National University of Singapore

School of Computing

Semester 1 (2015/2016)

CS2010 - Data Structures and Algorithms II

Written Quiz 1 (10%)

Saturday, September 19, 2015, 10.00am-11.30am (90 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 3 adjacent venues: LT19 ['A'..'P'] (111 students, capacity 200), SR@LT19 ['Q'..'U'] (37 students, capacity 42), and TR9 ['W'..'Z'] (31 students, capacity 32).
3. This question paper contains THREE (3) sections with sub-questions.
   It comprises EIGHT (8) 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/Tutor Name/Monday or Tuesday:** _____
   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 Tutor on Week07.

---

1 Analysis (15 marks)

Prove (the statement is correct) or disprove (the statement is wrong) 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. In `ShiftDown(i)` operation of Binary (Max) Heap, there is this pseudocode: "if A[i] $<$ than the larger of its children, swap A[i] with that child". Professor ABC claims that it is OK to change this pseudocode into: "if A[i] $<$ than any of its children, swap A[i] with that child".

   Professor ABC is wrong. This is easy and already discussed in tut01. A simple counter example: Use three integers: {1, 2, 3}, with 1 as the root, 2 as the left child, and 3 as the right child. If now we swap 1 with any of the children greater than 1, then we may end up swapping 1 with 2. Binary (Max) Heap property is still violated between 2 (new root) and 3 (it's right child).

2. There is an AVL Tree with $n = 1000$ vertices that has height $h = 30$.

   Impossible. The loose analysis of AVL Tree presented in Lecture 04 shows that $h < 2 * \log_2 n$ (a tighter analysis is $h < 1.44 * \log_2 n$). That means if $n = 1000$, at most the height of the worst possible AVL Tree is just $2 * \log_2 1000 \approx 2 * 9 = 18$.

3. The 'union-by-rank' and 'path-compression' heuristics of the Union-Find Disjoint Sets data structure are not that important. We can still achieve the $O(1)$ performance if we are only dealing with up to 1M (1 Million) items.

   Wrong. A simple counter example: Start with $n = 1M$ disjoint sets, then union item 1..999999 to item 0 one by one without using 'union-by-rank' and 'path-compression' heuristics. Then, we will end up with a very tall tree of height 999999. Then, find set operation will no longer runs in $O(1)$ but 1M, or $O(n)$ steps... Too slow.

4. An integer is stored in binary in computer memory. Therefore we can use this sequence of 0s and 1s to represent a medium-size set of **10000** Booleans very efficiently and can manipulate that integer via various $O(1)$ bit manipulation operations (e.g. '&', '|', '<<', '>>').

   Not possible with current computers. Bit manipulation operations only works with built-in integer data type. The largest one in typical modern programming languages including Java is 64-bit, or 64 Booleans. 10000 Booleans is too much. We need to resort to the standard array, ArrayList, or Vector of Booleans which has much slower performance $O(n)$ per operation – n is the number of bits involved – than various $O(1)$ bit manipulation operations on built-in integer data type.

5. An adjacency matrix representation M of a graph G is symmetrical, i.e. $M = M^T$. Therefore, graph G must be an Undirected Graph.

   Yes. If edges (i, j) and (j, i) are both present in M for all pairs of (i, j), then all edges in the graph must be bidirectional, i.e. undirected.

2 Not Yet in VisuAlgo Online Quiz (50 marks)

2.1 `Extract(i)` Method of a Binary (Max) Heap (10 marks)

In PS1, you were introduced with the `Extract(i)` method of a Binary (Max) Heap (disguised as `GiveBirth(womanName)` method). One possible implementation of this method is:

```
Extract(i) // i is the index of the element to be extracted
  swap(A[i], A[heapsize])
  heapsize--;
  ShiftUp(i);   // line 3, fix potential (Max) Heap property violation upwards
  ShiftDown(i); // line 4, fix potential (Max) Heap property violation downwards
```

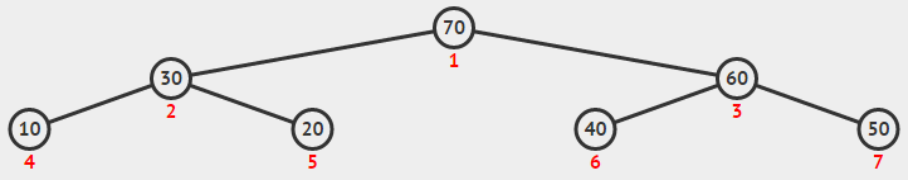

Figure 1: The Starting Binary Max Heap

Now your task is this: Given Binary Heap structure of $n = 7$ integers: {70, 30, 60, 10, 20, 40, 50} as shown in Figure 1 (note that the root 70 is at index 1, not at index 0), list down all indices $\in$ [1..7] so that if `Extract(i)` is called:

1. (3 marks) `ShiftUp(i)` in Line 3 triggers some swaps but `ShiftDown(i)` in Line 4 does not
   Indices {4, 5}. Many students missed this case in their first few submissions of PS1.
2. (2 marks) `ShiftDown(i)` in Line 4 triggers some swaps but `ShiftUp(i)` in Line 3 does not
   Indices {1}.
3. (3 marks) Neither `ShiftUp(i)` in Line 3 nor `ShiftDown(i)` triggers any swap
   Indices {2, 3, 6, 7}. Yes this case is possible.
4. (2 marks) Both `ShiftUp(i)` in Line 3 and `ShiftDown(i)` trigger some swaps
   No answer, it is impossible for this case to happen.

2.2 AVL Tree Insertion Balancing Act Challenge (10 marks)

Give a sequence of insertion of this set of $n = 7$ integers {1, 2, 3, 4, 5, 6, 7} into an initially empty AVL tree so that **no** rotation occurs at all. Please explain why that insertion order does not trigger any rotation.

The answer is to always insert the median of the sublist first so that it becomes the root of the subtree: 4, {2 or 6} next, then {1, 3, 5, or 7} in any order next. This way, the balance factor for any vertex is always -1/0/1. Try this at `http://visualgo.net/bst.html?mode=AVL`, click 'Create → Empty', and insert {4, 2, 6, 1, 3, 5, 7} (yes, you can insert more than one numbers one after another in VisuAlgo) to convince yourself that no rotation happens.

2.3 AVL Tree Deletion Domino Effect Challenge (10 marks)

In Lecture04 and tut03, you have been presented with this example (see Figure 2) where deleting vertex 7 causes two groups of rotations to be performed, that is: rotateRight(6)—the first group followed by rotateRight(16)+rotateLeft(8)—the second group. Notice that the second group consists of two individual rotations but counted as one group.

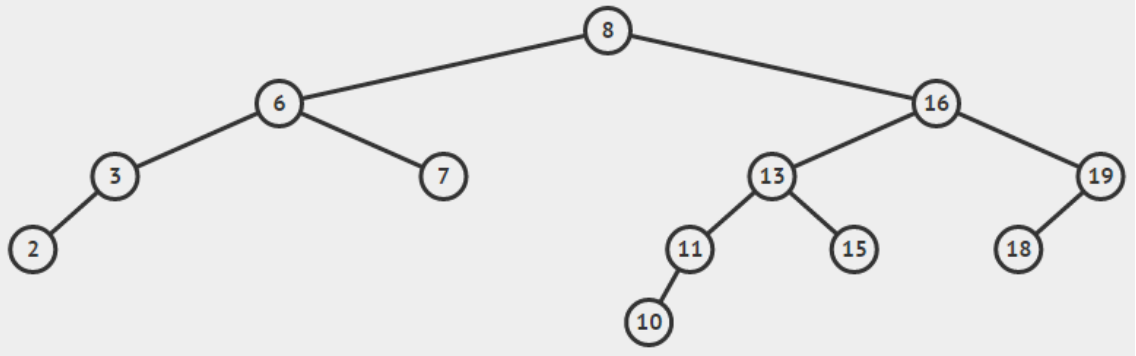

Figure 2: The Example AVL Tree

Now your task is simply this: Draw **any** AVL tree structure of n integers with value $[1..n]$, nominate a vertex between that range $[1..n]$ so that if that vertex is deleted, **THREE** groups of rotation(s) happen. To facilitate easier grading, mention what are those three groups of rotation(s) are.

This is a super hard question to answer that worth $\frac{10}{100} \times 10\% = 1\%$ of your CS2010 grade.
Visit `http://visualgo.net/bst.html?mode=AVL&create=21,13,29,8,18,26,32,5,11,16,20,24,28,31,33,3,7,10,12,15,17,19,23,25,27,30,2,4,6,9,14,22,1` Then, we delete vertex 33 and observe the cool domino effect :). The basic idea is that this AVL tree is basically the minimum size AVL tree of height 5 with $n_5 = 33$ vertices. It is almost unbalanced in every vertex and by deleting vertex 33, we triggers this domino effect :).

Steven predicts that not many students will be able to come up with this answer as it involves a staggering 33 :O vertices although the demo AVL tree in Figure 2 (that triggers two group of rotations after vertex 7 is deleted) is actually $n_4 = 12$ vertices (not labeled from 1 to 12 though).

2.4 UDFS Min/Max Tree Height Challenge (20 marks)

In Lecture05, you have learned about Union-Find Disjoint Sets (UFDS) data structure that uses both 'union-by-rank' and 'path-compression' heuristics. Given an UFDS structure that currently has 4 disjoint sets as shown in Figure 3 below.

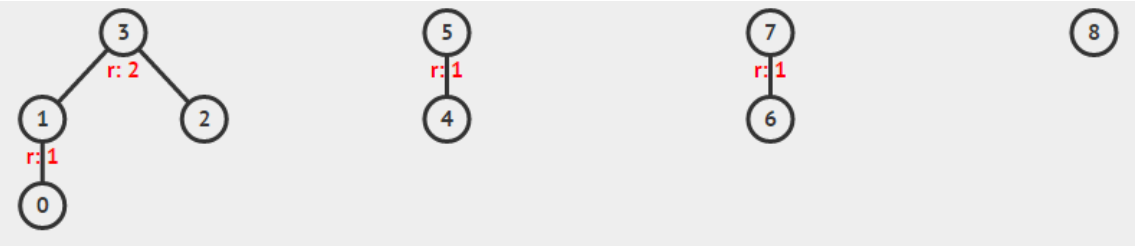

Figure 3: The current UFDS structure with 4 disjoint sets

Your task is to **perform exactly three** `unionSet(i, j)` operations so that there is only one set left and **draw the resulting tree**. There are two conditions that you have to fulfill.

1. (10 marks) The resulting tree is as tall as possible
   My answer: Do `unionSet(____, ____)`, `unionSet(____, ____)`, and then `unionSet(____, ____)`
   My resulting tree:

2. (10 marks) The resulting tree is as short as possible
   My answer: Do `unionSet(____, ____)`, `unionSet(____, ____)`, and then `unionSet(____, ____)`
   My resulting tree:

Baseline: `http://visualgo.net/ufds.html?p=1,3,3,3,5,5,7,7,8&rank=0,1,0,2,0,1,0,1,0`

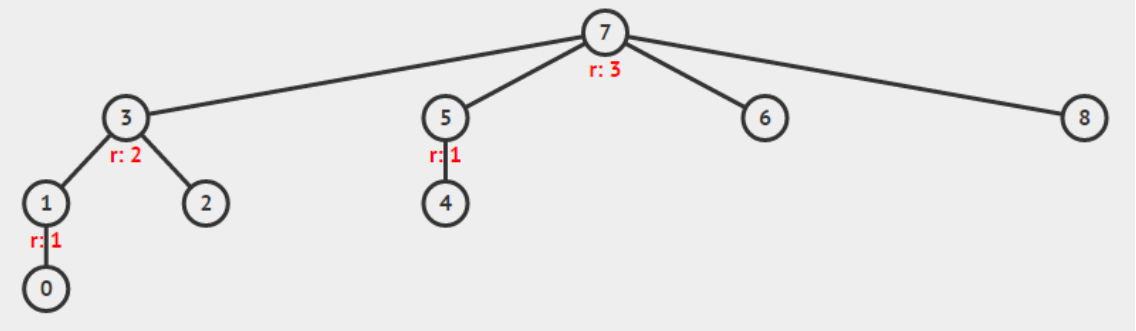

Figure 4: The Tallest Possible Tree (Height 3)

To get a single tree that is as tall as possible with the two UFDS heuristics applied, we need to union 2 subtrees that have equal rank (so that it's height increases) and when we choose two items to union, we need to select the root or at least the child of root so that 'path-compression' heuristic does not shorten the resulting tree. Here is the way to have a tree that is as tall as possible (height 3):
1. `unionSet(5, 7)` or combination of these 4 vertices from these 2 subtrees to create 1 subtree with rank 2. No path compression can be triggered with this action (yet),
2. Then the second `unionSet` must be between the resulting subtree from the step above (but select

vertex that will not trigger path compression) with subtree with root 3 (we cannot use vertex 0 as it will trigger path compression).
3. Lastly, we union the resulting subtree with rank 3 (actual height 3) with the single vertex 8 which will not increase the height of the subtree any further. Actually we can union the single vertex 8 with any other 3 subtrees earlier, it will not matter as 'union-by-rank' heuristic will ensure the resulting tree remains short.
Other valid answers are accepted.

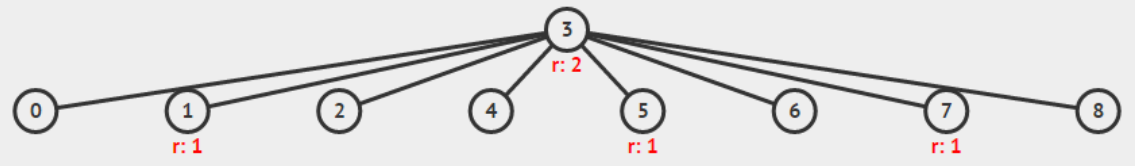

Figure 5: The Shortest Possible Tree (Height 1)

To get a single tree that is as short as possible with the two UFDS heuristics applied, we need to union the shorter trees to the currently tallest tree (based on rank value) and purposely triggering path-compression during union. Here is the way to have a tree that is as short as possible (height 1):
1. `unionSet(0, 4)` or `unionSet(0, 5)`, triggers path compression of vertex 0,
2. `unionSet(4, 6)` or `unionSet(0, 7)`, triggers path compression of vertex 4,
The first two operations above are interchangeable, then finally
3. `unionSet(6, 8)`, triggers path compression of vertex 6 :). This vertex 8 has to be last, otherwise it is impossible to have final tree of height 1 (rank 2, but actual height 1).
Answer with 1 tree of height 2 will get 5 marks only but other valid answers are accepted.

3 Think Outside the Box (35 marks)

3.1 CS2010 Leaderboard (20 marks)

You must have visited URL: `http://www.comp.nus.edu.sg/~stevenha/cs2010.html#leaderboard` at least once during this semester and possibly many more times (maybe every week) to monitor where you are in class :).

This leaderboard can be abstracted as a data structure that contains n triples of information: `(studentname, onlinequizscore, psscore)`. Two of the three fields `(onlinequizscore, psscore)` are **frequently updated** throughout the semester whereas field `studentname` is static. Consider that CS2010 is offered to worldwide audience (it is a MOOC – Massive Open Online Course) so that there are $n = 999999$ students enrolled (that is, 1 Million minus 1 students), an odd number. Steven wants to build an efficient leaderboard that can perform all these methods efficiently:

1. `UpdateScore(studentname, newonlinequizscore, newpsscore)`
   As the method name implies, this method updates the `onlinequiz` and `psscore` of `studentname` to new values. If `studentname` does not exist before, we create a new triple.

2. `GetBest(type)`
   This method returns the `studentname` who has the best score, depending on attribute type:
   (a) if `type = 0`, return the best `studentname` with the highest `onlinequizscore`
   (b) if `type = 1`, return the best `studentname` with the highest `psscore`
   (c) if `type = 2`, return the best `studentname` with the highest `(onlinequizscore+psscore)`

3. `GetMedian(type)`
   This method returns the `studentname` who has the median score, depending on attribute type:
   (a) if `type = 0`, return the best `studentname` with the median `onlinequizscore`
   (b) if `type = 1`, return the best `studentname` with the median `psscore`
   (c) if `type = 2`, return the best `studentname` with the median `(onlinequizscore+psscore)`

Using your best knowledge from CS2010, advise Steven how to implement his leaderboard system.

Use **THREE** balanced BSTs that is already augmented with size attribute to store `studentname` keyed based on `onlinequizscore`, `psscore`, and `(onlinequizscore+psscore)`, respectively. Then `UpdateScore` entails three deletions of old data (if any, see below for the need of the FOURTH balanced BST) and reinsertion of updated data, `GetBest` entails returning the max value of the correct bBST, and `GetMedian` is selecting the element with rank $(n/2)$ in the correct bBST using the Select operation as discussed in tut03 :).

One more, we need to use the **FOURTH** balanced BST to map `studentname` to pair of `onlinequizscore`, `psscore`, as we need this information to be able to delete the old information correctly from the other **THREE** balanced BSTs (remember that the other three balanced BSTs are NOT keyed using

`studentname`, they are keyed using respective scores).

Using only a single balanced BST is not OK as we cannot handle three different keys in a single bBST.

PriorityQueue is only OK for `GetBest` but it has problem with `GetMedian` query.

Final Grading Scheme:

1. More students got this question correct, i.e. use 3+1 copies of balanced BSTs (AVLs) or 3 AVLs + 1 HashMap. I have a feeling that is is about half, 1 correct every 2 scripts checked, or 50% of the class. This is much better compared to a mere 15-20% correct in Section 3.2. The most probable reason is because this Section 3.1. is basically to test if you did your PS1 and PS2 all by yourself and have mastered all the required techniques used in those two PSes.
2. The most common mistake is forgetting that as the 3 bBSTs are keyed based on scores, you need the 4th bBST (or a HashMap) to quickly map `studentname` to it's current `onlinequizscore` and `psscore`, otherwise you cannot update them efficiently.
3. Another interesting mistake is to only use 2 bBSTs, one for `onlinequizscore` and another `psscore`. For `GetBest(2)`, the student reports the name of student that has the best `onlinequizscore` + the best `psscore`... Beware that this is a Wrong Answer as the student with best overall `onlinequizscore+psscore` does not always be the best in both individual fields.
4. Some (minority) of students use Priority Queue (Binary Heap). While this can give you an $O(1)$ `GetBest(type)` performance, it has problem with `GetMedian(type)`. Another (minority) of students use balanced BSTs, but for `GetMedian(type)` operation, they don't use the Select operation discussed in tut03 but rather do inOrder traversal up to (n+1)/2 steps... This is also slow :(. Yet another (minority) students say that median element of a BST is always at the root, so $O(1)$... Think again and convince yourself that this is a 'Wrong Answer'...
5. Note 1: You can assume that the `studentname`s are distinct, and in the event Steven a student name that actually refers to two different student, he can append id, e.g. 'Steven1' and 'Steven2' so that the names remain distinct.
6. Note 2: In the original question, Steven forgot to specify what to return in the event of multiple `studentname`s with the same best/median `onlinequizscore`, `psscore`, or `onlinequizscore+psscore`. With hindsight, Steven should have written: 'If there is a tie, break ties by returning `studentname` with the smallest lexicographical order, like the current leaderboard system'. All assumptions about this issue are accepted during grading.
7. Note 3: There is a possible other inferior method to answer `GetMedian(type)` that involves calling successor/predecessor of the element with rank (n+1)/2 during each update. This is not as flexible as Select operation that can select item of any rank.
8. General grading scheme: 20 marks if all 3 operations are perfect, 15 marks if 1 of the 3 operations have problem(s), 10 marks if 2 of the 3 operations have problem(s), 5 marks: Mercy marks...

3.2 Graph Data Structure (15 marks)

What is the best graph data structure to store graphs with characteristics like the example shown in Figure 6 below: Near complete graphs of V vertices ($5 < V < 100000$) with only up to k edges missing ($0 \leq k \leq 7$)?

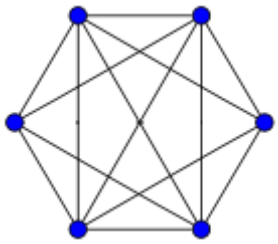

Figure 6: Near Complete Graph with $V = 6$ Vertices (One Edge is Missing)

This is also a challenging question that worth $\frac{15}{100} \times 10\% = 1.5\%$ of your CS2010 grade.
Using standard, text book Adjacency Matrix, Adjacency List, or Edge List will cause us to put in up to $O(n^2)$ edges inside the chosen data structure... However, if we just think out of the box a bit and store the list of edges **that are not present** in the graph, then we just need to store one integer n itself and those small number of k ($0 \leq k < 7$) edges in an Edge List :O... This information is enough to preserve the graph structure. You can do *(some amount of)* graph traversal (DFS/BFS) by assuming all edges are present except edges that are in this 'not-present Edge List'. We can't really explore the entire graph well as there are n^2 edges there though.

Final Grading Scheme:

1. Blank or 'just tikam' answer will get 0 or 1 mark, respectively.
2. The most common mistake is thinking that $O(V + E)$ of AdjacencyList or $O(E)$ of EdgeList is better than $O(V^2)$ AdjacencyMatrix in a near complete graph. It is not, as $E = O(V^2)$ in a near complete graph, i.e. all three are equally bad if used verbatim. Such confusion in the answer will get 3 marks.
3. The slightly better 5 marks is given to those who compare all three taught Graph Data Structures and then realizes all are $O(V^2)$ and then decide on one of them for whatever reason, e.g. select AdjacencyMatrix because he/she assumes that most frequent operation is checking existence of edge(u, v), or select AdjacencyList as it may run in $O(V - k)$ if that vertex has k less neighbors, etc. But this is not the best answer.
4. The best answer is 'storing the reverse'. ReverseAdjacencyMatrix won't be helpful as it is still $O(V^2)$. ReverseAdjacencyList is better but not the best as we need $O(V + k)$ (or $O(V)$ as k is very small). ReverseEdgeList is the best as we only need $O(k)$ (or $O(1)$ as k is very small) to store the graph information.

5. Note 1: You can assume that the vertices are labeled from 0..V-1 and if it is not, you can always relabel it to 0..V-1 so that you can memorize a complete graph by just storing one integer: V, this is an $O(1)$ space.

6. Note 2: The given graph in Figure 6 is an unweighted graph. If it is weighted, I don't think I have a good answer either.

7. This kind of 'top IT company interview question' really reveals who has the out-of-the-box brain. It has never appeared before in lectures, tutorials, lab demos, PSes... Brace for such questions in WQ2 and final exam too :). Usually it appears as my 'last question' in the test.

– End of this Paper –

Candidates, please do not touch this table!

Section	Maximum Marks	Average Marks	Comments from Grader
1	15		11.51 ± 2.54
2	50		33.13 ± 10.34
3	35		19.33 ± 8.14
Total	100	61.00 ± 19.54	