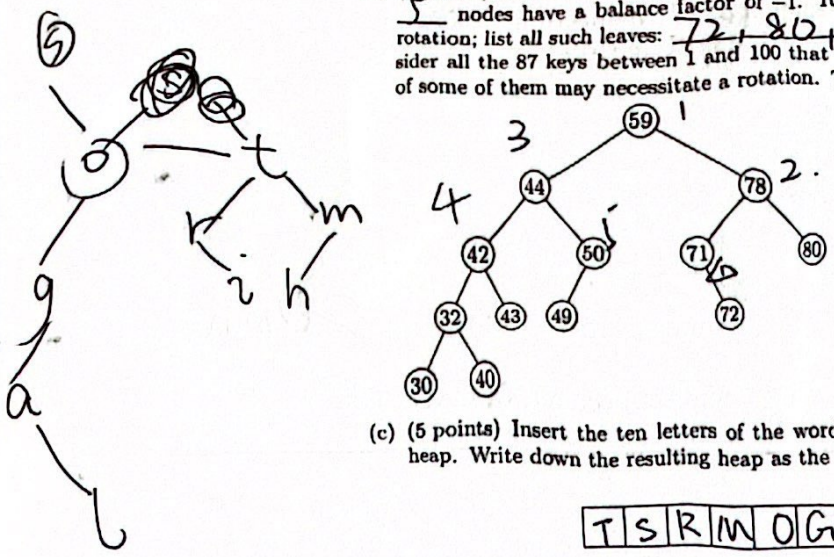
COMP 2011, Final Exam Name: ID:

2. (30 points) Fill in the blank space.

(a) (4 points) A binary tree of characters has postorder sequence "lagirhmtos" and inorder sequence "algorithms." In the level-wise traversal of this tree, the fourth element is (start counting from 1): t.

(b) (6 points) In the AVL tree below, 0 nodes have a balance factor of 1, and 1 nodes have a balance factor of −1. Removing a leaf may necessitate a rotation; list all such leaves: 72, 80, 43, 49, 60. Consider all the 87 keys between 1 and 100 that are not in the tree. The insertion of some of them may necessitate a rotation. There are _____ such keys.



(c) (5 points) Insert the ten letters of the word "ALGORITHMS" into an empty heap. Write down the resulting heap as the array representation.

T	S	R	M	O	G	I	A	H	L

(d) (5 points) Is the following sorting algorithm stable? If it is unstable, justify your answer with an example array of the *shortest* length.
The algorithm is $O(n^2)$. Example (if unstable): X.

```
void sort(int[] a) {sort(a,a.length-1);}
void sort(int[] a, int end) {
    if (end == 0) return;
    boolean flag = false;
    for (int i = 0; i < end; i++)
       if (a[i] >= a[i+1]) {
          flag = true;
          swap(a, i, i+1);
       }
       if (!flag) return;
       sort(a, end - 1);
}
```

(e) (10 points) Complete the algorithms below to return the *smallest* index i in a sorted array such that $a[i]$ equals the key (the first occurrence of the key). For example, searching the key of 1 in the array [1, 1, 2, 4] must return 0.
The first algorithm is iterative. Line 7: mid. Line 8: mid-1.

```
int search(int a[], int key){
    int low = 0, high = a.length - 1;
    int result = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (a[mid] == key) {
            result = __________;
            high = __________;
        }
        else if (a[mid]<key) low = mid+1;
        else high = mid - 1;
    }
    return result;
}
```

The second algorithm is recursive. Line 8: <=, mid+1 Line 9: mid.

```
int first(int a[], int key) {
    return first(a, 0, a.length - 1, key);
}
int first(int[] a, int l, int h, int key) {
    if(l > h) return -1;
    if(l == h) return (key == a[l])? l:-1;
    int m = l + (h - l) / 2;
    return (key ____ a[m]) ? first(a, ____, h, key)
           : first(a, l, ____, key);
}
```

Rules for problems 3–5:

- Provide the implementation if you need any data structure.
- You get three points for each problem by writing "I give up" and nothing else.

3. (10 points) Write an algorithm to check whether there are redundant parentheses in an infix expression, where the operators are '+', '-', '*', and '/', and all operands are single-digit numbers. A pair of parentheses is considered redundant if they enclose an operand or another pair of parentheses. For example, both expressions "((5 + 2))" and "((5) + 2)" contain redundant parentheses. Note that we do not consider any pair of parentheses in the expression "((2 - 0) + (1 - 1))" redundant.

You may assume that the input expression is correct and balanced.

```
// Running time: O(n); space: O(n)
boolean redundantParentheses(char[] s)
```

```
int[] st;
for i
  if s[i] == '(' { st.push(i); last='('
  if    == ')' {
    u = s.top().
    if (u==s[u]=='('
    if (i+1<s.length) && s[i+1]==')'
      && s.size() && s.peek()==u-1)
        return true;
    if (last == u) return true;
  }
return false.
```