$, then $\log_2 n =$ 10.
   If $n =$ $2^{20} = 1048576$, then $\log_2 n = 20$.

5. A perfectly balanced BST with n items has height as low as $\Omega(\log n)$, or just $\log n$.
   A totally unbalanced BST with n items has height as tall as $O(n)$, or just n.

6. List down two real-life examples where priority queue are used:
   Please show a case when the *priority* matters!
   1. Air Traffic Controller, Lecture 04, aircraft with more urgent situation can land first
   2. Scheduling deliveries, PS2, woman who is nearer to give birth is given higher priority

7. The time complexity of Heap Sort is Equal/On par/the same compared to Merge Sort.
   However, the time complexity of Heap Sort is Better/Smaller/Faster 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:

```
     10
    /  \
   7    \
  / \    \
 /   \    \
1    8    20
 \          \
  2          30
   \        /
    3      24
             \
             25
```

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:

```
         10
      /      \
    7         24
   /  \      /  \
  2    8  20    30
 / \           /
1   3         25
```

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:

```
      15
     /  \
   11    19
   /
  1
```

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	4	5	6	7	12	13	11

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:

```
              99
         /         \
       37            57
    /     \         /    \
  -5      34     -24    39
  / \     /
-14 -7  10
```

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:

```
              99
         /         \
       39            57
    /     \         /    \
  37      34     -14    -24
  / \     /
-5  -7  10
```

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:

```
inorderSpecial(BSTVertex T)
  if (T == null) // important check
    return
  inorderSpecial(T.right) // go to the right first
  if (T.left != null and T.right != null) // this node has two children
    print T.key
  inorderSpecial(T.left) // and finally go to the left

// marking scheme that is eventually used:
// start with 12 marks
// minus 3 marks for doing left and right in the wrong order
// minus 3 marks for not checking if T == null
//   (problem with recursion if this base case is not checked)
// minus 3 marks for not checking whether T has two children before printing T.key
// minus 3 marks for using O(n log n) solution
// minus 3 marks for other recognized bug
```

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:
Calling `enqueue(item, priority)` where priority is a constant value may not work 100 percent as the dequeue operation can return any of it. Therefore, we use a global counter to set the priority of each item with decreasing priority value for every enqueue.

I will implement `dequeue()` by:
Calling `dequeue()` as per normal.

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:
Use a global counter c that is initially set to 0. Every time I insert an item into priority queue, I call `enqueue(item, c)` and then increase c by one, i.e. $c = c + 1$. This way, items that are inserted later will have higher priority and therefore it will be dequeued first below by the priority queue, making it has the LIFO behavior. This is essentially the reversed question from Section 3.2.A above.

I will implement `pop()` by:
Calling `dequeue()` as per normal.

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. x has two children, that means it has at least a right children (which is definitely bigger than x – a potential successor). If the right subtree of x only contains one vertex, then this right children of x is the successor. Otherwise, the minimum of the right subtree of x is the successor of x.

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. Let $\{a, b, c\}$ be the three distinct integers where $a < b < c$. In a balanced BST according to AVL tree property, b must be the root, a must be on the left subtree, and c (the largest integer) must be on the right subtree, no matter what is the order of insertions.

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.

   False. Try this Binary Heap: {NIL, 4, 3, 1, 2}. The third largest integer, which is 2, is not the children of the root (4), but grand children of the root ($4 \rightarrow 3 \rightarrow 2$).

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

   True. The smallest element cannot have children (an even smaller element) in a Binary Max Heap as it will violate Max Heap property. Therefore the smallest element must be 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. This is the solution of this semester's PS2R. We can view enqueue operation as an insertion to a balanced BST, $O(\log n)$. We can view the dequeue operation as a finding and then deleting the minimum (or maximum) element in the balanced BST, $O(\log n)$.

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 $1 \le n \le 1,000$
10	An $O(n \log n)$ solution, should be able to solve test cases with $1 \le n \le 100,000$
15	An $O(n)$ solution, should be able to solve test cases with $1 \le n \le 1,000,000$

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.

At least 4 marks if students sort the 4 strings per each input line first so that the order does not matter. Sorting part is considered a constant factor as $4 \log 4 = 8$ of mergesort or $4^2 = 16$ of bubblesort are considered small constants.

7 marks solution: Use plain array to keep the frequency of a sorted combination and causes the runtime to be $O(n^2)$.

10 marks solution: Use a *balanced* BST (e.g. TreeMap) to keep track of the frequency of a sorted combination – maybe a QuartetObject of 4 strings – in $O(n \log n)$. Later, we go through the BST in $O(n)$ to get the highest frequency. Finally, we go through the BST again in $O(n)$ to sum items with frequency equals to the highest frequency. We output this counter. This uses CS2010 knowledge. PS: There is no need to use Max Heap for this question.

15 marks solution: We map each string (team name) to indices of [0..19] first. After sorting, we can hash these values into a small hash table of size ≈ 10000 as there are only ${}_{20}C_4 = 4845$ combinations of top 4 teams (a slightly larger table is preferred to minimize collisions). Or we can also use this 'base 20' direct addressing hash formula $team[0] * 20^3 + team[1] * 20^2 + team[2] * 20 + team[3]$ and use a Direct Addressing Table to count the frequency. As there are only $20 * 20 * 20 * 20 = 160000$ different top 4 combinations, an array of size 160000 is sufficient. Yet another alternative is to use bitmask of $2^{20} = 1048576$ potential combinations and again use Direct Addressing Table. All these solutions run in $O(n)$. This actually uses CS1020 knowledge.

Credit goes to the problem setters at University of Waterloo for setting this problem:
http://uva.onlinejudge.org/external/112/11286.html

– End of this Paper –

Candidates, please do not touch this table!

Question	Maximum Marks	Average Student's Marks
1	14	11.2
2	30	26.9
3	26	20.3
4	15	11.7
5	15	7.3
Total	100	77.5