# Data Structures

David Kauchak
cs302
Spring 2012

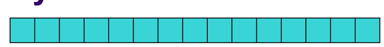
## Data Structures

- What is a data structure?
  - Way of storing data that **facilitates particular operations**
- Dynamic set operations: For a set *S*
  - Search(S,k) – Does *k* exist in *S*?
  - Insert(S,k) – Add *k* to *S*
  - Delete(S,x) – Given a pointer/reference, x, to an elkement, delete it from *S*
  - Min(S) – Return the smallest element of *S*
  - Max(S) – Return the largest element of *S*

## Data structures

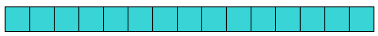
What are some of the data structures you've seen?

## Array



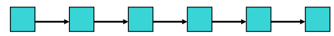
- Sequential locations in memory in linear order
- Elements are accessed via index
- Cost of operations:
  - Search(S,k) – O(n)
  - Insert(S,k) – Θ(1) if we leave extra space, Θ(n)
  - InsertIndex(S,k) – Θ(1)
  - Delete(S,x) – Θ(n)
  - Min(S) – Θ(n)
  - Max(S) – Θ(n)

## Array



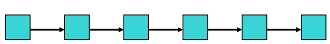
- Uses?
  - constant time access of particular indices

## Linked list



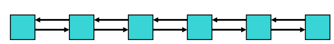
- Elements are arranged linearly.
- An element in the list points to the next element in the list
- Cost of operations:
  - Search(S,k) – O(n)
  - Insert(S,k) – Θ(1)
  - InsertIndex(S,k) – O(n) or Θ(1) if at index
  - Delete(S,x) – O(n)
  - Min(S) – Θ(n)
  - Max(S) – Θ(n)

## Linked list



- Uses?
  - constant time insertion at the cost of linear time access

## Double linked list

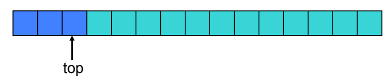


- Elements are arranged linearly.
- An element in list points to the next element **and** previous element in the list
- What does the back link get us?
- Θ(1) deletion (assuming a reference to the item)

## Stack

- LIFO
- Picture the stack of plates at a buffet
- Can implement with an array or a linked list

## Stack




- LIFO
- Picture the stack of plates at a buffet
- Can implement with an array or a linked list

push(1)

push(2)

push(3)

pop() 3

pop() 2

pop() 1

## Stack

- Empty – check if stack is empty
  - Array: check if "top" is at index 0
  - Linked list: check if "head" pointer is null
  - Runtime: Θ(1)

## Stack

- Pop – removes the top element from the list
  - check if empty, if so, "underflow"
  - Array:
    - return element at "top" and decrement "top"
  - Linked list:
    - return and remove at front of linked list
  - Runtime:
    - Θ(1)

## Stack

- Push – add an element to the list
  - Array:
    - increment "top" and insert element. Must check for overflow!
  - Linked list:
    - insert element at front of linked list
  - Runtime:
    - $\Theta(1)$

## Stack

- Array or linked list?
  - Array: more memory efficient
  - Linked list: don't have to worry about "overflow"
  - Other options?
    - ArrayList (expandable array): compromise between two, but not all operations are O(1)
- Uses?
  - runtime "stack"
  - graph search algorithms (depth first search)
  - syntactic parsing (i.e. compilers)

## Queue

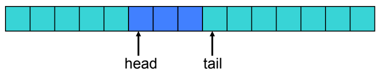
- FIFO
- Picture a line at the grocery store

Enqueue(1)
Enqueue(2)
Enqueue(3)
Dequeue() 1
Dequeue() 2
Dequeue() 3

## Queue

- Can implement with:
  - array?
  - singly linked list?
  - doubly linked list?

## Queue




- FIFO
- Can implement with an array, a linked list or a double linked list
- Array:
  - keep head an tail indices
  - add to one and remove form the other
- Linked list
  - keep a head and tail reference
  - add to the tail
  - remove from the head
- Runtimes?

## Queue

- Operations
  - Empty – Θ(1)
  - Enqueue – add element to end of queue - Θ(1)
  - Dequeue – remove element from the front of the queue - Θ(1)
- Uses?
  - scheduling
  - graph traversal (breadth first search)