Order of Complexities

1. Arrays, ArrayLists, Singly LinkedLists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Array | ArrayList | LinkedList |  |
| Find element by index | O(1) | O(1) | O(n) |  |
| Find element by value | O(n) | O(n) | O(n) |  |
| Clone a list | O(n) | O(n) | O(n) |  |
|  |  |  | With tail | Without tail |
| Add element to interior | O(n) | O(n) | O(n) | O(n) |
| Remove element from interior | O(n) | O(n) | O(n) | O(n) |
| Add element to end | O(1) | O(1) | O(1) | O(n) |
| Remove element from end | O(1) | O(1) | O(n) | O(n) |

NOTE: Adding and removing an element to the END in a doubly linkedlist WITH a tail reference will both be O(1).

1. Stacks (First In Last Out)

|  |  |  |
| --- | --- | --- |
| Arrays | top always index 0 | bottom always index 0 |
| push | O(n) | O(1) |
| pop | O(n) | O(1) |
|  |  |  |
| Singly LinkedLists | top is head | top is tail |
| push | O(1) | O(1) |
| pop | O(1) | O(n) |
|  |  |  |

NOTE: For a doubly linked-list, popping an element where the top is tail will be O(1)

1. Queues (First In First Out)

|  |  |  |
| --- | --- | --- |
| Arrays | front always index 0 | rear always index 0 |
| enqueue | O(1) | O(n) |
| dequeue | O(n) | O(1) |
|  |  |  |
| Singly LinkedLists | front is head | front is tail |
| enqueue | O(1) | O(1) |
| dequeue | O(1) | O(n) |

NOTE: For a doubly linked-list, dequeueing an element where front is tail will be O(1)

NOTE: For array implementation, make sure you understand how the circular queue works. The front and rear references will be dynamic, therefore both enqueue and dequeue will be O(1).

1. Binary Trees

|  |  |  |  |
| --- | --- | --- | --- |
|  | Search Tree | Balanced Search Tree | [Max] Heaps |
| Finding min value | O(n) | O(log n) | O(n) |
| Finding max value | O(n) | O(log n) | O(1) |
| Finding a target | O(n) | O(log n) | O(n) |
| Print all elements | O(n) | O(n) | O(n) |
| Insert | O(n) | O(log n) | O(log n) |
| Remove | O(n) | O(log n) | O(log n) |

NOTE: For [Min] Heaps, finding min value will be O(1), and finding max value will be O(n).

NOTE: A heap without specification is considered to be a max heap by default.

1. Searching Algorithms

Linear/Sequential Search: O(n)

Binary Search: O(log n)

NOTE: For binary search to work, the list MUST be sorted first.

1. Sorting Algorithms

Quadratic Sorts: O(n2­­)

Selection Sort

Insertion Sort

Bubble Sort

Divide and Conquer Sorts: O(n \* log n)

Merge Sort

Quick Sort –

NOTE: Average and best cases are O(n \* log n), but worse case is O(n2)

Heap Sort: O(n \* log n)

NOTE: Making a heap from any array will take O(n) time

Counting Sort: O(n)

Radix Sort: O(k \* n) // where k is the number of digits

NOTE: Counting Sort only works on a list of *integers* **in a limited range**.

NOTE: Radix Sort only works on a list of *integers*.

Major Topics are covered in the 7 programming projects

HW1: Abstract Data Type

HW2: LinkedLists

HW3: Stacks

HW4: Queues

HW5: Trees

HW6: Hashing

HW7: Graphs

Arrays, ArrayLists, and LinkedLists are the basis of implementation for other data structures. If you understand the time complexities for basic operations on these 3 data structures, you can apply them to the other ones.

e.g. Pushing into a Stack where the top is at position 0 is the same as adding an element in an array at position 0.

Dequeueing from a Queue where the front is pointing at the tail reference of a LinkedList is the same as removing the last element in a LinkedList.

**Any topic not mentioned in this review sheet does NOT mean it will not be on the exam. Likewise, there is no guarantee all the topics mentioned will be on the exam.**

**Please review the lecture slides, recitation material, as well as your old exams.**

Best of luck on all of your exams!

o(∩\_∩)o

Make sure you understand what was covered in lecture. Again, this review sheet is NOT a substitute for the lecture slides and will not cover all topics in detail. Use it as a checklist.

Unit 3 – Stacks

* Check for balance parenthesis
* Evaluating Expressions
* Prefix, Infix, Postfix conversions

Unit 4 – Queues

* Circular Queues (array implementation)
* Priority queues
* Random number generation

Unit 5 – Recursion

* Activation Record
* Backtracking
* Tail recursion

Unit 6 – Trees

* Preorder, Inorder, Postorder Traversals
* Add and Remove from Binary Search Tree

Unit 7 – Balanced Trees

* Heaps (Add/Remove – fixHeap())
* B – Trees
* 2 – 3 – 4 Trees
* Red – Black Trees
* AVL Trees

**NOTE: Balanced search Trees and heaps have a height/depth of O(log n).**

**NOTE: Do not stress over AVL Trees. All you have to know for AVL Trees is they are balanced.**

Unit 8 – Hashtables

* Sequential Search
* Binary Search
* Collision Resolution:
  + - Linear Probing
    - Quadratic Probing
    - Double Hashing
    - Chain Hashing
* α- load factor

Unit 11 – Sorting Algorithms

* Quadratic Sorts
* Divide and Conquer Sorts
* Heap Sort
* Counting Sort

Unit 12 – Graphs

* Adjacency Matrix
* Edge List
* Depth First Traversal
* Breadth First Traversal