COMPSCI 2C03 – Week 6 Exercises

Sample solutions and notes on sample solutions for some of this week's exercises. Corrected Nov 8.

Lecture 1: Symbol Tables and Binary Search

1. Suppose you have the following sorted list [3, 5, 6, 8, 11, 12, 14, 15, 17, 18] and are using the binary search algorithm. Give the sequences of elements examined to search for the following keys: 8, 12, 3, 10, -1.

   8: 11, 5, 6, 8
   12: 11, 15, 12
   3: 11, 5, 3
   10: 11, 5, 6, 8
   -1: 11, 5, 3

2. Exercise 3.1.13: What would be the best symbol-table implementation (Unordered Linked List, Unordered Array, Ordered Array) for an application that does 10^3 **put** operations and 10^6 **get** operations, randomly intermixed. Justify your answer.

   Unordered Linked List and Array are very similar: insert is constant, get is linear. A million linear, a thousand constant.
   Ordered Array: insert is linear, get is logarithmic. A thousand linear, a million logarithmic.
   Ordered Array is a better choice.

3. Exercise 3.1.17: The textbook defines a **floor** operation for the **SymbolTable** ADT. A call to floor(key) returns the largest key less than or equal to **key**. Implement the **floor** function for an ordered array implementation.

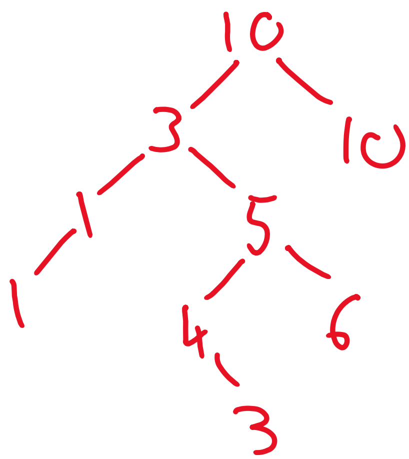
```
floor(st, key):
  if st.n == 0:
    return null

  int lo = 0, hi = st.n-1
  while lo <= hi:
    mid = lo + (hi – lo) // 2
    if st.items[mid] == key: return key
    else if st.items[mid] < key: lo = mid + 1
    else: hi = mid – 1

  if hi < 0: return null
  else: return st.items[hi]
```

Lecture 2: Binary Search Trees (BSTs)

4. Insert the keys 10, 3, 1, 1, 5, 4, 6, 10, 3 into a BST in the order given and draw the resulting tree.



(assuming < goes left, ≥ goes right.)

5. Give five orderings of the keys 65, 88, 67, 83, 69, 72 that, when inserted into an initially empty BST, produce the best-case tree. (Question modified from original version to remove key 82.)

Sort the keys first: 65, 67, 69, 72, 83, 88

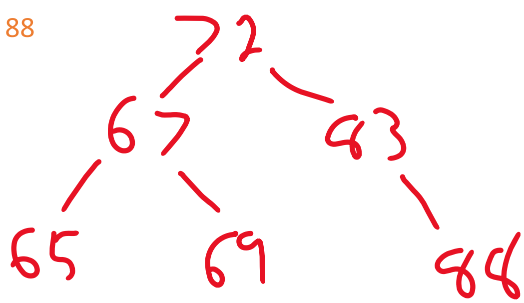
72, 67, 83, 65, 69, 88

72, 83, 88, 67, 65, 69

72, 67, 65, 69, 83, 88

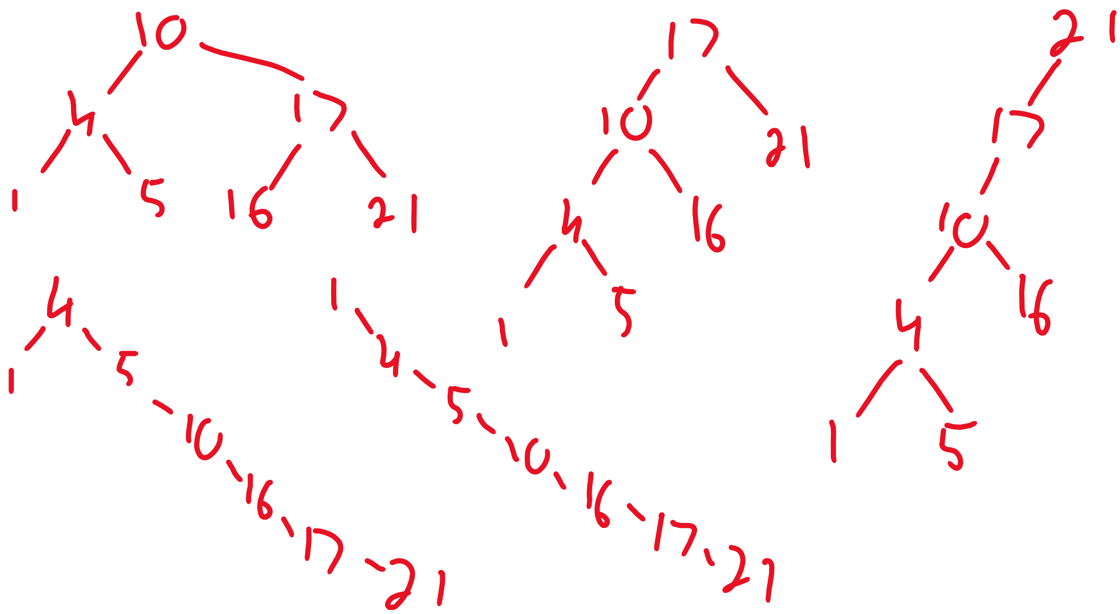
72, 67, 69, 65, 83, 88

72, 83, 67, 65, 69, 88

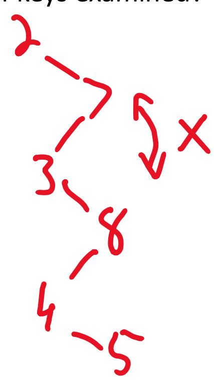


Can you find orderings that start with key 69?

6. For the set of keys {1, 4, 5, 10, 16, 17, 21}, draw BSTs of heights 2, 3, 4, 5, and 6.



7. Suppose that a certain BST has keys that are integers between 1 and 10, and we search for 5. Which sequence below cannot be the sequence of keys examined?
    a. 10, 9, 8, 7, 6, 5
    b. 4, 10, 8, 7, 5
    c. 1, 10, 2, 9, 3, 8, 4, 7, 6, 5
    d. 2, 7, 3, 8, 4, 5 x
    e. 1, 2, 10, 4, 8, 5



8. Binary Tree Properties
    a. Prove by induction that a tree of height h has no more than 2^h leaves.

    We want to prove that all binary trees of height h have at most 2^h leaves. We prove this by induction on the height h.

    Base Case: h = 0. A binary tree of height h = 0 is just a single node with no children. The number of leaves in it = 1 = 2^0. Hence, the base case satisfied.

    Induction Hypothesis: Suppose T(k) is any binary tree of height h = k, k > 0, with no more than 2^k leaves.

    Then, we need to show that by increasing the height of T by one; that is, for T(k + 1), the number of leaves in it is no more than 2^{k+1}.

Since T(k + 1) is a binary tree and obtained by increasing the height of T(k) by one, the maximum number of leaves in T(k + 1) = 2 ∗ max. no of leaves in T(k) - that is each leaf in T(k) has two leaf nodes attached. By induction hypothesis, the maximum number of leaves in T(k) is 2^k. Therefore, the maximum number of leaves in T(k + 1) is = $2 * 2^k = 2^{k+1}$. Hence the hypothesis holds for k + 1, and so the theorem is proved.

b. A **full binary tree** is a tree in which every node is a **leaf** (no children) or an **internal node** with exactly 2 children. How many internal nodes are there in a full binary tree with **n** leaves?

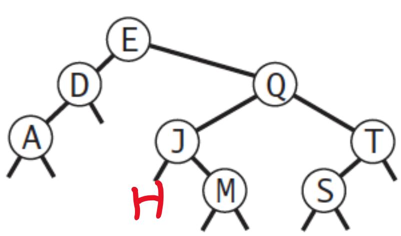
Suppose there are i internal nodes, each having 2 children, therefore total children in the tree is 2 × i.

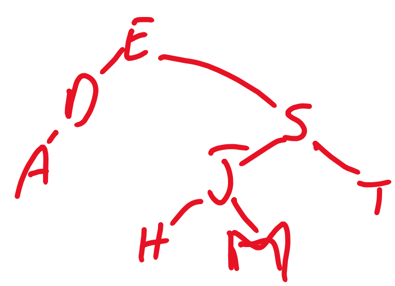
There are i − 1 internal nodes which are children of some other node (root has been excluded hence one less than the total number of internal nodes)

That is, out of these 2×i children, i−1 are internal nodes and therefore the rest are leaves. There are n = ((2 × i) – (i − 1)) of them. Hence n = i + 1 and if we have n leaves, then i = n − 1.

Lecture 3: More BST Algorithms

9. Consider the BST given below.
   a. Draw the tree that results from inserting the key H.
   b. Draw the tree that results from deleting the key Q.
   c. Show the sequence of nodes visited in a pre-, post-, and in-order traversal for the tree obtained from (a) and (b).
   d. Give the sequence of nodes visited to compute the **min** and **max** operations, on the tree obtained from (a) and (b).
   e. What is the rank of the node T in the tree obtained from (a) and (b).





Pre: E D A S J H M T

Post: A D H M J T S E

In: A D E H J M S T

Min: E D A   Max: E S T   Rank(T)=7

10. Give a pseudocode implementation the **keys** operation from the ordered **SymbolTable** ADT using a BST implementation.

```
keys(st):
  if st == null: return []
  return keys(st.left)+[st.key]+keys[st.right]
```

11. Exercise 3.2.14: Give non-recursive pseudocode implementations of **min** and **get**.

```
min(st):
  if st == null: return null
  c = st
  while c.left != null:
    c = c.left
  return c.left.key
```

```
get(st, key):
  c = st
  while c != null and c.key != key:
    if c.key > key: c = c.left
    else: c = c.right
  if c == null: return null
  return c.value
```

12. Exercise 3.2.23: Is BST deletion commutative? That is, does **delete(x)** followed by **delete(y)** yield the same tree as **delete(y)** followed by **delete(x)**? Justify your answer.

No. It's the empty subtree steps that make it not commutative.

Delete A, then B

```
 A
/ \
B  D
  /
 C

 C
/ \
B  D

C
 \
  D
```

Delete B, then A

```
 A
/ \
B  D
  /
 C

A
 \
  D
 /
C

 D
/
C
```