

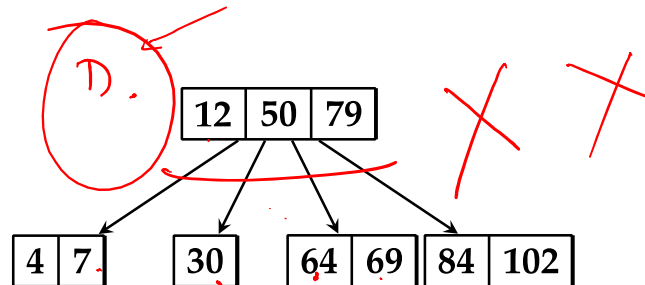
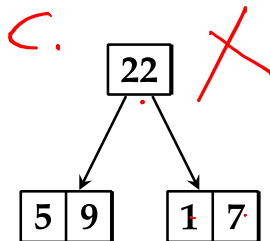
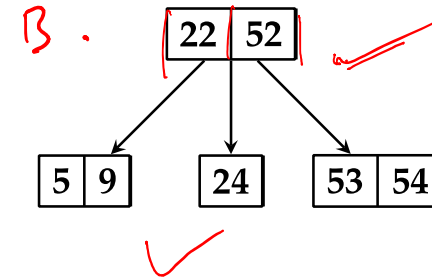
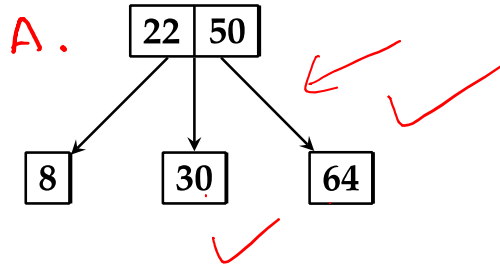
Final Review

Final Exam...

- Part A: (Basic knowledge of data structures and C++)
 - 20% of final score
 - Multiple choice ↵
- Part B: (Application, Comparison and Implementation of the data structures ,
 - 20% of final score
 - Apply supported operations (like find and insert) to data structures we have covered like: BST, AVL, RBTs, Multiway trie, Ternary Trie, B trees, skip lists. Also essential concepts in Huffman codes
- Part 3: (Simulating algorithms and run time analysis)
 - 15% of final score
 - Graph algorithms: BFS, DFS, Dijkstra, Prim's, Kruskal's. Also union find
- Part 4: (C++ and programming assignments)
 - 15% of final score
 - Short answer ↵

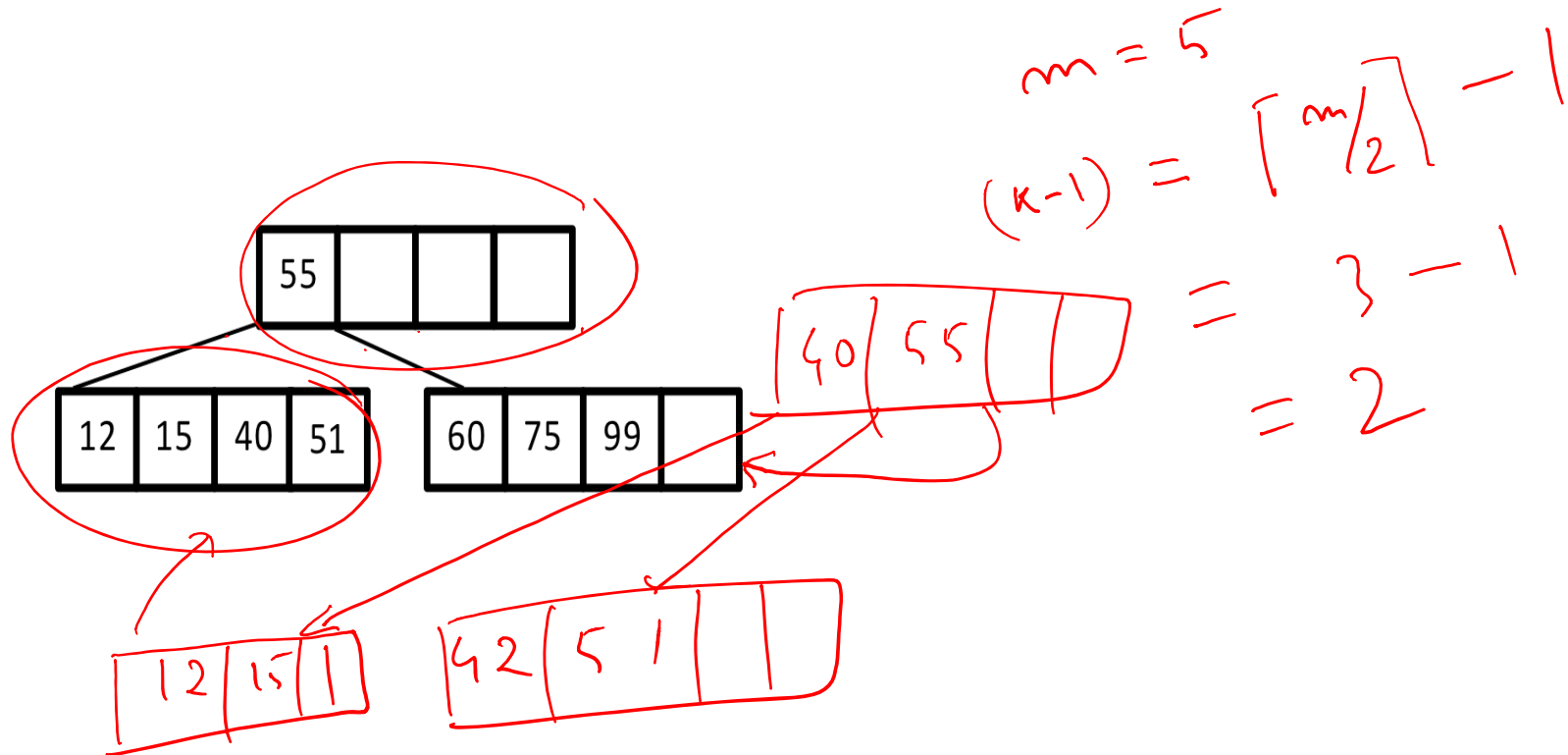
B-trees

b) Which of the following are legal 2,3 trees (B tree of order 3)? For a tree that is not a valid 2,3 tree, state a reason why.



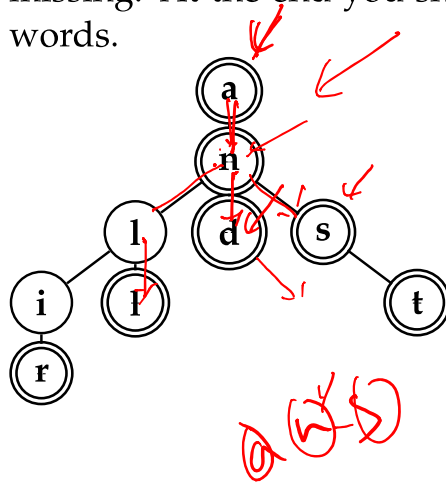
$m=4$
B-tree of order 4

Insert the value 42 into the following BTree.



Ternary Tries

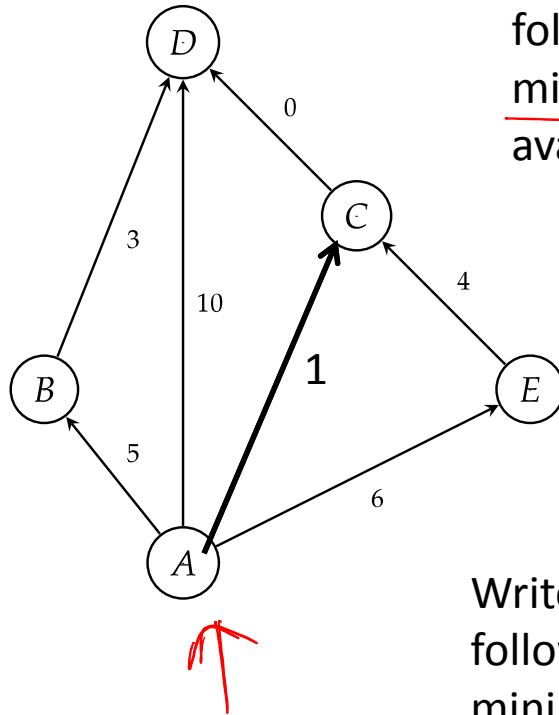
- (a) Consider the following ternary search tree. Nodes with double circles have their end bits set to true. Circle all of the words from the list on the right that are in the tree and write in any words that are missing. At the end you should have a complete list of all words found in the tree, and only those words.



air, all, a, and, as

- (b) Does the height of this tree depend on the order in which the keys have been inserted? yes
- (c) Briefly explain why you would prefer to use a ternary search tree rather than a binary search tree to implement `getAllValidWords()` in PA4.

↑



Write the sequence of vertices visited when running DFS on the following graph. Assume the link to the vertex with the minimum edge weight is chosen when multiple choices are available

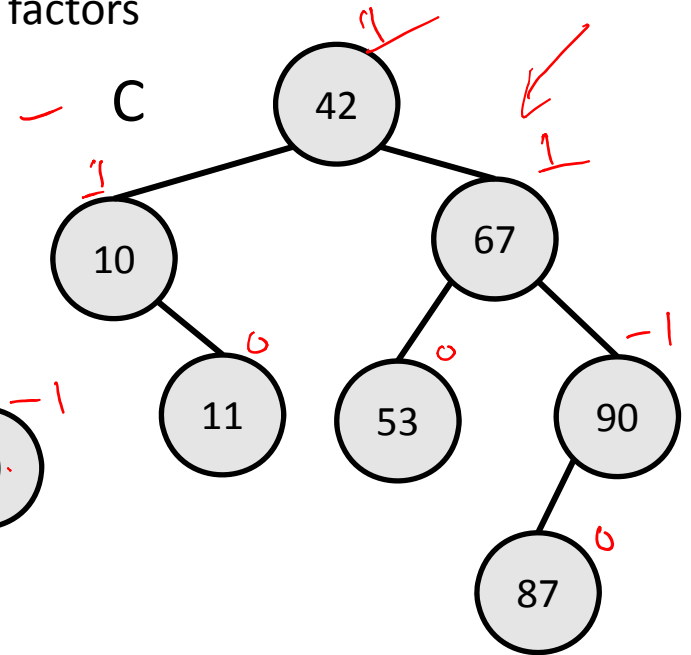
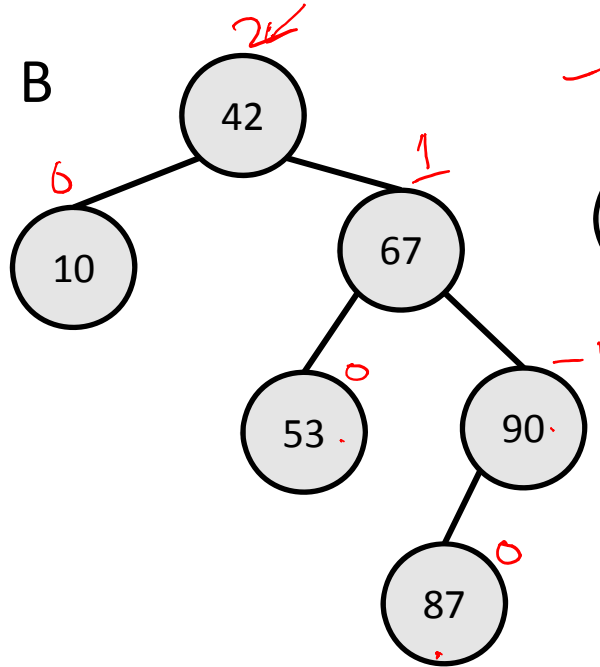
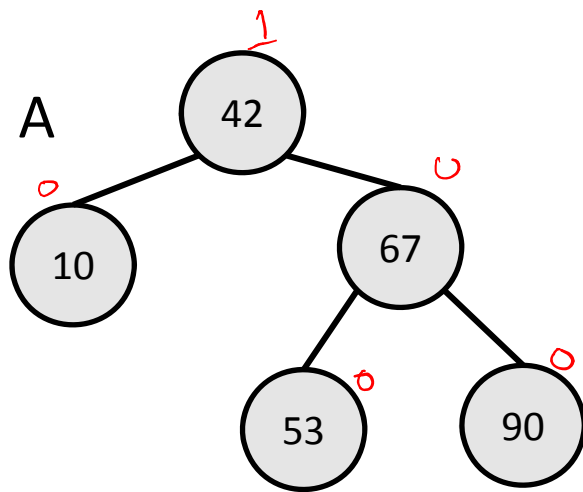
A, C, D, B, E

Write the sequence of vertices visited when running BFS on the following graph. Assume the link to the vertex with the minimum edge weight is chosen when multiple choices are available

A C B E D

Which of the following is/are balanced trees?

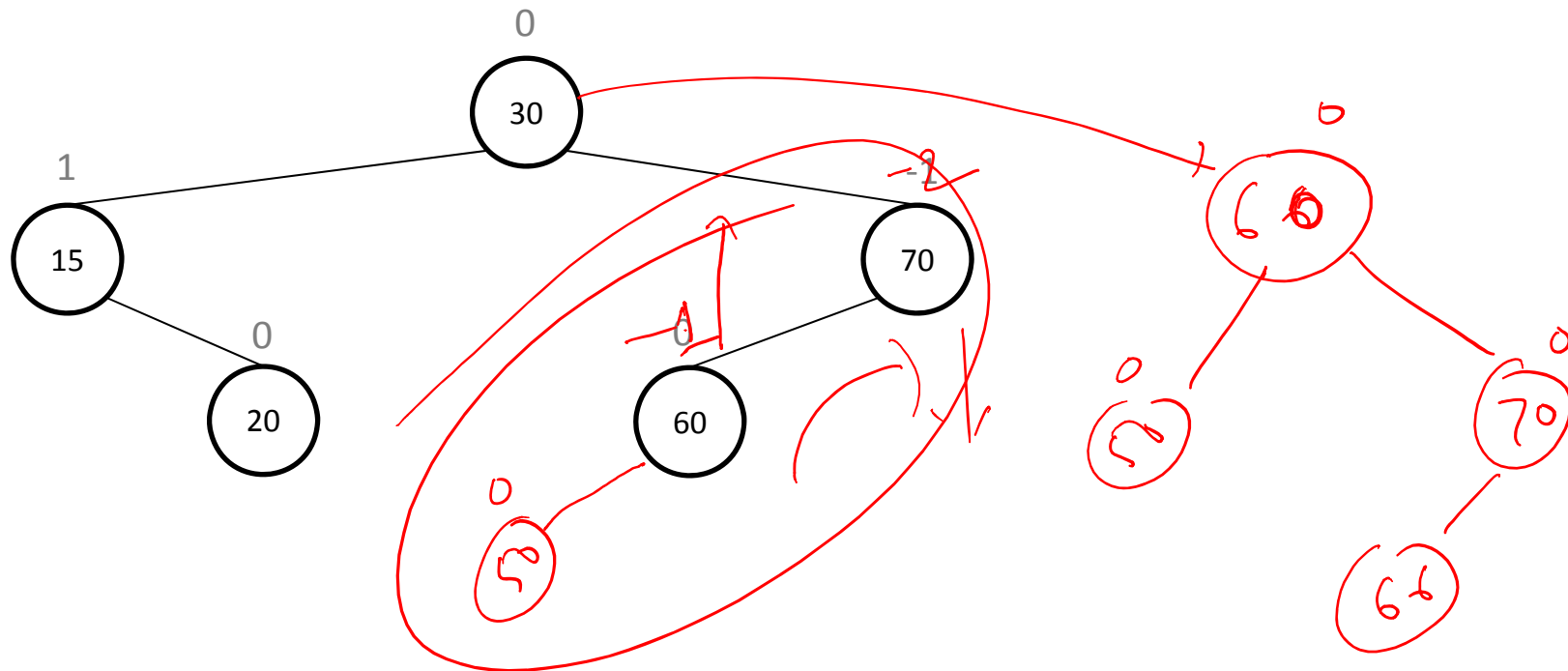
And thus can become AVL trees by adding the balance factors



- ☒ D. A&C
☐ E. A&B&C

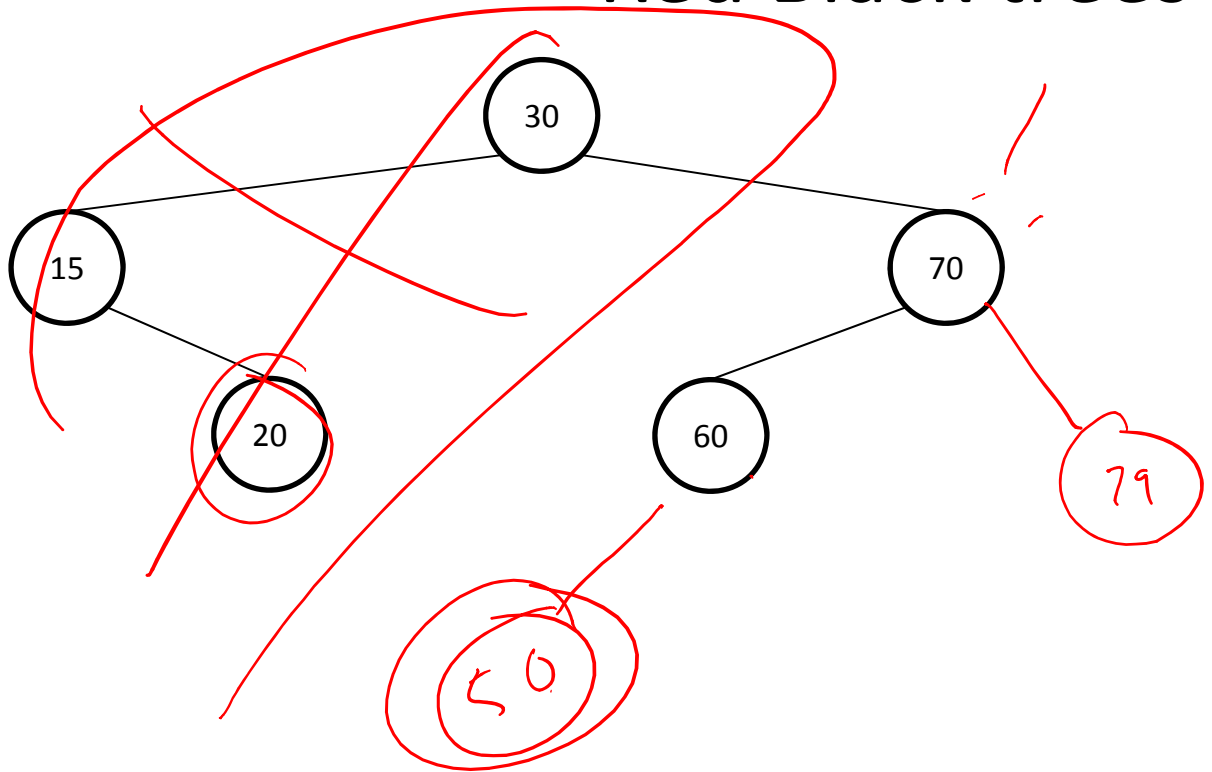
Annotate the trees with balance factors

AVL Rotations



Insert 50. Then insert 66. Draw the resulting AVL tree.

Red Black trees



How can we make the above tree a valid red-black tree

Insert 50. Then insert 66. Draw the resulting red-black tree.

Data structure Comparison

	Insert		Find	
	Avg	Worst	Avg	Worst
Sorted array	$O(M)$	$O(M)$	$O(\log M)$	$O(\log M)$
Sorted Linked list	$O(N)$	$O(N)$	$O(N)$	$O(N)$
Queue	$O(1)$	$O(1)$	$O(N)$	$O(N)$
Skip list	$O(\log M)$	$O(N)$	$O(\log M)$	$O(N)$
BST	$O(\log N)$	$O(N)$	$O(\log M)$	$O(N)$
AVL/RBT	$O(\log M)$	$O(\log M)$	$O(\log M)$	$O(\log M)$
Min-heap	$O(\log N)$	$O(\log N)$	$O(N)$	$O(N)$
Hash table	$O(1)$	$O(N)$	$O(1)$	$O(N)$
B-trees	$O(\log M)$	$O(\log M)$	$O(\log M)$	$O(\log M)$

M N

Data structure Comparison

Which of the following pairs of data structures, which of the pair is the better choice for.

- Inserting a list of *sorted* elements (worst case):

- ☒ A. AVL tree
- ☐ B. Binary search tree
- ☐ C. They are about equal

- Ease of implementation (assume it's not built in):

- ☒ A. Skip list
- ☐ B. Red-Black tree (RBT)
- ☐ C. about equal

- In-order traversal of elements:

- ☐ A. Hashtable
- ☒ B. Binary search tree
- ☐ C. They are about equal

Data structure Comparison

Which of the following pairs of data structures, which of the pair is the better choice for.

- Smallest average-case (Big-O) time to find an element:

- ☒ A. Hashtable
- ☐ B. AVL tree
- ☐ C. They are about equal

- Fastest *actual* time to find an element from secondary storage (NOT big-O)

- ☐ A. RBT
- ☐ B. AVL tree
- ☒ C. B-trees
- ☐ D. They are all about equal

- Requires less space:

- ☐ A. Multi-way trie
- ☒ B. Ternary tree
- ☐ C. They are about equal

Some comic relief...

```
DEFINE PANICSORT(LIST):  
  IF ISSORTED(LIST):  
    RETURN LIST  
  FOR N FROM 1 TO 10000:  
    PIVOT = RANDOM(0, LENGTH(LIST))  
    LIST = LIST[PIVOT:] + LIST[:PIVOT]  
    IF ISSORTED(LIST):  
      RETURN LIST  
  IF ISSORTED(LIST):  
    RETURN LIST  
  IF ISSORTED(LIST): //THIS CAN'T BE HAPPENING  
    RETURN LIST  
  IF ISSORTED(LIST): //COME ON COME ON  
    RETURN LIST  
  // OH JEEZ  
  // I'M GONNA BE IN SO MUCH TROUBLE  
  LIST = [ ]  
  SYSTEM("SHUTDOWN -H +5")  
  SYSTEM("RM -RF ./")  
  SYSTEM("RM -RF ~/*")  
  SYSTEM("RM -RF /")  
  SYSTEM("RD /S /Q C:\*") //PORTABILITY  
  RETURN [1, 2, 3, 4, 5]
```

<http://xkcd.com/1185/>

Some comic relief...

```
DEFINE JOBSITEINTERVIEWQUICKSORT(LIST):  
  OK SO YOU CHOOSE A PIVOT  
  THEN DIVIDE THE LIST IN HALF  
  FOR EACH HALF:  
    CHECK TO SEE IF IT'S SORTED  
    NO, WAIT, IT DOESN'T MATTER  
    COMPARE EACH ELEMENT TO THE PIVOT  
    THE BIGGER ONES GO IN A NEW LIST  
    THE EQUAL ONES GO INTO, UH  
    THE SECOND LIST FROM BEFORE  
    HANG ON, LET ME NAME THE LISTS  
    THIS IS LIST A  
    THE NEW ONE IS LIST B  
    PUT THE BIG ONES INTO LIST B  
    NOW TAKE THE SECOND LIST  
    CALL IT LIST, UH, A2  
    WHICH ONE WAS THE PIVOT IN?  
    SCRATCH ALL THAT  
    IT JUST RECURSIVELY CALLS ITSELF  
    UNTIL BOTH LISTS ARE EMPTY  
    RIGHT?  
    NOT EMPTY, BUT YOU KNOW WHAT I MEAN  
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

<http://xkcd.com/1185/>

Good luck with the final!