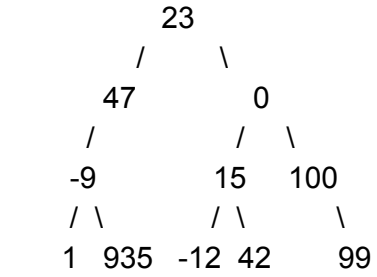
CS61CS61B Fall 2014 - Guerrilla Section 2 - Solutions

1. Trees

a) For the binary tree shown below, what are the preorder, inorder and post order traversals.




```
Preorder  : 23 47 -9 1 935 0 15 -12 42 100 99
Inorder   : 1 -9 935 47 23 -12 15 42 0 100 99
Postorder : 1 935 -9 47 -12 42 15 99 100 0 23
```

b) Given the following definition of a Node, fill in these methods so that they obey their corresponding javadoc comments:

```
class Node {
          int value;
          Node left;
          Node right;
}

/** Performs a preorder traversal of a
 *  binary tree rooted at ROOT. A node
 *  is 'visited' when its value is printed
 *  to stdout.
 */
static void preorderTraversal(Node root) {
     if (root == null)
         return;
      System.out.print(root.value + " ");
      preorderTraversal(root.left);
      preorderTraversal(root.right);
}
```

```
/** Performs an inorder traversal of a
   *  binary tree rooted at ROOT. A node
   *  is 'visited' when its value is printed
   *  to stdout.
   */
  static void inorderTraversal(Node root) {
          if (root == null)
                return;
          inorderTraversal(root.left);
          System.out.print(root.value + " ");
          inorderTraversal(root.right);
  }

  /** Performs a postorder traversal of a
   *  binary tree rooted at ROOT. A node
   *  is 'visited' when its value is printed
   *  to stdout.
   */
  static void postTraversal(Node root) {
          if (root == null)
                 return;
          postorderTraversal(root.left);
          postorderTraversal(root.right);
          System.out.print(root.value + " ");
  }
```

2. Sorting

For each of the sequences below identify the sorting algorithm being used:

1) 5103 9914 0608 3715 6035 2261 9797 7188 1163 4411

2261 4411 5103 1163 9914 3715 6035 9797 0608 7188

5103 0608 4411 9914 3715 6035 2261 1163 7188 9797

6035 5103 1163 7188 2261 4411 0608 3715 9797 9914

0608 1163 2261 3715 4411 5103 6035 7188 9797 9914
LSD radix sort.

2) 5103 9914 0608 3715 6035 2261 9797 7188 1163 4411

5103 9914 0608 3715 2261 6035 7188 9797 1163 4411

0608 3715 5103 9914 2261 6035 7188 9797 1163 4411

0608 2261 3715 5103 6035 7188 9797 9914 1163 4411

0608 1163 2261 3715 4411 5103 6035 7188 9797 9914
Merge sort.

3) 5103 9914 0608 3715 6035 2261 9797 7188 1163 4411

0608 1163 2261 3715 4411 5103 6035 7188 9914 9797

0608 1163 2261 3715 4411 5103 6035 7188 9797 9914
MSD radix sort.

4) 5103 9914 0608 3715 6035 2261 9797 7188 1163 4411

0608 5103 9914 3715 6035 2261 9797 7188 1163 4411

0608 1163 5103 9914 3715 6035 2261 9797 7188 4411

0608 1163 2261 5103 9914 3715 6035 9797 7188 4411

0608 1163 2261 3715 5103 9914 6035 9797 7188 4411

0608 1163 2261 3715 4411 5103 9914 6035 9797 7188

0608 1163 2261 3715 4411 5103 6035 9914 9797 7188

0608 1163 2261 3715 4411 5103 6035 7188 9914 9797

0608 1163 2261 3715 4411 5103 6035 7188 9797 9914

Selection sort.

5) 5103 9914 0608 3715 6035 2261 9797 7188 1163 4411

9914 9797 5103 7188 6035 2261 0608 3715 1163 4411

9797 7188 5103 4411 6035 2261 0608 3715 1163 9914

7188 6035 5103 4411 1163 2261 0608 3715 9797 9914

6035 4411 5103 3715 1163 2261 0608 7188 9797 9914

5103 4411 2261 3715 1163 0608 6035 7188 9797 9914

4411 3715 2261 0608 1163 5103 6035 7188 9797 9914

3715 1163 2261 0608 4411 5103 6035 7188 9797 9914

2261 1163 0608 3715 4411 5103 6035 7188 9797 9914

1163 0608 2261 3715 4411 5103 6035 7188 9797 9914

0608 1163 2261 3715 4411 5103 6035 7188 9797 9914

Heap sort.

3. More Sorting

BSTs and Sorting

Patrick has noticed that since an inorder traversal of a Binary Search Tree returns the elements in order (note that this is only true for Binary Search Trees, it is not true for Binary Trees that are not Binary Search Trees), one way to sort a list is as follows:

1. Create a new empty Binary Search Tree
2. Iterate over the list and add each element to the tree
3. Create a new empty list
4. Perform an inorder traversal of the tree, adding each element to the new list

Q. Let n be the number of elements in the original list. What is the best case running time of this algorithm in terms of n?

$\theta(nlogn)$

Q. How about the worst case running time?

$\theta(n^2)$

Q. What is an example of a list for which this sorting algorithm is inefficient?

A sorted list.

Q. Bonus question: What common sorting algorithm is this procedure equivalent to?

Either, insertion sort or quick sort.

Sorting Things Other than Arrays (CS1B Fa13 MT2)

One can perform a merge operation (such as used in merge sorting) on any number of sorted lists. Suppose that we have an array of Iterator<String> in which each Iterator delivers Strings in sorted order. We want to create a sorted list containing all the items produced by these iterators.

a. Describe how to do this quickly. You may use any of the data structures we've talked about in your implementation. You can use pseudo-code in your description, and don't need to provide actual Java code, but you must be precise in your description. If you use a particular search structure, for example, say exactly what you are storing in that structure and how it gets used.

Call the iterators S_i, for $0 \le i < N$, where N is the number of sources. Create a priority queue containing pairs (E_i, S_i), where E_i is an item from S_i and S_i is positioned just after that element. The queue is ordered by the values E_i, with the smallest having highest priority. Repeatedly

– Remove an item (E_j , S_j) from the queue.
– Add E_j to the output list.
– If S_j has more elements in it, fetch a new value E'_j from S_j , and then add (E'_j, S_j) to the queue

b. As a function of K, the number of Iterators and N, the total number of Strings delivered by all the iterators, give a worst-case bound on the number of string comparisons required by your solution.

$\theta(N log\ K)$

4. Asymptotic Analysis of Data Structures

Building a Queue - CS61BL Su13 Review

Give the big-Theta estimate for enqueueing/dequeueing in the average case for these different Queue implementations: (from Michelle's review worksheet from last summer) Assume n is the size of the list/number of elements in the queue.

a. Using java.util.LinkedList where the end of the list is the beginning of the queue

enqueue: $\theta(1)$

dequeue: $\theta(1)$

b. Using java.util.LinkedList where the beginning of the list is the beginning of the queue

enqueue: $\theta(1)$

dequeue: $\theta(1)$

c. Using java.util.ArrayList where the end of the list is the beginning of the queue

enqueue: $\theta(n)$

dequeue: $\theta(1)$

d. Using java.util.ArrayList where the beginning of the list is the beginning of the queue

enqueue: $\theta(1)$

dequeue: $\theta(n)$

e. Using a singly-linked list without tail pointer where the beginning of the list is the beginning of the queue

enqueue: $\theta(n)$

dequeue: $\theta(1)$

f. Using a singly-linked list without tail pointer where the end of the list is the beginning of the queue

enqueue: $\theta(1)$

dequeue: $\theta(n)$

g. Using a singly-linked list with a tail pointer where the beginning of the list is the beginning of the queue

enqueue: $\theta(1)$

dequeue: $\theta(1)$

h. Using a singly-linked list with a tail pointer where the end of the list is the beginning of the queue

enqueue: $\theta(1)$

dequeue: $\theta(n)$

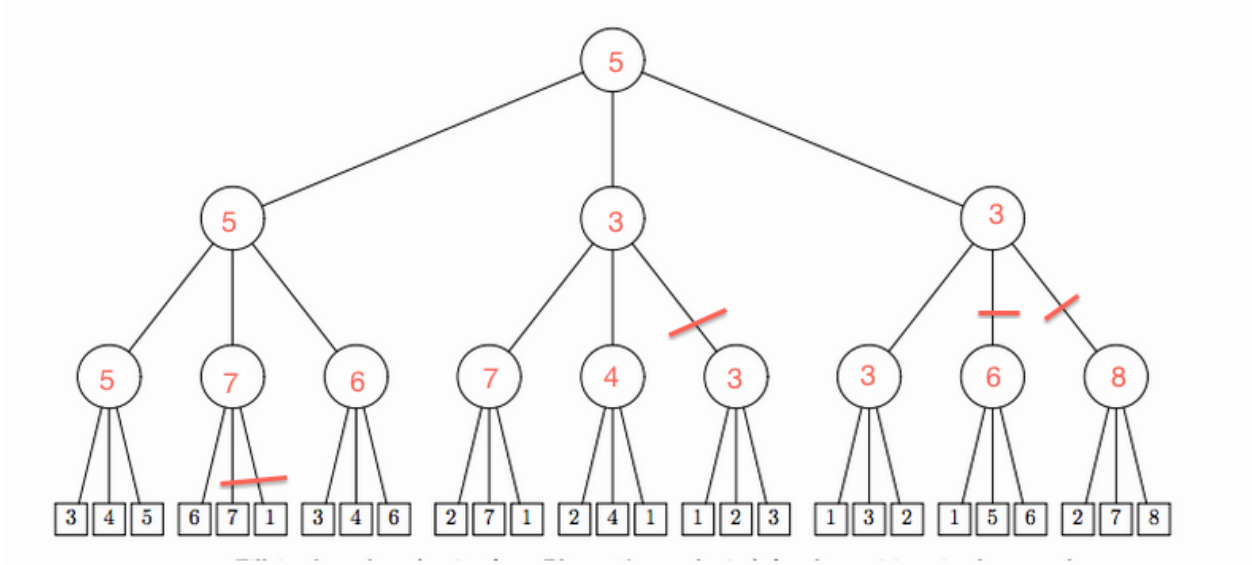
i. Using a binary min heap for a priority queue that has smallest elements at beginning of list

enqueue: $\theta(log\ n)$

dequeue: $\theta(log\ n)$

5. Game Trees (CS61B Fa13 Final)

Consider the following game tree. The values in the square nodes at the bottom are static evaluations of positions where it is Player 2's move. The top node represents a position in which it is Player 1's move. All values represent the position values from the standpoint of Player 1 (larger means better for Player 1).

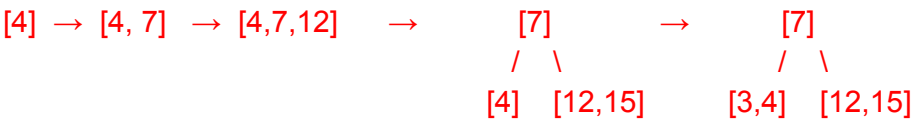


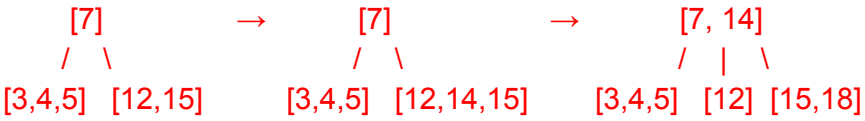
a. Fill in the values (again, from Player 1's standpoint) for the positions in the tree above (represented by circles).

b. Show the effect of alpha-beta pruning, by circling the nodes (square and round) whose values the move evaluator need not compute (assume that each player evaluates positions left to right).

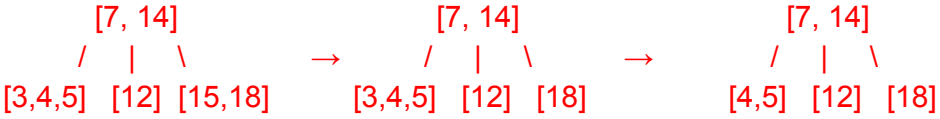
6. 2-3-4 Trees (a.k.a. 2-4 Trees)

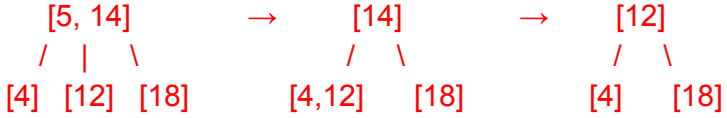
a) Starting with an empty (2, 4) Tree show the result of inserting each of these items in order [4, 7, 12, 15, 3, 5, 14, 18]





b) Show the result of removing the following items(in order) from the above (2, 3, 4) tree: [15, 3, 7, 5, 14]





7. Priority Queues

Conceptual questions

0. What is the heap-order property?
   Heap Order Property: No child has a key less than its parent's key.

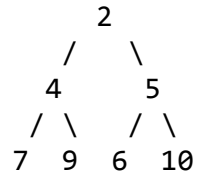
1. (Spring 2013 MT2, Question 1d) We can determine the minimum key in a **binary heap** in $\Theta(1)$ time. The fastest possible algorithm for finding the *maximum* key in an ordinary (min) binary heap with *n* keys runs in Θ(_____) worst-case time. Explain why.
   Finding the maximum key takes $\Theta(n)$ worst-case time, because any one of the leaf nodes could be the maximum key, and a binary heap has roughly $n/2$ leaves.

Heap Operations

0. In an array based heap, where the root is at index 1, given a node at index i...

   Its parent is at __floor(i/2)_____ Its left child is at ___2i_____ Its right child is at __2i + 1__

1. Fill in the corresponding array for this binary heap



X	2	4	5	7	9	6	10

2. **Spring 2009, MT 2, 1a.**
   Draw the following binary heap after `removeMin()`, then again after `insert(2)`

X	1	4	3	7	6	8	9	8	7

X	3	4	7	7	6	8	9	8

X	2	3	7	4	6	8	9	8	7

3. **Autumn 2006, MT2 2c.**
   Show the sequence of swaps by which the method `bottomUpHeap` converts the following array into a heap. Redraw the array once for each swap (in the boxes provided)
   (Note: We've provided room for up to seven swaps, the maximum number possible for eight items. That doesn't necessarily mean that this particular example takes seven swaps)

X	7	2	5	9	3	4	1	8
X	7	2	5	8	3	4	1	9
X	7	2	1	8	3	4	5	9
X	1	2	7	8	3	4	5	9
X	1	2	4	8	3	7	5	9
X								
X								
X								

Heap coding questions

1. Write the `insert` function for an array-based binary heap. The field `current` corresponds to where the next empty element in an array is.
   (Note: Don't worry about resizing the array, assume the element would fit in)

```
public class Heap {
    public int[ ] heapArr;
    public int current;
    public int size;
    public Heap() {
        //initializes the Heap and does other stuff
    }
    public void insert(int element) {
        //Fill in your implementation here!
        int i = current;//keeps track of where in heap we are
        //checks if we can still bubble up
        while(i > 1 && element < heapArr[i/2]) {
            heapArr[i] = heapArr[i/2];//performs bubbling up
            i = i/2;
        }
        heapArr[i] = element;//inserts the element
        size = size + 1;//increments the size and the current pointer
        current = current + 1;
    }
```

```
}
```

2. Write a function `increaseMin` for an array-based heap, that takes in an integer to increase the minimum value of the heap by. After the function is completed, the heap should maintain the heap-order property.

```
//still within the Heap class we defined
// The lazy way
public void increaseMin(int value) {
    int oldMin = removeMin();
    insert(oldMin+value);
}

// Better solution
public void increaseMin(int value) {
    if (size != 0) {
        heapArr[1] += value;
        bubbleDown(1);
    }
}

private void bubbleDown(int i) {
    int toSwitch = i;
    if (current > 2*i && heapArr[i] > heapArr[2*i]) {
        toSwitch = 2*i;
    }
    if (current > 2*i + 1 && heapArr[2*i + 1] < heapArr[2*i]) {
        toSwitch = 2*i + 1;
    }
    if (toSwitch != i) {
        int temp = heapArr[i];
        heapArr[i] = heapArr[toSwitch];
        heapArr[toSwitch] = temp;
        bubbleDown(toSwitch);
    }
}
```

8. Hash Tables and Dictionaries

1)(Spring 2013 MT 2, Question 1c) Suppose we speed up a **hash table** by replacing each chain with a **binary search tree**. If our modified hash table stores n entries, in the very worst case, removing an entry takes Θ(_____) time. Explain why.

$\theta(n)$ time because every key could hash to the same bucket, and the elements might appear in sorted order (resulting in an unbalanced or 'stringy' tree).

2) **Spring 2004 Problem 7.**
If each key in a dictionary has two independent hash codes, we can use a hash table where buckets are arranged as a square array table[N][N]. This hash table has N^2 buckets and n items, so the load factor is n/(N^2).
Suppose we start with an empty 1 x 1 table. The only operations we allow are inserting or finding an item (no remove operation). Whenever an insert operation causes the load factor n/(N^2) to become strictly greater than 1 (where n includes the item being inserted), we resize the table by adding one bucket in each dimension - a table of N^2 buckets is resized to have (N+1)^2 buckets. This resizing takes exactly (N+1)^2 "dollars," where a dollar is a unit of time.

a. If we start with a 1 x 1 table and perform n insert operations, exactly how many resizing operations occur? (Give an exact expression, not an asymptotic one)

$\lceil \sqrt{n} \rceil - 1$.

b Write a summation that expresses the maximum total resizing time (in dollars) for the sequence of n insert operations discussed above. You do not have to solve the summation.

$$\sum_{i=2}^{\lceil \sqrt{n} \rceil} i^2$$

c. If the table is not resized, an insert operation takes exactly one dollar of time. If the table is resized, insert takes one dollar plus the cost of resizing the table. What is the amortized asymptotic time for an insert operation? Express your answer in Big-Oh notation as a function of n, reduced to its simplest form (not a summation). Show your reasoning, and try to derive the tightest possible Big-O bound. Hint: the averaging method works well here.

Compute the sum of all the resizing operations, add an additional *n* dollars for the *n* insert operations, and take the average by dividing the total by *n*.

$$\sum_{i=2}^{\lceil\sqrt{n}\rceil} i^2 \leq \sum_{i=2}^{\lceil\sqrt{n}\rceil+1} (\sqrt{n}+1)^2 \leq \sqrt{n}(n+2\sqrt{n}+1) = n\sqrt{n}+2n+\sqrt{n}$$

$$\text{Average time} = \frac{n\sqrt{n}+3n+\sqrt{n}}{n} = \sqrt{n}+3+\frac{1}{\sqrt{n}} \in O(\sqrt{n})$$

9. Trickier Operations on Trees

Mirror Image

```
/** Given a binary tree rooted at node N, mirrorTree
 *  changes it to its mirror image:
 *       7
 *      / \
 *     2   8
 *        / \
 *       1   3
 * Becomes:
 *       7
 *      / \
 *     8   2
 *    / \
 *   3   1
 */
 public static void mirrorTree(Node n) {
          if (n == null) {
                return;
           }
          mirrorTree(n.left);
          mirrotTree(n.right);
          Node temp = n.left;
          n.left = n.right;
          n.right = temp;
 }
```

Lowest Common Ancestor (CS61B Fa13 MT2)

The lowest common ancestor of two nodes in a tree is the deepest node in the tree (furthest from the root) that has both of the nodes as descendants. It can be one of the two nodes (if one is an ancestor of the other). Write a function that, given the root of a binary search tree containing integers, and two integer keys that are in that tree, finds the key at the lowest common ancestor of the nodes containing the given keys

```
class IntTree {
    public int label;
      public IntTree left, right;
}

public class Utils {
      /** Assuming that T is a binary search tree and ITEM1 and ITEM2
      * are labels in it, return the label in the lowest common
      * ancestor of the nodes containing ITEM1 and ITEM2. */
      public static int findLCA(IntTree t, int item1, int item2) {
                if (item2 < item1) {
                      return findLCA(t, item2, item1);
                } else if (item2 < t.label) {
                      retun findLCA(t.left, item1, item2);
                 } else if (item1 > t.label) {
                      return findLCA(t.right, item1, item2);
                 }
                 return t.label;
      }
}
```