River Kelly
Basic Data Structures And Algorithms Final Review
05/02/2016

Terms

- Abstract Data Type -
    - An Abstract Data Type (ADT) is a model of a data structure the specifies
        - The characteristics of the collection of data
        - The operations that can be performed of the collection
    - It is abstract because it does not specify how the ADT will be implemented
    - A given ADT can have multiple implementations
    - An ADT is like a bag, which has items within it, its is abstract because it doesn't define any certain operations for the data within it.
- Arrays
- Singly Linked List
- Circularly Linked List
- Doubly Linked List
- Stack - a collection of objects that are inserted and removed according to the **last-in, first-out (LIFO)** principle.
    - A user may insert objects into a stack at anytime, but may only access or remove the most recently inserted object, or the "top" of the stack.
    - push(e) - add element e to the top of the stack
    - pop() - removes and returns the top element from the stack (or null if the stack is empty)
    - peek() - returns the top element of the stack without removing it
    - size() - returns the number of elements in the stack
    - isEmpty() - returns a boolean indicating whether the stack is empty
- Queue ADT - A collection of objects that are inserted and removed according to the **first-in, first-out (FIFO)** principle. An element can be inserted at anytime, but the only element that has been in the queue the longest can be removed first.
    - enqueue(e) - add element e to the back of the queue
    - dequeue() - removes and returns the first element from the queue () or null if the queue is empty)
    - first() - returns the first element of the queue, without removing it
    - size() - returns the number of elements in the queue
    - isEmpty() - returns a boolean indicating whether the queue is empty
- Deque or "deck" ADT - Double-Ended Queues - a queue like data type which supports insertion and deletion at both the front and back of the queue.
    - addFirst(e) - insert a new element e at the front of the deque

    - addLast(e) - insert a new element e at the back of the deque
    - removeFirst() - return and remove the first element (return null if empty)
    - removeLast() - remove and return the last element (return null if empty)
    - first() - returns first element without removing it
    - last() - returns last element without removing it
    - size() - returns the number of elements
    - isEmpty() - returns boolean value indicating whether the deque is empty
- Stacks, Queue, and deque
    - Each of these data types represent a linear sequence of elements
- List ADT - a list is a linear sequence of elements, like a stack, queue and deque, but has more support for adding or removing elements at arbitrary positions.
    - Locations of an element within an array are described with an integer **index.**
    - Java interface java.util.List has the following index based methods
        - size() - returns the number of elements in the list
        - isEmpty() - returns a boolean indicating whether the list is empty
        - get(*i*) - returns the element having index *i*,
        - set(*i, e*) - replaces the element at index *i* with element e, and returns the odd element that was replaced.
        - add(*i, e*) - insert a new element *e* into the list so the is has index *i,* moving all subsequent elements one index later in the list
        - remove(*i*) - removes and returns element at index *i*, moving all subsequent elements one index earlier in the list.
- Priority Queue - a collection of prioritized elements that allows arbitrary element insertion, and allows the removal of the element that has the first priority. When an element is added to the priority queue, the user designates its priority by providing an associated **key**.
    - insert(k, v) - creates an entry with key k, and value v in the priority queue
    - min() - returns, but does not remove, a priority queue entry (k,v) having minimal key
    - removeMin() - removes and returns an entry (k, v) having minimal key from the priority queue
- Sorting With a Priority Queue -
-
- Array List -
- Positional Lists -
- Iterators -
- Trees -
- Binary Trees -
- Heaps -

- Bubble Sort - Largest item will bubble to the last position
    - Start by taking the value of the first item. If this item is larger than the next item, they will switch positions. This will continue until the selected item is less than the next item. If this is true, take the larger item and continue to compare and swap until the largest item is in the last position of the array. Then the next largest item, will bubble to the second to last position, so on and so forth until all items are in order.
- Selection Sort -
    - Iterate through the entire array to find the largest item. Once you have found the largest item, move it to the last position in the array. The find the next largest item, and move it to the second to last position, so on and so forth until the array is sorted.
- Insertion Sort -
    - Phase 1 - take the first two items in the array, and sort them. Comparing the last on, to the previous one. Now the first two items should be sorted.
    - Phase 2 - take the 3rd item in the array, and compare it to the previous item, swap them if necessary. Continue until the first 3 items are sorted correctly.
    - Phase 3 - take the 4th item, and compare it with the previous, swap in necessary and continue until the first 4 positions are sorted
    - Continue until the array is sorted
- Merge Sort -
    - Phase 1- Start by sorting the first 2 items into a new array of length 2, so elements at index 0, and 1. Then sort elements at index 2 and 3 into another list of length 2, then sort elements at index 4 and 5, and so forth. Then copy merged items back into original list.
    - Phase 2 - Next merge items 1 and 2, so elements of index 0 - 3(first 4 items), and sort them.
- Quicksort -
    - 1 - pick an element called a pivot from an array
    - 2 - reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it. After all comparisons, the pivot is in its proper position.
    - 3 - Recursively apply the above steps to the subarray of elements with smaller values and separately to the subarray of elements with greater values.

- Time Complexity
    - Bubble sort - O(n^2)
    - Selection sort - O(n^2)
    - Insertion sort - O(n^2)
    - Merge sort - O(n log(n))
    - Quicksort - O(n log(n))
- Insertion bubble and selection and n^2
- Linear and Binary Search
- Recursion
- GUI
- Inheritance and Abstract Classes
- Exceptions
- Casting and Generics