CS251 Homework 3: Sorting, Trees, and Graphs

Out: February 16, 2018 @ 9:00 PM

Due: March 2, 2018 @ 11:59 PM

Submission Instructions: Please submit a typeset PDF on blackboard. For multiple choice questions, you must provide an explanation along with your answer. Answers without explanations will receive 0 points, even if correct.

Sorting:

1. What would be the complexity of Quicksort if the pivot is always the median? (Assume the median can always be found in linear time).
   a. Always O(n log n)
   b. Always O(n^2)
   c. Worst case: O(n^2), average case: O(n log n)
   d. Worst case: O(n^2 log n), average case: O(n^2)

The answer is A. Because if we know that the pivot is always the median then we know the worst case can never occur. The worst case can only occur if the quicksort was already sorted and that the pivot was the leftmost element of the partition. Since this cannot happen with the pivot being the median, the time complexity will always be the average case which is O(n log n).

2. Which of the following statements about sorting is incorrect?
   a. Insertion sort is more efficient on a partially sorted array.
   b. Mergesort has the best worst-case time complexity for comparison-based sorting algorithms.
   c. Radix sort's running time depends on the keys and base.
   d. For large n, Heapsort is less efficient than insertion sort.

The answer is D. This is because the time complexity for Heapsort in worst case is O(n log n), and insertion sort is O(n^2) which is not as good as heapsort. This means that even for a large n of 1 million, heapsort would be 6 million and insertion sort would be 1 trillion. So heapsort is more efficient than insertion sort for large n and the statement of D is incorrect.

3. Given a pre-sorted array of unique integers, what is the expected running time of Quicksort if the pivot is always chosen to be either the first or last element?
   a. O(n log n)
   b. O(n^2 log n)
   c. O(n^2)
   d. None of the above

The answer is C. Because the list is presorted then the first element would be the smallest and the last element would be the largest. If that is the case then the list would be divided into two sublists 0 and n-1. This would make it so every recursive call would take n-1 times and creates the worst case time of O(n^2).

4. What would be the complexity of Mergesort if the list is divided into 4 chunks instead of 2 each time?
    a. O(n log n)
    b. $O(n^2)$
    c. $O(n^{3/4})$
    d. $O(n \log^2 n)$

The answer is A. This is because dividing into 4 chunks would give it the same time complexity as 2 chunks. This is because in both cases there would be the same number of comparisons and therefore they would have the same time complexity of O(n log n).

5. Show the sorting steps of Radix sort for the following input:
    25, 57, 48, 37, 12, 92, 86, 33

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Pass 1			**12**	**33**		**25**	**86**	**57**	**48**	
			92					**37**		
Pass 2		**12**	**25**	**33**	**48**	**57**			**86**	**92**
				37						

6. Trace the Mergesort algorithm on the following array of integers. Be sure to show all split and merge steps.
    23, 39, 59, 26, 30, 62, 27, 83

    1. **(23, 39, 59, 26, 30, 62, 27, 83)**
    2. **(23, 39, 59, 26) (30, 62, 27, 83)**
    3. **(23, 39) (59, 26) (30, 62) (27, 83)**
    4. **(23) (39) (59) (26) (30) (62) (27) (83)**
    5. **(23, 39) (26, 59) (30, 62) (27, 83)**
    6. **(23, 26, 39, 59) (30, 27, 62, 83)**
    7. **(23, 26, 27, 30, 39, 59, 62, 83)**

Trees:

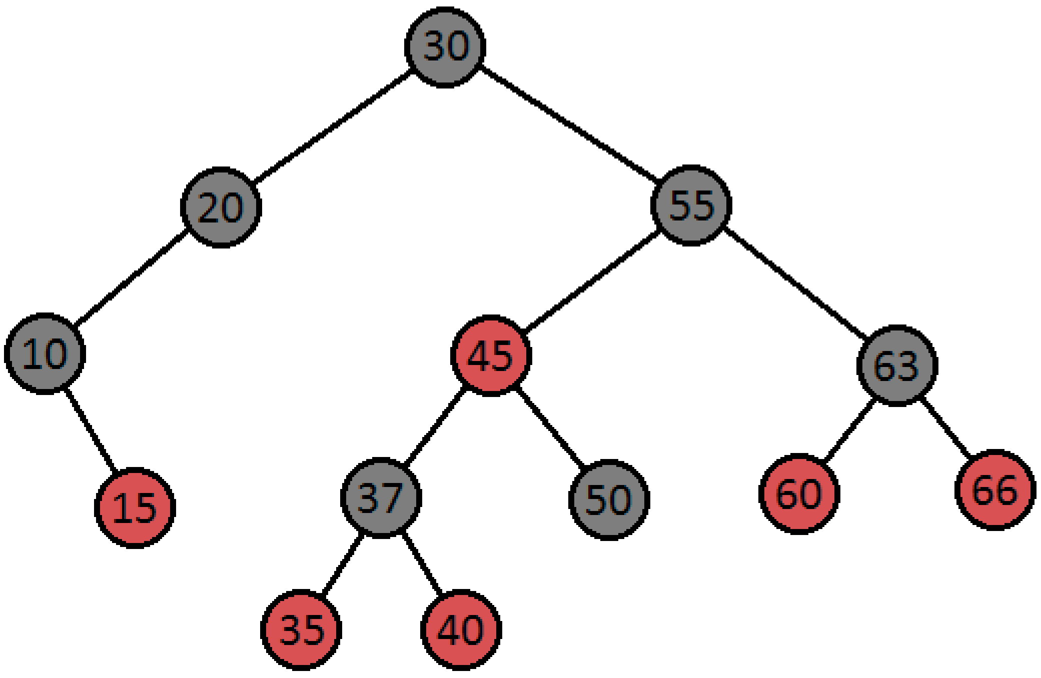
7. Which of the following statements about trees is most accurate?
   a. The purpose of AVL trees is to efficiently implement search algorithms.
   b. The order of search algorithms in all binary trees is the same
   c. The depth of a binary tree is as most O(log n)
   d. The balance factor of AVL trees is 0 or 1.

The answer is C. This is because the height h of a complete binary tree can be shown with $n = 1 + 2 + 4 + \ldots + 2^{h-1} + 2^h = 2^{h+1} - 1$. Solving for h we get O(log n).

8. True or false: a Red-Black tree with 128 nodes has at least one red node.
   a. True
   b. False

The answer is A. The statement is true because whenever a node is inserted it is colored red and then changed later if needed. So if that is the case then there will always be at least one red node.

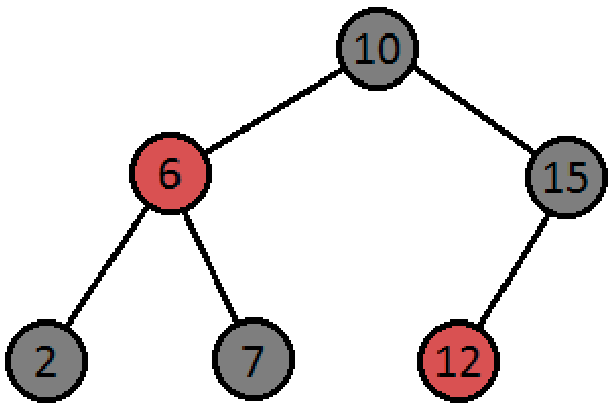
9. Is the following Red-Black tree (with leaf nodes hidden) valid? If not, which property is violated?



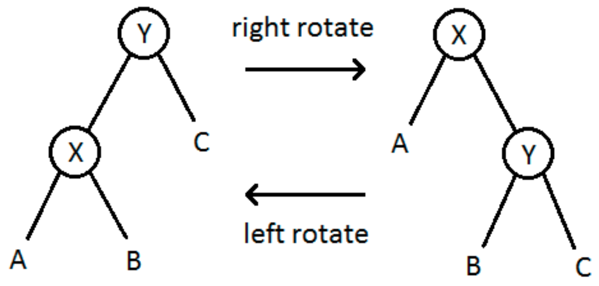
1. **The root is black? Yes**
2. **Each node is red or black? Yes**
3. **All leaves are black? Yes**
4. **A red node has both black children? Yes**
5. **All paths from a single node to any leaf is the same amount of black nodes? Yes**

No properties are violated so this must be a valid Red-Black tree.

10. Consider the following Red-Black tree (with leaf nodes hidden):



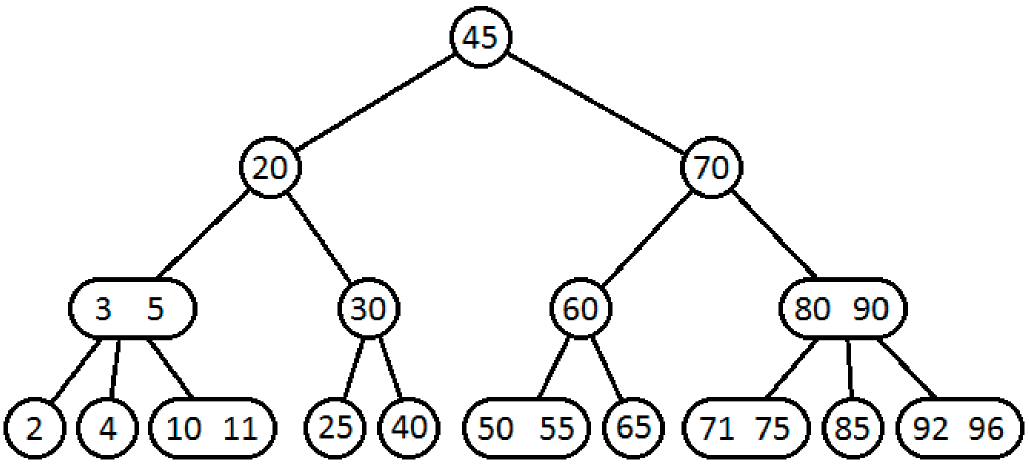
Suppose we want to insert the element 14. Which of the following is true?
For reference:

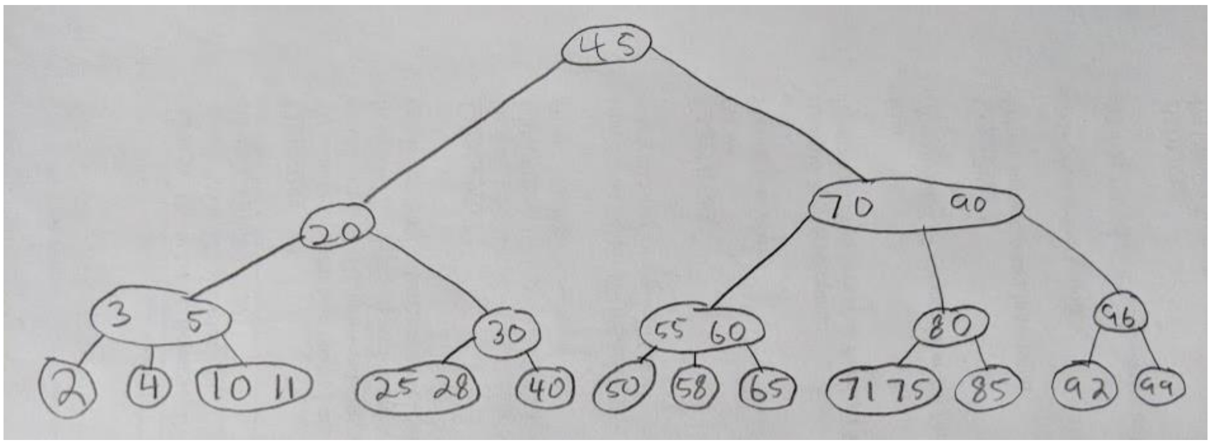


a. We must perform a recoloring only
b. We must perform a left rotation only
c. We must perform a right rotation only
d. We must perform a left rotation followed by a right rotation
e. The element can be inserted without any further action

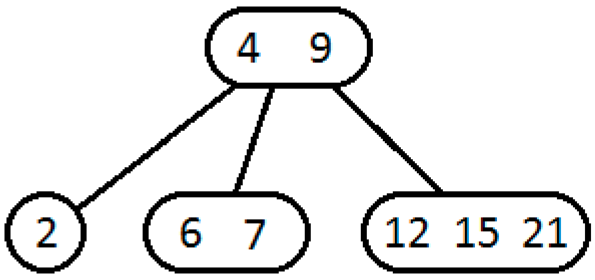
The answer is D. When we insert 14 it will be inserted as the right child of node 12. Since both are red and 14 is a right child of 12 and 12 is a left child of 15 we perform a left rotation. This makes 14 the left child of 15 and 12 the left child of 14. In order to fix the extra redness in the tree we do another rotation but to the right. Making 14 the parent and 12 as the left node and 15 as the right node. In turn this causes 14 to become black and 15 to become a red node.

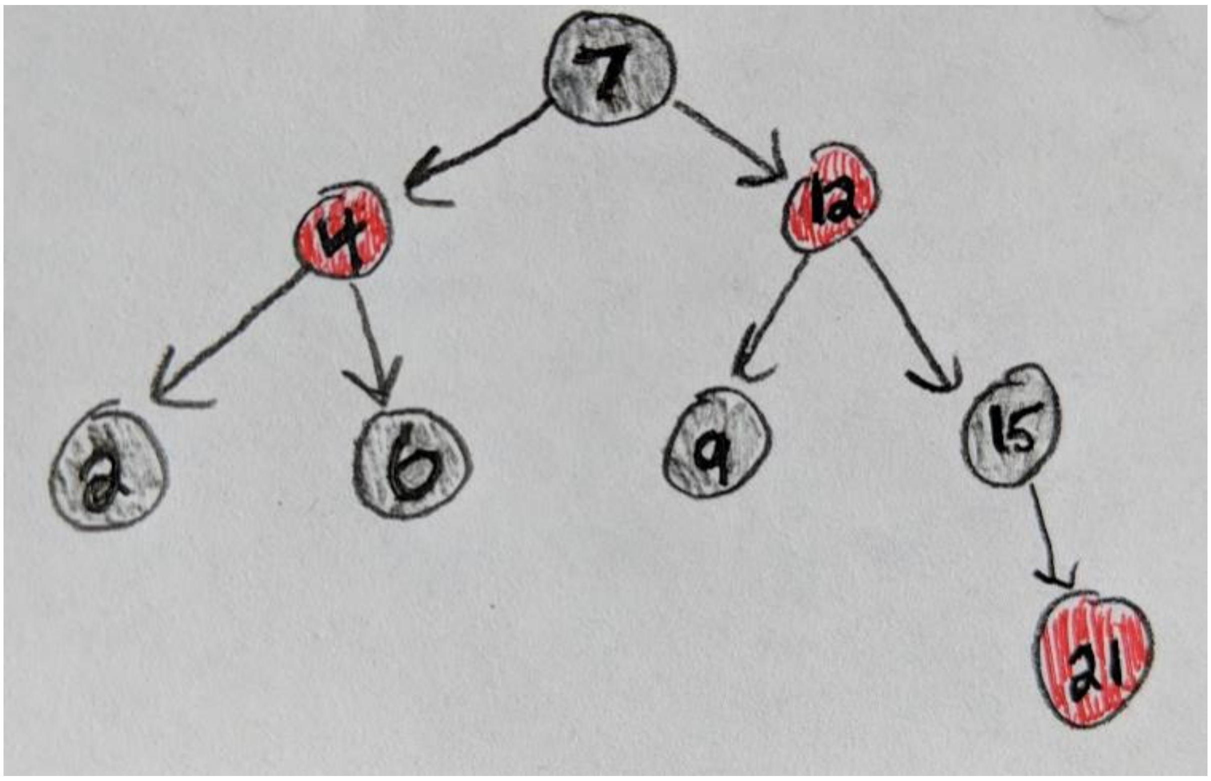
11. In the 2-3 tree given below (i.e., NOT a 2-3-4 tree), execute insert(28), insert(99), and insert(58), in that order, making sure to rebalance after each insertion. Draw the resulting 2-3 tree after executing these operations.





12. Given the following 2-3-4 tree, draw an equivalent Red-Black tree.





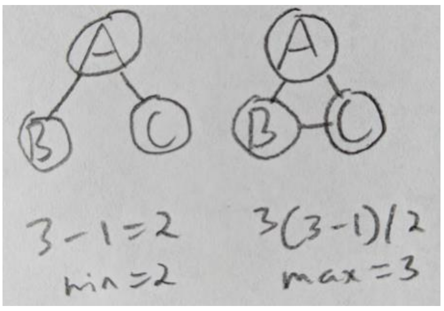
Graphs:

13. A graph G is an undirected graph with n vertices and each vertex has exactly degree d. Assume n is odd. What can you say about d?
    a. d must be odd
    b. d must be even
    c. d can be odd or even

    The answer is B. It must be even because if it is a simple graph then it would have a degree of n-1. Which if n is odd this would make d an even number.

14. Given a connected graph of n vertices and m edges, with no self-loops or parallel edges, the minimum and maximum number of edges of the graph is:
    a. n <= m <= 2 n
    b. n - 1 <= m <= n (n - 1) / 2
    c. n <= m <= n^2
    d. None of the above

    The answer is B. This can be shown with a very simple graph of 3 vertices. If there are 3 vertices then it can be connected in two different ways where it is connected by the minimum number of edges or the maximum number of edges. An example is illustrated below to prove my point.



15. Let G be an undirected graph with n vertices and m edges. If m >= n, then G has a cycle.
    a. Always true
    b. Sometimes true
    c. Never true

    The answer is A. Because if it has no cycle then by the Euler formula for the number of edges it would have n-1 edges which is a contradiction. If it has 2 cycles or more the number of edges would have to be greater than the number of vertices and if that is the case it would be true for us. Also if m = n then we would get a cycle as well. So it must always have a cycle if m >= n.

16. After running DFS on a graph G, how many of the following statements about the edge e=(u -> v) are correct?
    - e is a tree edge if and only if $d_u < d_v < f_v < f_u$ and while exploring the edge e, vertex v has not been visited yet.
    - e is a forward edge if and only if $d_u < d_v < f_v < f_u$ and while exploring the edge e, vertex v has been visited before.
    - e is a back edge if and only if $d_v < d_u < f_u < f_v$.
    - e is a cross edge if and only if $d_v < d_u < f_u < f_v$.

    d_x is a *discovery time* of vertex x, when it is first processed and f_x is a *finish* time of vertex x, when all of its descendants are finished.

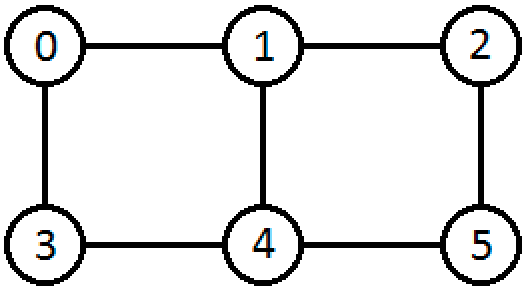
    a. 1
    b. 2
    c. 3
    d. 4

    The answer is C. The first point is true. The second point is true. The third point is true. The fourth point is false because it should be $d_v < d_u < f_v < f_u$. So with three being true the answer must be C.

17. A unicycle graph is a connected graph with exactly one cycle. For a given undirected graph G with n vertices and m edges, what is the fastest algorithm to find out if G is unicycle or not?
    a. Θ(n)
    b. Θ(m+n)
    c. Θ(n^2)
    d. Θ(mn)

The answer is A. This is because in order to find the cycles we must first do a DFS. In doing this it would take O(n) time to find a cycle with n vertex, since at most n-1 edges can be tree edges.

For questions 18, 19, and 20, consider the following graph G:



18. Write down the adjacency matrix for G.

0 1 0 1 0 0
1 0 1 0 1 0
0 1 0 0 0 1
1 0 0 0 1 0
0 1 0 1 0 1
0 0 1 0 1 0

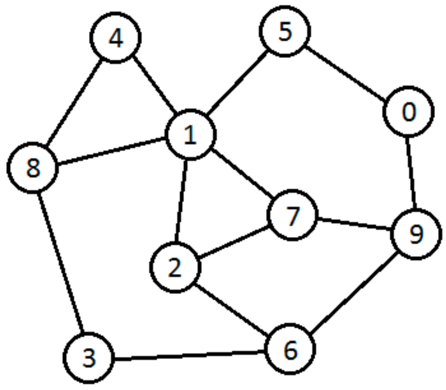
19. Compute the square of the adjacency matrix of G and explain what property of the graph each entry a_{ij} shows.

2 0 1 0 2 0
0 3 0 2 0 2
1 0 2 0 2 0
0 2 0 2 0 1
2 0 2 0 3 0
0 2 0 1 0 2

20. Consider the k-th power of the adjacency matrix (you do not have to compute it). What property of the graph does each entry a_{ij} show?

21. Which sequence corresponds to a BFS on the following graph, starting from vertex 7?

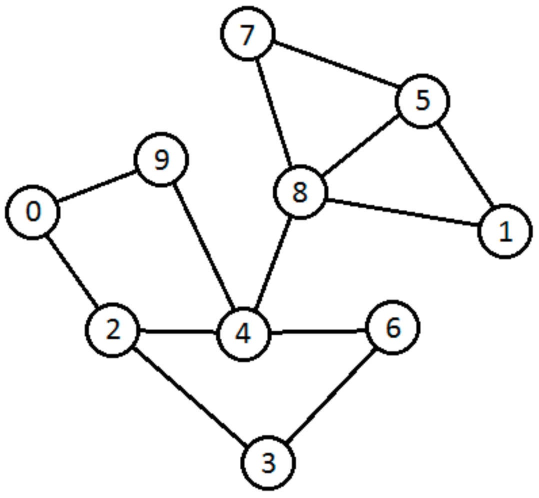
   a. 7, 1, 2, 9, 0, 4, 5, 6, 8, 3
   b. 7, 9, 1, 2, 6, 0, 5, 8, 4, 3
   c. 7, 2, 1, 9, 6, 5, 4, 8, 3, 0
   d. 7, 1, 2, 9, 8, 4, 0, 6, 3, 5



The answer is A. That is because the order of a BFS traversal is the closest and lowest value first. So the answer cannot be B or C because 1 is the same distance as 2 and 9 but it is a lower value so it would be entered first. This same reasoning is done for why A is correct over D. The point where they differ is the 5th entry where A is 0 and D is 8. Because at both points the path would be the same distance it goes to 0 because it is a lower value. So therefore the answer must be A.

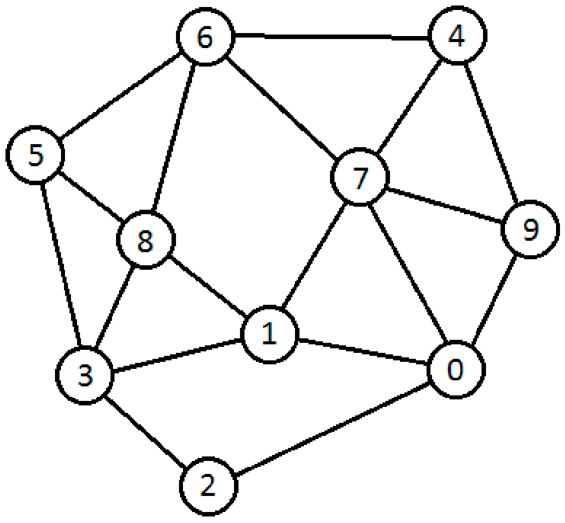
22. Which sequence does NOT correspond to a DFS on the following graph, starting from vertex 3?

   a. 3, 6, 4, 8, 5, 7, 9, 0, 2, 1
   b. 3, 6, 4, 9, 0, 2, 8, 1, 5, 7
   c. 3, 2, 4, 6, 8, 1, 5, 7, 9, 0
   d. 3, 2, 0, 9, 4, 6, 8, 7, 5, 1



The answer is A. Because it would have to trace over paths that were previously visited before taking paths that have not been. And by traversing these paths it would be terribly inefficient and incorrect.

23. Which vertices, if individually removed, would cause the following graph to no longer be biconnected? Choose all that apply.



a. 0
b. 1
c. 2
d. 3
e. 4
f. 5
g. 6
h. 7
i. 8
j. 9

The answer is none of the above. Because each vertex has more than one edge connected to it, by removing any one vertex it will not cause another to lose all its edges. So there would be no disconnect in the graph and so it would still remain biconnected. The closest vertex that could cause this would be to remove vertex 3 and 0. However by the definition this could never happen and so it is a biconnected graph no matter what individual vertex is removed.