CS106B Instructor: Cynthia Lee

Autumn 2019 Solutions

FINAL EXAM - SOLUTIONS

---

1. Graphs

```
int findLargestTree(Grid<bool>& graph) {
    int largestTreeSize = 0;
    int largestTreeRoot = -1;

    for (int v = 0; v < graph.numRows(); v++) {
        Set<int> visited;
        int treeSize = findLargestTree(v, graph, visited);
        if (treeSize > largestTreeSize) {
            largestTreeRoot = v;
            largestTreeSize = treeSize;
        }
    }
    return largestTreeRoot;
}

int findLargestTree(int v, Grid<bool>& graph, Set<int> visited) {
    if (visited.contains(v)) return -1; // contains cycle
    visited.add(v);
    int treeSize = 1;
    for (int neighbor = 0; neighbor < graph.numRows(); neighbor++) {
        if (graph[v][neighbor]) {
            int subTreeSize = findLargestTree(neighbor, graph, visited);
            if (subTreeSize < 0) return -1;
            treeSize += subTreeSize;
        }
    }
    return treeSize;
}
```

2. Pointers and Linked Lists

```
struct listnode {
   int val;
   listnode * next;
};

bool contains(listnode* list, listnode* sub) {
    if (sub == NULL) {
        return true;
    } else if (list == NULL) {
        return false;
    }

    if (list->val == sub->val) {
        return contains(list->next, sub->next);
    } else {
        return contains(list->next, sub);
    }
}
```

3. Recursion

```
Set<int> maxSumSubset (treenode* root) {

    if (root == NULL) return Set<int>();

    Set<int> childSet = maxSumSubset(root->left) +
                        maxSumSubset(root->middle) +
                        maxSumSubset(root->right);

    int childSum = 0;
    for (int i : childSet) {
        childSum += i;
    }

    if (childSum > root->key) {
        return childSet;
    } else {
        Set<int> us;
        us += root->key;
        return us;
    }
}
```

4. BSTs and Heaps

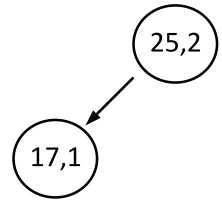
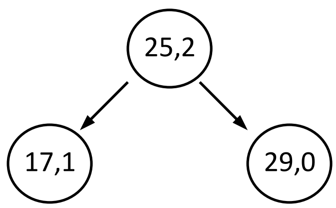
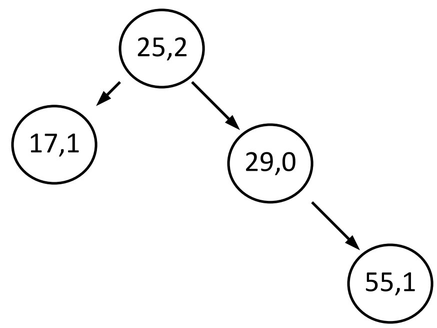
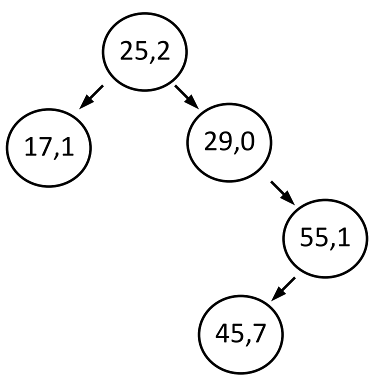
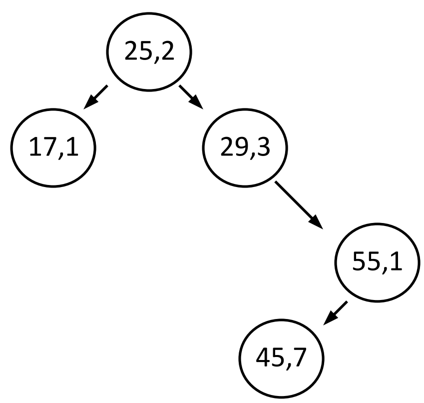
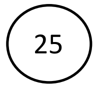
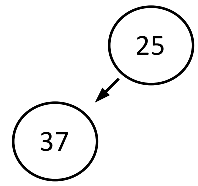
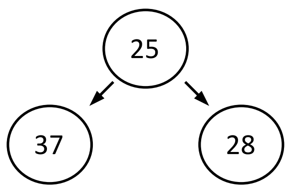
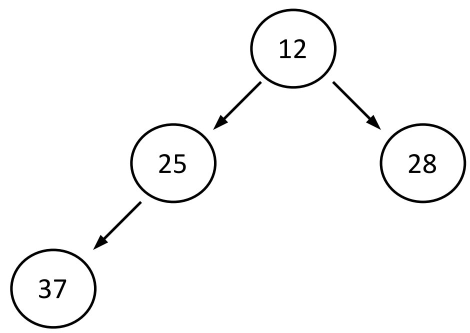
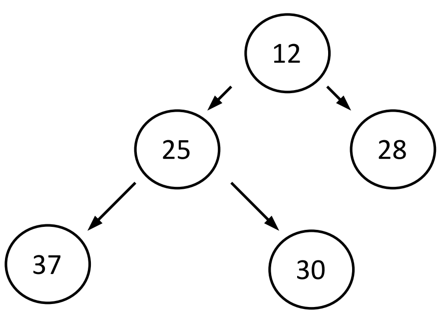
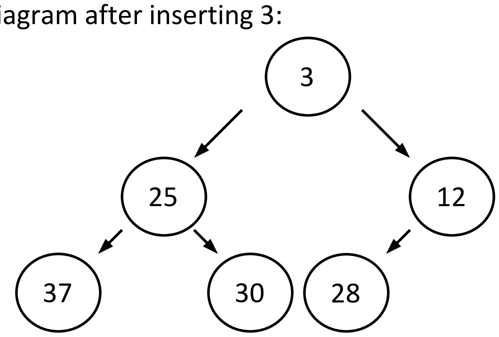
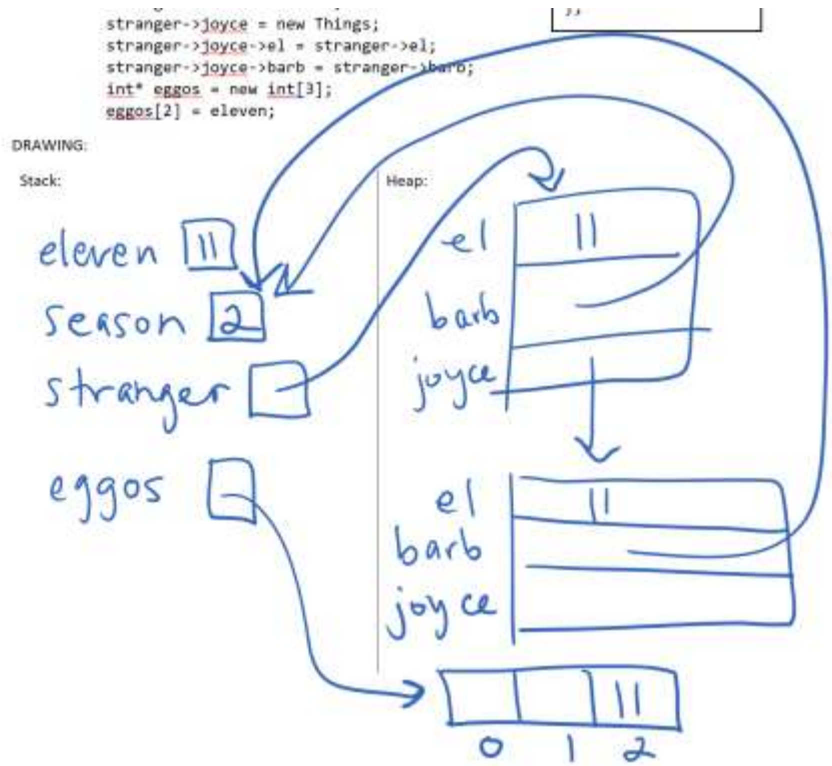
Diagram after inserting (25,2):  *This one is completed for you.*	Diagram after inserting (17,1): 
Diagram after inserting (29,0): 	Diagram after inserting (55,1): 
Diagram after inserting (45,7): 	Diagram after inserting (29,3): 

Diagram after inserting 25:  *This one is completed for you.*	Diagram after inserting 37: 
Diagram after inserting 28:  	Diagram after inserting 12: 
Diagram after inserting 30:  	Diagram after inserting 3: 

5. **Memory Diagram**



Notes:
-- Make a box for pointers to hold the beginning of the arrow. This makes it clear where the space for the storage of the memory address is located.
-- Remember that the only way something ends up on the heap is as the result of a call to "new."
-- You do NOT need to show separate local variables on the stack "attached" to each other, but you must show structs and arrays (stack or heap) as adjacent/attached boxes.
-- Array pointers should point to 0^{th} element of the array.

---

6. Algorithms

(a)

O(log n)

If n is even, we divide by two, otherwise we add one to n. Clearly, **`Binky`** cannot add one to n twice in a row. There must therefore be at least as many steps where we divide n by 2 as there can be steps where we add one to n. As n gets large, the number of times we have to divide n by two will be the factor that determines how quickly we approach zero or one. There can be at most log(n) of those steps, so the running time is therefore O(log n)

(b)

Does this strategy work? YES **NO** (circle) Briefly explain why or why not:

If we are deleting the last cell in the list, **`ptr->next`** is **`NULL`**. When try to access assign to **`*(ptr)`** in the next line, the right hand side will dereference **`NULL`** and crash.

(c)

	Worst-case big-O
1.	**O(n^2)**
2.	**O(n log(n))**
3.	**O(n^2)**