Today, we’ll be doing review for the midterm. We’ll go over questions/ important points on each of the following topics:

- Unbounded Arrays
- Hash Tables
- Amortized Analysis
- Heaps/ Priority Queues
- Binary Search Trees
- AVL Trees
- Polymorphism
- Memory Management in C

The midterm is cumulative, so you need to know earlier material also. But the main focus will be on the material since the last midterm i.e the topics stated above

Unbounded Arrays

Quick Recap

- The costs are amortized
- Cost of insertion and deletion is $O(1)$ amortized
- Double the size of the array when $size == limit$
- Don’t reduce by half when it is half full - may lead to $O(n)$ operations
- While removing elements, resize when the array is a quarter full

Hash Tables

Quick Recap

- $O(1)$ finding of an element
- Hash function must be randomized and deterministic
• Collisions occur when multiple keys have the same value
• Collisions are resolved by separate chaining or probing (linear/quadratic) - Focus on separate chaining
• Average chain length is the load factor $\mu$
• $\mu = \frac{\text{(number of keys in hash table)}}{\text{(size of the underlying table)}}$

Amortized Analysis

Important examples include

• Unbounded Arrays
• Binary Counter
• Queue as two Stacks - which we’ll go over if we have time at the end of recitation
Heaps/Priority Queues

Quick Recap

• We usually deal with min-heaps
• The root is the minimum element
• Finding minimum element is $O(1)$
• Deleting minimum element is $O(\log n)$
• Inserting an element is $O(\log n)$

• **Ordering Invariant**: Any element in the heap is smaller than both its children

• **Shape invariant**: An element is inserted in a manner that at any point only the last level is unfilled and elements are filled in this level from left to right

Quick Exercise:
Insert elements from 1 to 15 into a heap in such a way to get the smallest possible number in the last level:
Binary Search Trees

Quick Recap

- **Ordering Invariant**: states that every element in the left subtree must be less than the root and every element in the right subtree must be greater than the root.

- There is no shape invariant - so the worst case is a linked list

Quick Exercise:
Construct a BST by inserting 4,2,3,1,5,7,6 in the given order

Write code to print out the elements in ascending order
(Assume that you have a print_elem function)
What was the cost of printing the entire tree in ascending order?

So we printed out the elements in sorted order in less than $O(n \log n)$ time!! Did we just come up with a more efficient sorting algorithm?

---

### AVL Trees

**Quick Recap** For unbalanced trees, where $P$ is the node at which it is unbalanced:

**Single rotation** -
```
P   C
  \   / \  
 C  ->  P  G
    \   / 
     G
```

**Double rotation** -
```
P   P   G
  \   \   / 
 C  ->  G  -> P  C
 /   \   / 
G    C
```

**Quick Exercise:**
Is the tree in the previous part balanced? Is it left heavy or right heavy? Perform operations to balance the tree.
Polymorphism

Quick Recap
void* is the generic datatype which we usually use to pass pointers around. These need to be cast into appropriate types before use.

Quick Exercises:
(Thanks to Soojung Ha, a former 122 TA, for the following questions)

1. Complete the following gt (greater than) function, which returns 1 if the integer m is pointing to is greater than the integer n is pointing to

```c
int gt(void* m, void* n){
    int *m_ptr, *n_ptr;
    m_ptr =
    n_ptr =
    return
}
```

2. What does the following function do?
   (Hint: In C, the assignment statement (=) returns the assigned value)
   ```c
   void mystery(char* p, char* q){ while (*p++ = *q++); }
   ```

Memory Management in C

Quick Recap

- **The Golden Rule**
  Always free what you malloc
  Remember that C is not garbage collected

- Don’t free memory that you didn’t allocate. This can lead to undefined behaviour

- Usually you are only responsible for freeing memory that you have allocated.
  I like to think of this as

  number of mallocs = number of frees