CSE 250 Spring 2010
Homework 4 Solutions
Due Date: April 6, Wed, by 2:05pm
Total Points: 34

1. (3 pts) Consider the array:

   1, -9, 89, 25, 0, 5, 99, -4

   Show the progress and the array after each pass, when `quick_sort` is performed on this array. (Assume the first element of the array is used as the pivot element).

   Solution. :

`Input`	1	-9	89	25	0	5	99	-4
`Pivot` $= 1$								
`After Partition`	0	-9	-4	1	25	5	99	89
`Input`	0	-9	-4	1				
`Pivot` $= 0$								
`After Partition`	-4	-9	0	1				
`Input`	-4	-9						
`Pivot` $= -4$								
`After Partition`	-9	-4						
`Input`	-4							
`Input`		-9						
`Input`				1				
`Input`					25	5	99	89
`Pivot` $= 25$								
`After Partition`					5	25	99	89
`Input`					5			
`Input`							99	89
`Pivot` $= 99$								
`After Partition`							89	99
`Input`							89	
`Final`	-9	-4	0	1	5	25	89	99

2. ($3 + 3 + 1 = 7$ points) As discussed in class, a `dictionary` is an abstract data type that supports the following operations:

   - `insert(key)`;
   - `find(key)`;
   - `delete(key)`;

   Here we assume that the `insert(key)` function simply inserts a new entry into the dictionary, without checking if there is already an entry with the `key` value in the `dictionary`.

   `Dictionary` can be implemented by using a sorted array, or an un-sorted array. The runtime of functions for the two implementations are summarized in the following table:

	insert(key)	find(key)	delete(key)
sorted array	$O(n)$	$O(\log n)$	$O(n)$
un-sorted array	$O(1)$	$O(n)$	$O(n)$

(a) We can also use an *un-sorted linked list* to implement a `dictionary`. Briefly describe how to implement the three functions for this implementation. (Just description, not code). State the runtime of each function. (Your implementation should be as efficient as possible).

(b) Consider the following program segment:

```
Dictionary<int> D;

for (int i = 0; i < n; i++) {
     D.insert(i);
     for (int j = 0; j < i; j++)
           D.find(j)
    }
D.delete(2);
D.delete(n-2);
```

Determine the numbers of times each of the three functions of dictionary are called in the segment. (Either exact number, or in O notation).

(c) For this program segment, which implementation (sorted array, or un-sorted array) should be used for the dictionary `D`? Explain why.

Solution. :

(a) `insert(key)` can be implemented with an unsorted linked list by inserting a new node at the head of the list ($O(1)$). `find(key)` can be implemented in $O(n)$ time by stepping through the linked list to find the desired key. `delete(key)` can be implemented in $O(n)$ time by using `find` to locate the desired key, and then adjusting the links in the list for the entry on either side.

(b) `insert` is called n times ($O(n)$), and for each time `insert` is called, `find` is called i times ($O(n)$), so `find` is called a total of $\frac{n^2}{2} = O(n^2)$ times. `delete` is called 2 times ($O(1)$).

(c) Our overall runtime for the different implementations are:
`Sorted Array`: $O(n)O(n)+O(n^2)O(\log n)+O(1)O(n) = O(n^2)+O(n^2 \log n)+O(n) = O(n^2 \log n)$
`Unsorted Array`: $O(n)O(1) + O(n^2)O(n) + O(1)O(n) = O(n) + O(n^3) + O(n) = O(n^3)$
`Unsorted Linked List`: $O(n)O(1) + O(n^2)O(n) + O(1)O(n) = O(n) + O(n^3) + O(n) = O(n^3)$
So the sorted array implementation is most efficient given the operation usage pattern of this program segment.

3. (3 points) Consider the tree T shown in Figure 1. Show the listing of nodes in T by using:

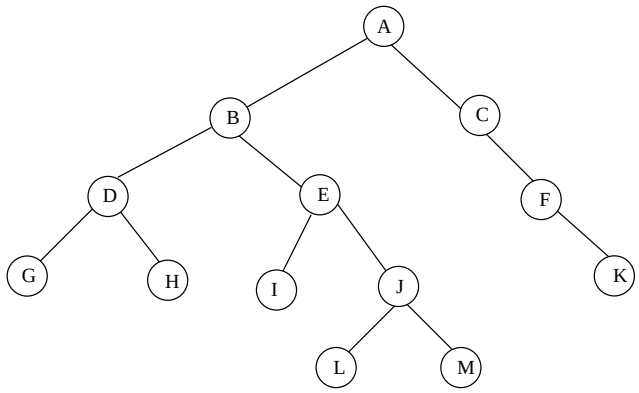


Figure 1: A binary tree.

(a) In-order traversal.
(b) Pre-order traversal.
(c) Post-order traversal.

Solution. :

(a) `G D H B I E L J M A C F K`

(b) `A B D G H E I J L M C F K`

(c) `G H D I L M J E B K F C A`

4. (1+2+2=5 points). Draw an expression tree corresponding to each of the following.

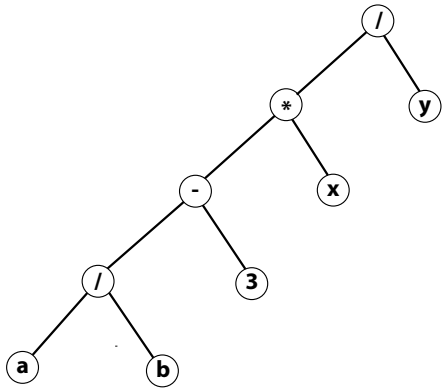
   (a) Inorder Traversal is `a / b - 3 * x / y`

   (b) Preorder Traversal is `* + a - x y / c d`

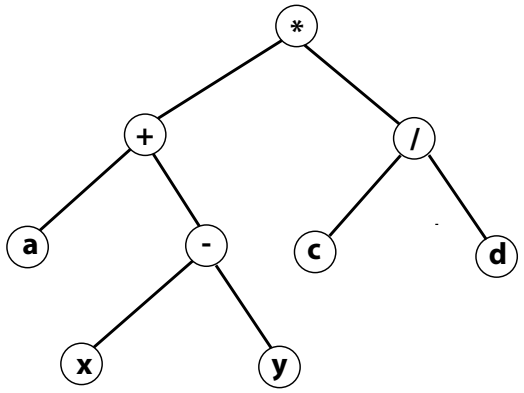
   (c) Postorder Traversal is `a b c + x y - z * / -`

Solution. :

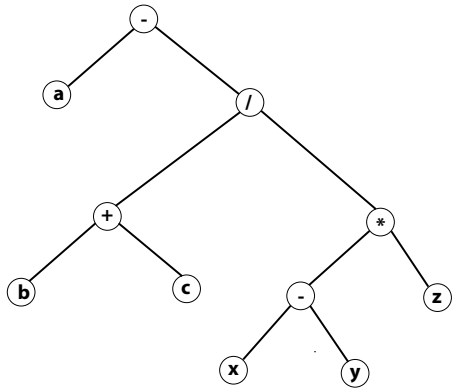
(a)



(b)



(c)

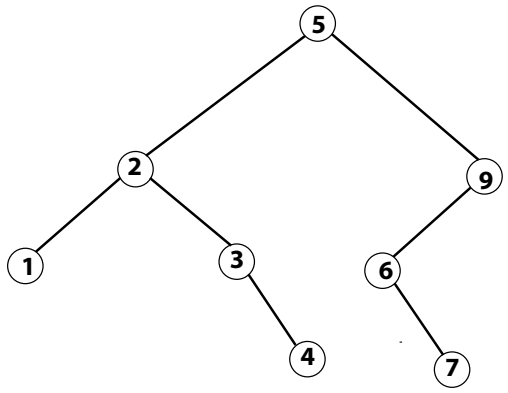


5. (4+2 = 6 points) (a) Items with the following key values 5, 2, 9, 6, 3, 1, 4, 7 are inserted into an initially empty BST (binary search tree). Show the resulting tree after each insert operation.

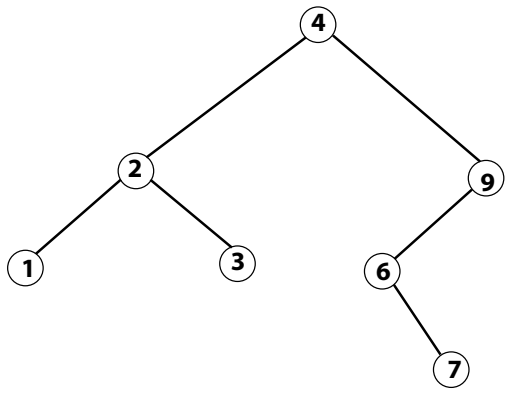
   (b) Show the result of deleting the root.

   Solution. :

(a)



(b)



6. (2+2+2= 6 points) Write efficient functions that take a pointer to the root of a binary tree T, and computes:

   (a) The number of nodes in T.

   (b) The number of leaves in T.

   (c) The height of T. (The height of T is the length of the longest path from the root of T to any leaf of T.)

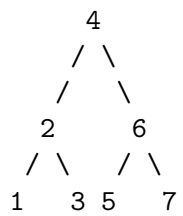
   Solution. :

   (a)
```
int TreeSize(BTNode* root) {
   if(root == NULL)
      return 0;
   else
      return 1 + TreeSize(root->left) + TreeSize(root->right);
   }
```

   (b)
```
int NumLeaves(BTNode* root) {
   if(root == NULL)
      return 0;
   else if(root->left == NULL && root->right == NULL)
      return 1;
   else
      return NumLeaves(root->left) + NumLeaves(root->right);
   }
```

   (c)
```
int TreeDepth(BTNode* root) {
   if(root == NULL)
      return 0;
   else
      return 1 + max(TreeDepth(root->left),TreeDepth(root->right));
   }
```

7. (4 points) Write a function that generate a perfectly balanced binary search tree of height h with keys 1, 2, ... $2^{h+1} - 1$. For example, when $h = 2$, the balanced binary search tree looks like:



   Solution. : Algorithm Idea:

   - Insert 2^h;
   - Recursively insert the set of the keys $\{1 \ldots (2^h - 1)\}$
   - Recursively insert the set of the keys $\{(2^h + 1) \ldots 2^{h+1} - 1\}$.

   The details of the algorithm are as follows:

```
BST T;  // T is a global BST, initially empty
```

   Let function pbBST(h, S) be the function that inserts a sequence of keys into T, to form a perfectly balanced binary search tree of hight h (so contains $2^{h+1} - 1$ keys) with keys beginning at S.

```
void pbBST(int h, int S){
    if (h>=0) {T.insert(2^h+S-1);
           pbBST(h-1,S)
           pbBST(h-1,2^h+S)
          }
    return;
}
```

We obtain a perfectly balanced binary search tree T by calling pbBST(h, 1);