CSE 30331/34331: Data Structures

(Almost) Everything You Need to Know for the Midterm

Abstract data types

	insert	delete
stack	back	back
queue	back	front
deque	front/back	front/back
priority queue	anywhere	min
set	anywhere	by value

Sequence data structures

	find by position	find min/max or by value	insert at position	delete at position	delete min/max or by value	concatenate
array	O(1)	O(n)	back: O(1) else: O(n)	back: O(1) else: O(n)	O(n)	O(n)
circular array	O(1)	O(n)	front/back: O(1) else: O(n)	front/back: O(1) else: O(n)	O(n)	O(n)
linked list	front: O(1) else: O(n)	O(n)	front: O(1) else: O(n)	front: O(1) else: O(n)	O(n)	O(n)
linked list with tail pointer	front/back: O(1) else: O(n)	O(n)	front/back: O(1) else: O(n)	front: O(1) else: O(n)	O(n)	O(1)
doubly linked list	front/back: O(1) else: O(n)	O(n)	front/back: O(1) else: O(n)	front/back: O(1) else: O(n)	O(n)	O(1)

Set data structures

	find min/max	find by value	insert	delete min/max	delete by value	create from list	union
sorted array	O(1)	O(log n)	O(n)	max: O(1) else: O(n)	O(n)	O(n log n)	O(n)
sorted linked list	min: O(1) else: O(n)	O(n)	O(n)	min: O(1) else: O(n)	O(n)	O(n log n)	O(n)
binary heap	min: O(1) else: O(n)	O(n)	O(log n)	min: O(log n)	O(n)	O(n)	O(n)
B-tree red-black tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(n log n)	O(n)
hash table	O(n)	O(1)	O(1)	O(n)	O(1)	O(n)	O(n)

Sorting algorithms

intermediate data structure	one and rest	half and half
array	selection sort: O(n^2)	quicksort: O(n log n)
sorted array	insertion sort: O(n^2)	merge sort: O(n log n)
binary heap	heap sort: O(n log n)	–
order-preserving hash table	bucket sort	–

Graph search algorithms

agenda	algorithm
stack	depth-first search
queue	breath-first search
priority queue	Dijkstra's algorithm

Explanation of Tables

Abstract data types

The first column of this table lists all the abstract data types (ADTs) covered in the class. The second column lists what kinds of *insert* operations that ADT supports; the third column lists what kinds of *delete* operations that ADT supports. For example, a stack supports insertion at the back but not anywhere else. “Insert anywhere” means that the ADT has an insert operation that doesn’t let you specify the location. “Delete by value” means that the ADT has a delete operation that lets you specify which value you want to delete.

The next two tables then describe possible ways of *implementing* these ADTs.

Sequence data structures

The first column lists all the data structures covered in Unit 1. The second column lists the time complexity of getting the value at a particular position in the sequence. The third column, searching for the minimum/maximum value or a particular value anywhere in the sequence. The fourth column, inserting a value at a particular position. Similarly the fifth and sixth columns. The last column lists the time complexity of concatenating two sequences.

Set data structures

The first column lists all the data structures covered in Units 2–4. The second column lists the time complexity of finding the minimum or maximum element in the set. The third column, a particular value in the set. The fourth column, inserting a value (anywhere) into the set. The fifth and sixth columns, deleting the minimum/maximum element or a particular value in the set. The seventh column (“create from list”) lists the time complexity of inserting n elements into the set all at once (which is non-obvious only for binary heaps). The last column lists the time complexity of finding the union of two sets.

The next two tables describe the relationships between certain data structures and algorithms.

Sorting algorithms

This table summarizes the sorting algorithms we learned in Unit 2. The first and second columns show the connection between various priority queue data structures and the sorting algorithm you would get by pushing all the elements into the priority queue and then popping all of them out. The third column is somewhat obscure. It just tries to say that quicksort is related to selection sort and merge sort is related to insertion sort in the following way: if you do quicksort by splitting the array into 1 and n–1 elements, you get selection sort, and if you do merge sort by splitting the array into 1 and n–1 elements, you get insertion sort.

Graph algorithms

This table summarizes the shortest-path algorithms covered in Unit 5.