Final Exam

• Dec 4, 8-10 am, **PH 131**
• Covers labs 1-10
• Closed-book, closed-notes. Calculators are not permitted
• Higher scores on later exams will replace lower-scoring, earlier exams
• Multiple choice questions + code-written questions
• Useful materials: notebooks, labs, past year exams, additional materials on piazza (slides, animation), textbooks
Exam Topics

• Python syntax & semantics
  – Basic language structures
  – Built in types
    • String, tuple, range, list, dictionary ...
  – Iteration, iterators, and generators
  – OOP
    • Classes and Inheritance
    • Inner classes/methods
    • Static methods

• ADTs & Data structures
  – Array-backed list
  – Linked list
    (single/double/circular/sentinel-head)
  – Hashtable
  – Stack
  – Queue
  – Priority queue / Heap
  – Binary search tree

• Misc Algorithms
  – Insertion sort
  – Linear search
  – Binary search
  – Heap sort
  – Merge sort

• Recursion
  – "Base cases" and recursion as delegation
  – Single recursion
  – Multiple recursion
Array-backed List (or Array)

• **Pros**
  – Easy to create, Easy to use
  – Direct indexing: $O(1)$
  – Sequential access: $O(N)$

• **Cons**
  – Sorting: $O(N \log N)$
  – Searching: $O(N)$, and $O(\log N)$ if sorted
  – Inserting and deleting: $O(N)$ because of shifting items
Linked List

• **Pros**
  – Inserting and deleting: O(1)
  – Sequential Access: O(N)
  – Inserting and deleting operations refers to the operation itself, as you might need to sequentially access all the nodes until the node you’re looking for.
  – Inserting and deleting is easier with doubly linked list.

• **Cons**
  – No Direct Access; Only Sequential Access
  – Searching: O(N)
  – Sorting: O(NLogN)
Stacks and queues have very specific purposes.
- Stacks are last in, first out (LIFO)
- Queues are first in, first out (FIFO)

**Pros**
- Push/Enqueue: O(1)
- Pop/Dequeue: O(1)
- Peek: O(1)

**Cons**
- If trying to do anything else with stacks or queues (e.g., how can I pull an item from the middle?), then better looking at a different data structure
**Pros**

- Inserting and deleting: $O(1) + \text{Hashing \\& Indexing}$
- Direct access: $O(1) + \text{Hashing \\& Indexing}$
- It takes a little processing for the hashing and indexing. But the good thing about that is it’s the same amount of processing every time, even if the hash table gets very large.
- ["Load factor" and Rehashing] When the hash table gets full, it will increase it’s size. And, when the number of filled buckets is much smaller than the size of the hash table, it will then decrease it’s size. Both operations take a complexity of $O(N)$. That’s why insertion and deletion takes $O(1)$ amortized.

**Cons**

- Some overhead as require a little more space in memory than arrays
- Retrieval of elements does not guarantee a specific order
- Searching for a value (without knowing its key)
Binary Search Trees (BST)

• **Pros**
  – Inserting and deleting
  – Speed of Access
  – Maintains sorted order; retrieval of elements is in order.
  – The complexity of insertion, deletion, and accessing would be $O(\log N)$, and $O(N)$ if the tree is unbalanced

• **Cons**
  – Some overhead because of their creation and management
Heap

- Heaps are a type of binary tree that are great for priority queues.

**Pros**
- Find Min/Find Max: O(1)
- Inserting: O(LogN)
- Delete Min/Delete Max: O(LogN)

**Cons**
- Searching and deleting: O(N)
- In searching and deleting, we will have to scan all the elements as they don’t guarantee a specific order, unlike BST.
- Deleting requires to traverse the whole tree to access the element first, then delete it, where the deletion operation itself requires O(LogN).
# Worst-Case Run-Time Complexity

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Access</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array-Backed List</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Sorted Array-Backed List</td>
<td>$O(1)$</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Singly Linked List</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Doubly Linked List</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Stack</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Queue</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Heap</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>HashTable</td>
<td>N/A</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Binary Search Tree</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>AVL Tree</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>

Sometimes average-case run-time complexity makes more sense e.g., HashTable, BST, Heap

CS 331 Data Structures and Algorithms