CSE 373

APRIL 26TH – EXAM REVIEW
EXAM FRIDAY

• Exam Review Tonight
  • 5:30pm - 7:00 – EEB 105
• Section
  • Also Exam review
• Practice Midterm Solutions
  • Out tonight after review session
EXAM FRIDAY

• Topics
  • Definitions
  • Stacks and Queues
  • Heaps
  • Runtime Analysis
  • Dictionaries
  • BSTs
  • Traversals

• AVL Trees
• Hash Tables
DEFINITIONS

• Important terms
  • Abstract Data Type
    • Example: Dictionary
      • Supports functions: insert, find, delete
      • Has expected behavior
  • Data Structure
    • Language independent structure which implements an ADT
      • Example: AVL tree
      • Can be analyzed asymptotically
DEFINITIONS

• Important terms
  • Implementation
    • Low-level design decisions
    • Language specific

• Example
  • The Queue ADT supports enqueue, dequeue and front.
  • Arrays and Linked Lists are examples of the data structures
  • Implementation: front and back pointers
STACKS AND QUEUES

• Our first two ADTs
  • Stack:
    • Supports: push(), pop(), top()
    • LIFO order
  • Queue:
    • Supports: enqueue(), dequeue(), front()
    • FIFO order
STACKS AND QUEUES

• Data structure choices
  • Arrays and Linked Lists
  • Considerations
    • Memory usage
    • Ease of implementation
    • Resizing time
  • Runtimes:
    • O(1) for all functions
HEAPS

- Priority Queue ADT
  - Supports: insert(), findMin(), deleteMin(), changePriority()
  - Data is stored in priority, value pairs
  - In this class, we use the min-heap, where a lower value means it should dequeue first
HEAPS

• Data Structure
  • Heap
    • Complete binary tree
    • Heap property
  • Implementation
    • Array
    • Find parents/children arithmetically
• Runtimes
  • Insert: $O(\log n)$, findMin: $O(1)$, deleteMin $O(\log n)$
  • ChangePriority: $O(\log n)$
  • buildHeap, $O(n)$
RUNTIME ANALYSIS

• Counting the number of operations
  • Comparisons, mathematical operations, assignments
• For loops and while statements
  • Count the number of times relevant code is executed
• Important summations
  • Sum of all numbers from 1 to n
  • Sum of the powers of two
RUNTIME ANALYSIS

- Asymptotic Analysis
  - Best-case, worst-case, average-case
  - Usually we discuss worst-case complexity
  - If we increase the input size, how does the computation time change

- BigO notation
  - Upper bound for a given function
  - \( f(n) = O(g(n)) \) if there exists a \( c \) and \( n_0 \) for which \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \)
RUNTIME ANALYSIS

• Basic ideas
  • $O(1)$: Input size has no effect on runtime
  • $O(\log n)$: doubling the input increases the runtime by some constant amount
  • $O(n)$: linear time, each additional input increases execution time by a constant amount
  • $O(n^2)$: doubling the input increases the runtime by a factor of 4.
  • $O(2^n)$: exponential, increasing the input by one doubles the runtime
DICTIONARIES

- ADT
  - Supports the following functions
    - Insert(key k, value v)
    - find(key k)
    - delete(key k)
  - Data is stored in key, value pairs
  - In this course, duplicate keys are not allowed
  - Most data structures can implement a dictionary
BINARY SEARCH TREES

• Binary trees
• Nodes with two children
• Maintains search property
  • All values in the left subtree must be less than the parent
  • All values in the right subtree must be greater than the parent
• With each increase in height, the number of nodes in a tree roughly doubles
• A completely full tree has $2^h-1$ nodes
• Roughly half of a binary search tree are nodes
TRAVERSALS

• Two main traversal families
  • Depth First Search
  • Breadth First Search

• DFS
  • Usually implemented recursively
  • Whether the parent is processed before, after or in the middle of its children determines if the traversal is pre-order, post-order or in-order respectively

• BFS
  • Put the root into a queue
  • Dequeue a node, process it and enqueue its children
  • Top to bottom left to right traversal
  • Queue is largest at the widest part of the tree
AVL TREES

• Specific type of binary search tree
• Still must implement binary search
• Nodes in AVL trees have two extra fields, height and balance
• Balance = | height(left) – height(right) |
• Balance for each node must be less than or equal to 1
• Trees with this condition still have $O(\log n)$ height
• No covering delete in this course
• Find: $O(\log n)$: Insert $O(\log n)$
AVL ROTATIONS

• AVL Rotations occur when an insertion makes a node out of balance
  • Relative to the node that is unbalanced, there are four rotations depending on which grandchild received the new node.
  • Left-left and right-right rotations involve the child of the affected node being rotated up into position
  • Left-right and right-left rotations involve the grandchild being rotated up into position. The grandparent and parent become the two children
  • It is important that these rotations preserve BST property
HASH TABLES

• A large data set $M$ with a smaller set that should be saved, $D$

• A hash function maps $M$ onto $D$
  • It should run in $O(1)$ time
  • It should distribute into all of the available spots evenly

• Hashtables provide $O(1)$ runtime IF
  • Collisions are not a problem
  • Decrease the chance of collisions by increasing the amount of memory
    • Resizing is costly
  • Resolve collisions by finding the next open space: linear probing
HASH TABLES

- Linear probing results in clustering
  - This slows down the expected runtimes of the hash table
  - Needs lots of free space in order to have fast runtimes
- A good overall data structure
  - Faster runtimes, but more maintenance
  - Important to know when making design decisions
DESIGN DECISION PROBLEM

• Think about runtime
• Memory constraints
• Function prioritizing
• Experimental considerations
GOOD LUCK!

• Practice Exams
• Review tonight
• Review in section tomorrow
• Email any questions
• No office hours Friday or next Monday
• Grades back in class on Monday