# CS 15 Midterm Review Sheet

Spring 2025

## About this Review Sheet

- **This is not a practice exam.** The length of this review and the types of questions do not necessarily reflect what the actual exam will look like.
- This review *does* give you a chance to practice and apply your knowledge.

## Topics Covered so far include (no particular order)

- Data Structures and ADTs
  - Array Lists
  - Linked Lists
  - Stacks/Queues
  - Sets
  - Trees
    - N-ary Trees
    - Binary Trees
    - Binary Search Trees
    - AVL Trees
- Complexity
- Lists
  - Array Lists
  - Linked Lists
- Problem Solving
  - Recursion
- Huffman Coding

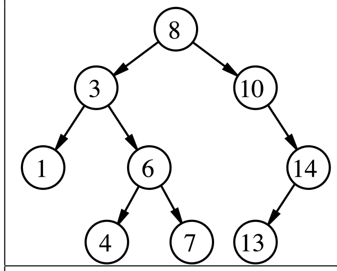
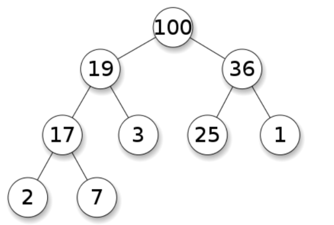
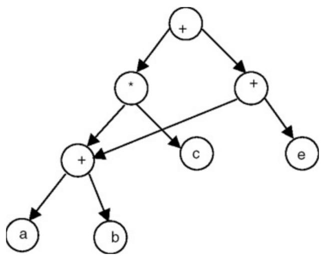
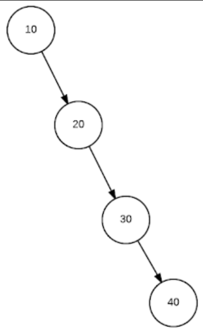
## Practice Problems

1. Time Complexity

| | |
|---|---|
| $T(n) = n^2 + \frac{3}{2} n \log n + 3$ | |
| `for (int i = 0; i < n; i++) {`<br>`    for (int j = 0; j < 7; j++) {`<br>`        // constant work`<br>`    }`<br>`}` | |
| `for (int i = 0; i < n; i++) {`<br>`    for (int j = 0; j < n; j++) {`<br>`        // constant work`<br>`    }`<br>`}` | |
| `for (int i = 0; i < n * n; i++) {`<br>`    for (int j = 0; j < n; j++) {`<br>`        // constant work`<br>`    }`<br>`}` | |
| `for (int i = 0; i < n * n; i++) {`<br>`    for (int j = 0; j < 5n - 3; j++) {`<br>`        // constant work`<br>`    }`<br>`}` | |
| `for (int i = 0; i < 2 * n; i++) {`<br>`    for (int j = 0; j < i; j++) {`<br>`        // constant work`<br>`    }`<br>`}` | |

2. To Tree or Not To Tree

   Given the following linked structures, determine whether it is a **Tree, Binary Tree, or Binary Search Tree**. Be as **specific** as possible; if a structure is both a BST and a Binary Tree the answer would be BST. If, the structure is none of the above, then say **Not a Tree**.

| | |
|---|---|
|  |  |
|  |  |
| *empty* *tree* | |

3. Miscellaneous

   For each of **Unsorted Array, Sorted Array, Unsorted Linked List, Sorted Linked List**, say whether binary search can be performed. Justify your answer.

4. Zip

   **Write a function that *zips* two linked lists.** The zip operation creates a list by inserting nodes from two other lists in alternating positions.

   $A = 5 \rightarrow 7 \rightarrow 17$

   $B = 12 \rightarrow 10 \rightarrow 2 \rightarrow 4 \rightarrow 6$

   $Zip(A, B) = 5 \rightarrow 12 \rightarrow 7 \rightarrow 10 \rightarrow 17 \rightarrow 2 \rightarrow 4 \rightarrow 6$

   **Note that the extra nodes of the longest list were appended to the end**

```
struct Node {
  DataType data;
  Node    *next;
};
/**
* @brief      "Zips" the two lists beginning by a and b
*
* @param      a     The front of list a
* @param      b     The front of list b
*
* @return     The pointer to the head of the zipped list
*
* @note       This operation will be done *inplace*.
*             That is, there should be no dynamic memory
*             allocation (new) or memory deallocation (delete)
*/
Node *zip(Node *a, Node *b) {













}
```

}

5. Stacks and Queues.

   Implement a Queue using 2 Stacks. You may assume that your stack has `pop`, `push`, `empty`, and `top` functions defined in the usual way.

```
struct Queue {
    stack<int> s1, s2;
    void enqueue(int i);
    int  dequeue();
};

void Queue::enqueue(int i) {
```

```
}

int Queue::dequeue() {
```

```
}
```

6. N-ary Trees

   (a) Write a recursive function that recursively counts the number of leaf nodes in the tree. You may assume the following `Node` definition:

```
struct Node {
    vector<Node *> children;
    int numChildren;
    bool isLeaf();
};
```

```
int Tree::countLeaves(Node *curr) {









}
```

   (b) What traversal would you use to delete all nodes in a tree? Why?

7. Binary Trees

   (a) Write a recursive that recursively finds the height of a Binary Tree. You may assume your `Node` has `left`, `right`, and `data` member variables defined in the usual way.

```
int BinaryTree::height(Node *curr) {









}
```

   (b) Write a recursive function that returns whether an element is in a Binary Tree

```
int BinaryTree::find(Node *curr, int data) {









}
```

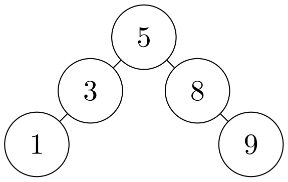
8. Binary Search Trees

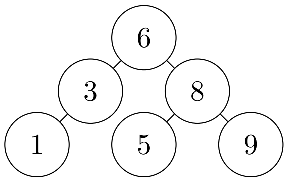
   Write a recursive function that returns whether an element is in a Binary Search Tree

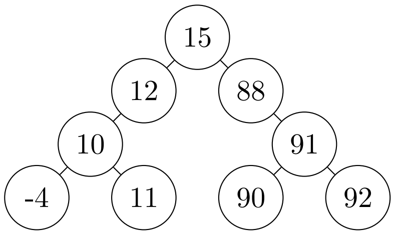
```
bool BinarySearchTree::find(Node *curr, int data) {





}
```

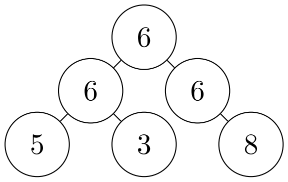
9. AVL Trees

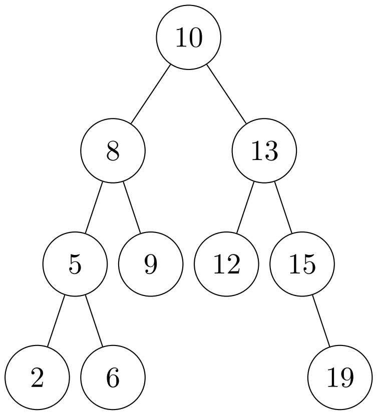
   1. Insert the following numbers, in order from left to right, into a Binary Search Tree and an AVL Tree.
      (a) Tree 1: [10, 15, 18, 5, 7, 21, -3]
      (b) Tree 2: [1, 2, 3, 4, 5, 6]
   2. Are the following trees valid AVL trees?

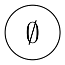
(a)


(b)


(c)


(d)


(e)


(f)


   3. Runtime Questions: Fill in the worst-case runtime in big-O notation; then for all the below, describe the tree that gives the worst case runtime.

| | BST | AVL Tree |
|---|---|---|
| insert | | |
| delete | | |
| find | | |
| findMax | | |
| findMin | | |

10. Huffman Coding
    (a) Build a Huffman tree for the word “engineering”.

    (b) Using tree you created in part (a), what is the size of the encoding for the word “engineering”?

11. Recursion

   (a) Fill in the blank

   Every recursive function needs a ________ ________ to know when to stop, and needs to ________ itself to continue recursing

   (b) Reverse a Linked List

```
class LinkedList {
public:
    // Reverse function that client will call
    void reverse();
private:
    // Private reverse for recursion
    Node *reverse(Node *curr, Node *newNext);
};
void LinkedList::reverse() {
    front = reverse(front, nullptr);
}
LinkedList::Node *LinkedList::reverse(Node *curr, Node *newNext) {






}
```

12. Your boss needs an ArrayList that holds **positive** numbers. But they get confused really easily, so they ask you to reduce the number of member variables. That is instead of the usual `size, capacity ints` and a pointer to `data`, your ArrayList should *only* have a pointer to data and a pointer to "something."

```
class ArrayList {
public:
    void insert(int i);
private:
    ...
    int *data;
    int *something;
};
```

(a) Describe how you would design this ArrayList for your boss. Be sure to mention what each pointer points to.

(b) Describe how you would insert an element into your ArrayList, and how it would expand.