**COMP1927 PAST EXAM**

**You better all collaborate!!!!**

**Did q1 -> 3 just apparate away o.0? q1->3 are coding questions in all the past exams[[1]](#footnote-0)**

# 15s1

**=== Question 4 ===**   
(2 marks)  
Which of the following statements are true?  
 .[A] Splay trees always have a height of log n.  
 .[B] Red black trees always have the most recently inserted item at the root of the tree  
 .[C] 2-3-4 trees are trees that contain either 2 data items, 3 data items or 4 data items in each node.  
 .[D] All of the above  
 **.[E] None of the above**

**=== Question 5 ===**   
(2 marks)  
  
After running tests and determining that an unknown sorting algorithm has a time complexity of O(n) for descending ordered data, we would assume the algorithm is  
  
.[A] Quicksort with Median of Three Partitioning  
**.[B] A non-comparison based sorting method**  
.[C] Is unstable  
.[D] In-place

**Quicksort** median of 3 with ordered data should be super fast

Non-comparison based sorting methods **like radix and bucket sort we didn’t cover**, but I would bet it’s one of those

**we did cover in week 7, go through the slides and video if u missed~~!**

**Quicksort has best case nlog(n)**

**Should this question 5 be chosen B? Or A?**

**B. Imagine you have i = 0, j = N-1 and you’re just swapping a[i++] with a[j--] until i >= j**

**=== Question 6 ===**   
(2 marks)  
  
Consider the following Binary Search Tree:  
  
 15  
 / \  
 3 16  
 / \ \  
 1 8 17  
 \  
 9

A pre-order traversal would give the output in which order:  
  
 .[A] 1 3 8 9 15 16 17  
 .[B] 1 9 8 3 17 16 15  
 .[C] 15 3 16 1 8 17 9   
 **.[D] 15 3 1 8 9 16 17**  
 .[E] None of the above

Pre, in and post order are are recursive DFS algo’s.

Pre-order calls Display data, DFS left, DFS right

In-order calls DFS left, display data, DFS right

Post order calls DFS left, DFS right, display data

For the record, A is an in-order traversal, B is a post-order traversal and C is a level-order traversal

Wouldnt in-order h

**=== Question 7 ===**   
(2 marks)  
  
The following alternatives list expressions for the worst-case time complexity T(n) of various algorithms. Which alternative has an asymptotic worst-case complexity of O(n^2)?  
  
 .[A] n^2  
 .[B] 0.3n^2 + n + 1000  
 .[C] 1.3n(n+1) + 10logn  
 .[D] 2n^2 - 10  
 **.[E] All of the above**

All have big O notation of O(n^2)

Big O notation

**=== Question 8 ===**   
(2 marks)  
  
Consider the following tree:  
  
 10  
 / \  
 3 15  
 / / \  
 1 11 99  
  
Which of the following statements are true about this tree?  
  
 **.[A] The tree is height balanced** .[B] The tree is size balanced  
 .[C] The tree is not height balanced or size balanced  
 .[D] The tree is height balanced and size balanced

.[E] The tree is height balanced and size balanced and complete

**Idk what size balanced means**

size mean the number of nodes on both sides(left and right)

E.g here is a tree that is size-balanced but not height balanced

10

/ \

5 20  
 / / \  
 4 15 23  
 / / \  
 3 14 17

/

2

/

1

Isn’t height balanced depth(left) == depth(right)? So wouldn’t the above tree be height balanced? Can someone please confirm lol..

Ahh you’re right, I was getting height balance confused with minimum height. Fixed it up

Cant be E, bc otherwise we’d have node left of 3

http://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/

**=== Question 9 ===**   
  
(5 marks)  
   
Assume we have the following minHeap  
 5  
 / \  
 / \  
 10 29  
 / \ / \  
11 12 30 31

(a)  
Show the resulting heap after inserting the key 7. You should  
show your answer as both a tree structure and an array (please show array indexes as well as values for the array).

Answer:

5  
 / \  
 / \  
 7 29  
 / \ / \  
10 12 30 31

/

11

0 1 2 3 4 5 6

[ X 5 7 29 10 12 30 31 11]

the code implementation in the lecture example did not use 0 index in the array. However some implementation online did use 0 index.

(b)  
Draw the ORIGINAL heap shown above after removing the minimum.

31  
 / \  
 / \  
 10 29  
 / \ /

11 12 30

and you got to fix down from this state.

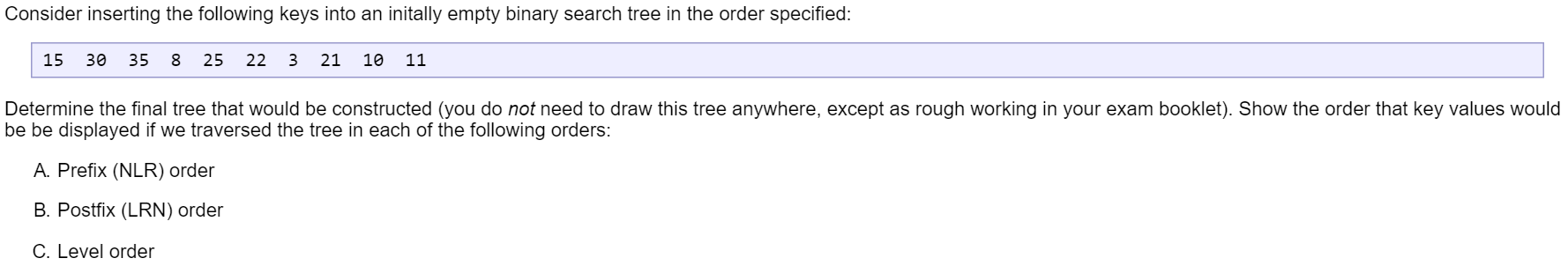
=== Question 10 ===   
(5 marks)  
Name and describe three ways of handling collisions in a hashtable.

Linear probing: When a collision has occurred, move through the array one by one until the first empty spot has been found.

Separate Chaining: In a collision, add a linked list structure to the current array index so that it is also found under the required index by traversing the list.

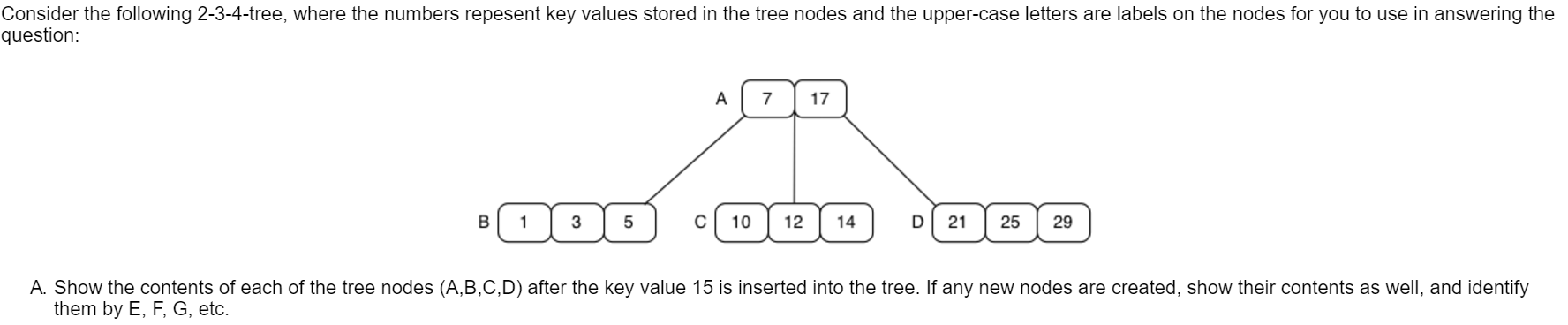
Double hashing: Create another hash function that returns a new number n and move to the next index n spaces away from the initial one. Keep moving n spaces until a free spot has been found, second hash function can't produce increments is 0.

=== Question 11 ===



1. 15,8,3,10,11,30,25,22,21,35
2. 3,11,10,8,21,22,25,35,30,15
3. 15,8,30,3,10,25,35,11,22,21

=== Question 12 ===

A. 

.

A(7, 12, 17)

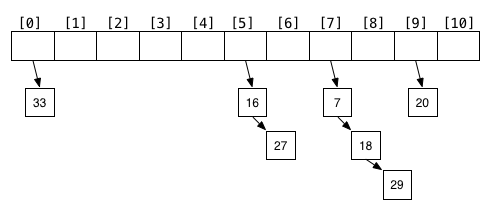
B(1, 3, 5) C(10) D(14,15) E(21, 25, 29)

### **Question 8** (8 marks)

Consider a hash table that uses *chaining* for collision resolution and stores items in the chains in ascending order on key value. The table has 11 slots and uses the following hash function:

int hash(int k) { return (k % 11); }

As an example, the following diagram shows the state of such a hash table after items containing the keys 7, 16, 18, 20, 27, 29 and 33 are inserted into an initially empty table:



**Now consider a scenario where we start with an initially empty table and insert items with key values 1 to 100 (inclusive) into the table.**

**(Therefore, there are 100 items in the tables. Most of slots have chain length 9 except bucket 1)**

1. What is the *average* chain length in this table? Show your working. (2 marks)

Average chain length is about 9, as there are 100 items added to the array of size 11, so if all arrays were evenly spread (as they are) then all indexes will have 9 items chained to it, while one of them (at index 1) will have 10 items. (**100/11** ≈ 9)

1. How many items are examined in searching for the key 42? (1 mark)

**4** items, starting at index 9 we look through **9, 20, 31 and 42**.

1. How many items are examined in searching for the key 999? (1 mark)

9 items, starting at index 9 we traverse through multiples of 11 upto 99.

*Edit: It’s in the same group as 9 to 97, which is still 9 items big*

Why ? can someone explain this ?

*Look at the remainder when 999 is divided by 11.* ***It’s 0****. The same goes for 99/11. So, 99 and 999 would have been in the same index. 90\*11+9=999*

*^^True. Should be in the same group as 9, 20, 31, 42, 53, 64, 75, 86, 97..*

1. If we continue inserting keys in sequence (101, 102, 103, ...), what is the worst-case search cost **after *N* items have been inserted**? (Measure search cost in terms of the number of items examined) (1 mark)

Average chain length is ceiling((100+N)/11). Worst- case: you need traversal the whole chain to search an item. worst case from before 100 is 10, plus whatever N insert...  
**10**+ceil(N/10)

Why not 10 + ceil(N/11)? *// it would be floor, not ceiling, as you only increment the max height by 1 after each number after a multiple of 11 is added, eg. 1, 12, 23, 34...*

underlying is longest chain of N Items inserted.( the worse case after N items inserted after 100 key inserted, so far there are 100+N items)

*^^ The case with 100 items is 10 in a single chain, and with 101 items it is also 10 in a single chain (being at indexes 1 and 2). By that formula, you’re saying that there’s 11 in a single chain as worst case.*

1. What is the best-case search cost after *N* items have been inserted? (Measure search cost in terms of the number of items examined) (1 mark)

1 item, either by finding it right after implementing the hash function, or seeing that the array index is empty.

Now consider starting with an empty hash table once again.

F. Write a C for-loop to generate a sequence of key values that would produce a single long chain containing 100 items**,** if items with these keys were inserted into an initially empty table. (2 marks)

// Hashtable is defined by h and a function to insert an item i into h is insertIntoTable(h,i)

int i;

for (i = 0; i < 100; i++) {

insertIntoTable(h,11\*i);

// or simply insertIntoTable(h, 1) anything % 1 == 0

// well thats if we want to add the same number into the chain, not sure if thats allowed, but we cant just do 1, has to be multiple of 11 + a set number for the same index to be chained up/

}

Depth of tree with n nodes : min = floor(logN) max = n-1(degenerate tree)

**Question: if you only have the root node (also a leaf node, left and right pointers are NULL)**

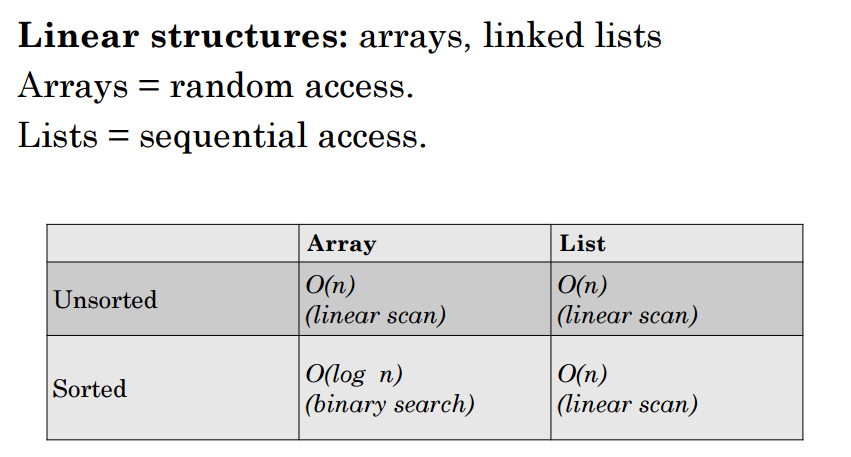
**Is the depth == 0? Or 1? thanks**

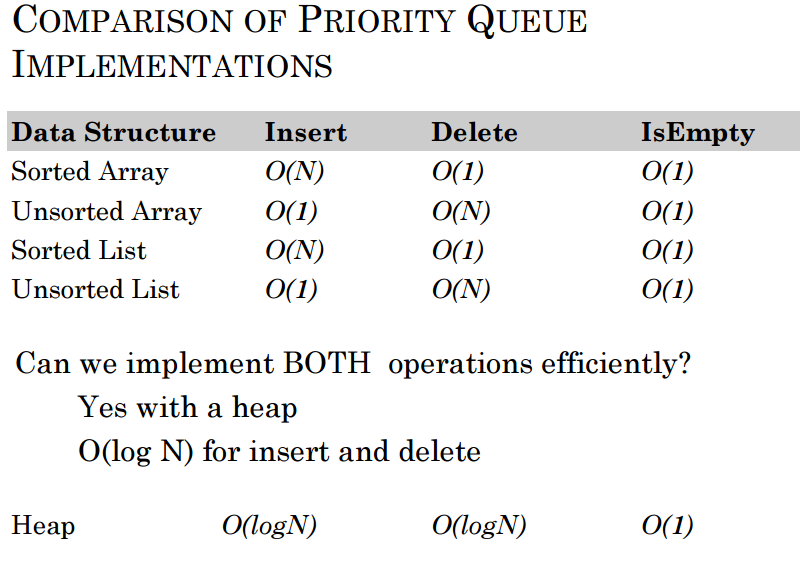
**0**

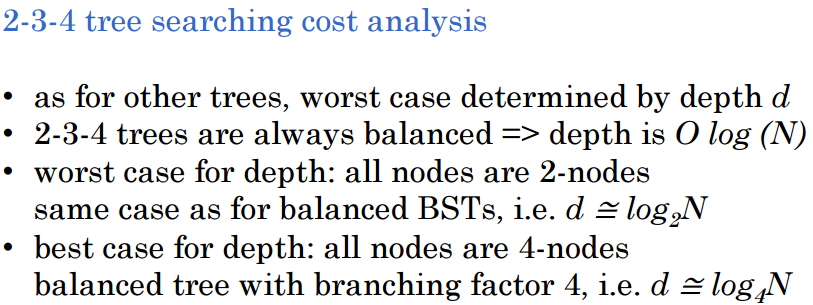
pre order NLR

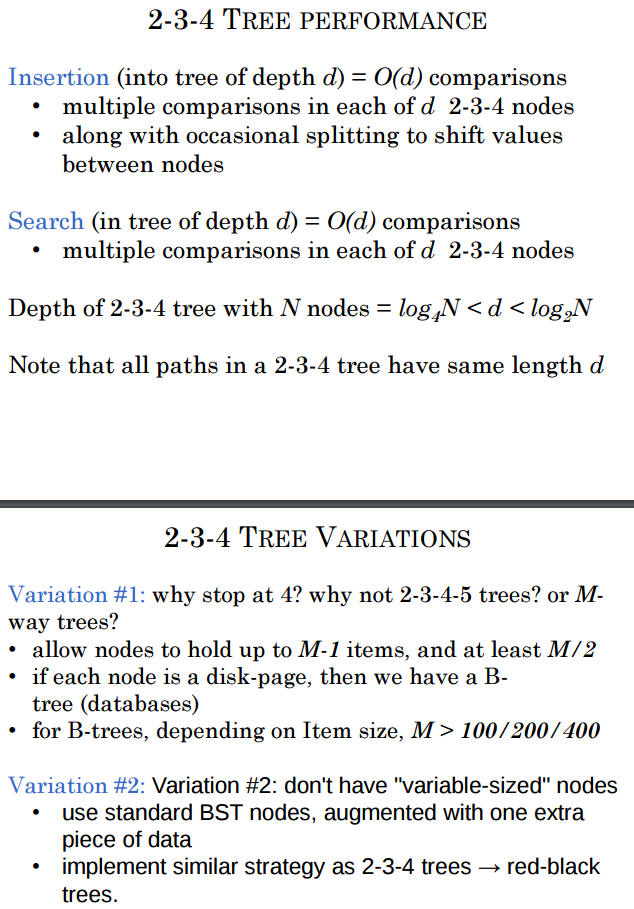
in order LNR

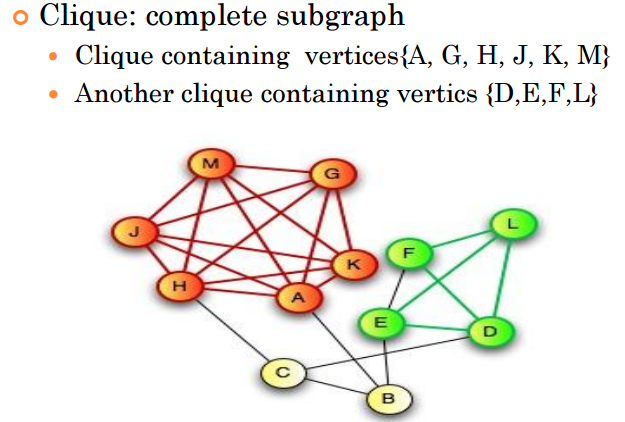
post order LRN



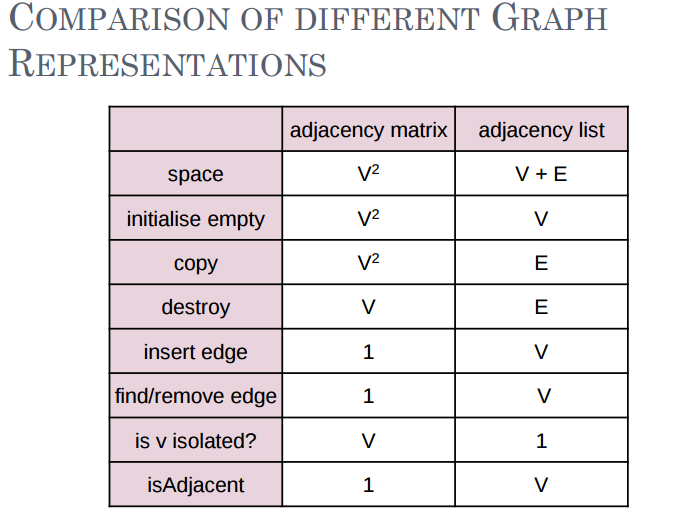


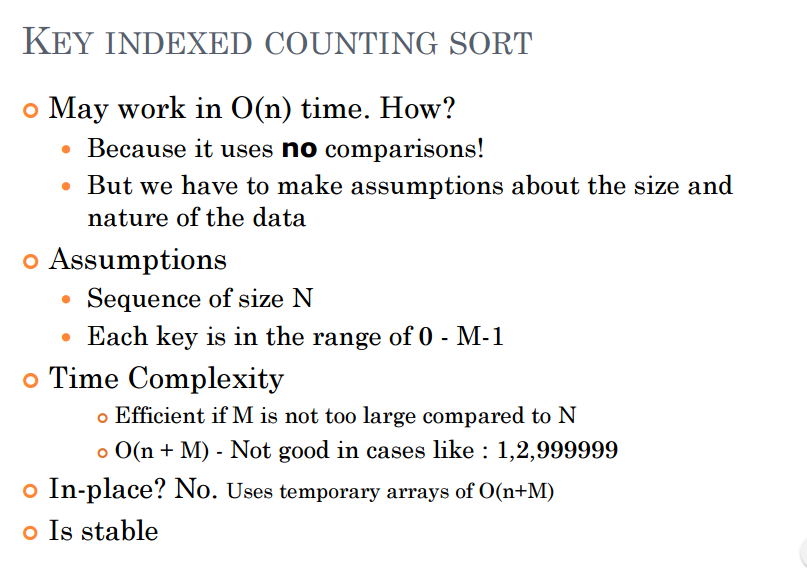




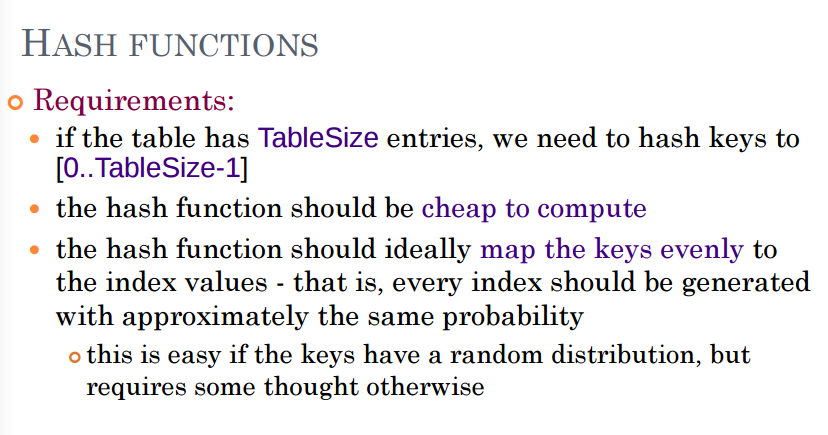


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sorting | Time Complexity  best avg worst | Space Complexity | Stable | In-place | Adaptive |
| Insertion | O(N) O(N^2) O(N^2) | O(1) | Y | Y | Y |
| Shell | O(N^3/2) | O(1) | N | Y | Y |
| Selection | O(N^2) O(N^2) O(N^2) | O(1) | Depends on the implementation | Y | N |
| Bubble | O(N) O(N^2) O(N^2) | O(1) | Y | Y | Y |
| Quick | O(NlogN) O(NlogN) O(N^2) | O(logN) Stack call | Y-Linked list implementation  N-Array | Y | N |
| Merge | O(NlogN) O(NlogN) O(NlogN) | O(N)-Array  O(logN)-Linked list | Y | N | N |
| Heap | O(NlogN) O(NlogN) O(NlogN) | O(1) | N | Depends on the implementation | N |
| Bucket | O(N+K) O(N+K) O(N^2) | O(n) | N | Y | N |
| Radix | O(NK) O(NK) O(NK) | O(N+K) | Y | Y | N |

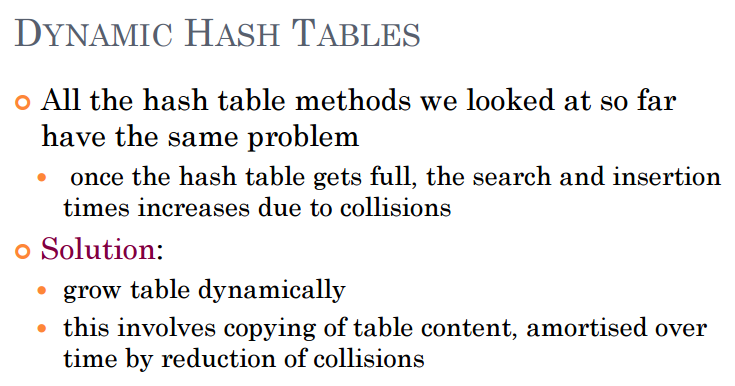




for more non-comparison sort, watch last week lecture video. Lecturer did more examples in classes than in the slides

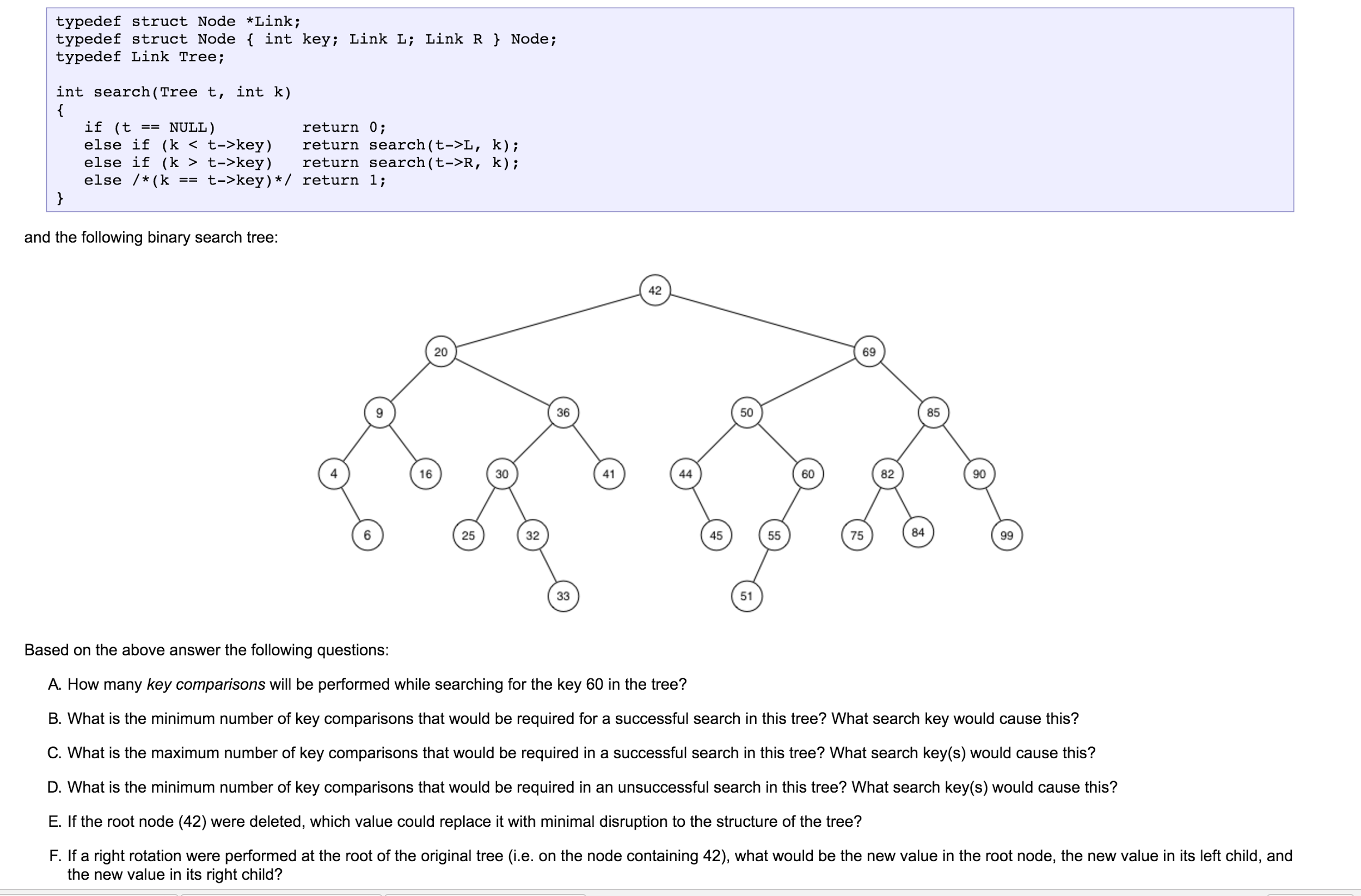


*this is useful for when you answer what is wrong with the hash function. you can pick some of these requirements, to tell that problem hash functions did not matched these requirements.*



What about splay Tree ?

Time Complexity of tree algorithms is typically O(logN) logN is the depth of the tree.



This question has answered in the comment of the past exam by John.

Thanks

*Q5: Does "Key comparisons" want us to consider every if >, <, == statement per node or just count every time it compares to a node?*

*Every comparison. You don't have to count the NULL check, because it asked for key comparisons. There are also no explicit == on keys. So, there are two key comparisons on the root node, one key comparison on the (69) node, two key comparisons on the (50) node, and two key comparisons on the (60) node to work out it's equal.*

*Answers for 15s2 Final (Q5 above is from 15s2 Final)*

*https://gist.github.com/ekohilas/e9d8f757b4173b3b9a7d94681898efe1#file-q4-txt*

[*https://github.com/andrewjbennett/cs1927-prac-solutions/blob/master/q3-Graph.c*](https://github.com/andrewjbennett/cs1927-prac-solutions/blob/master/q3-Graph.c)

*Go*

1. [↑](#footnote-ref-0)