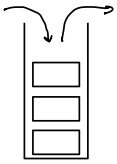
Stack Abstract Data Types

- Mathematical description of an object and a set of operations on the object
  - Dictionary ADT: stores pairs of strings (word & definition)
  - Operations:
    - Insert(word, definition)
    - Remove(word)
    - Lookup(word)

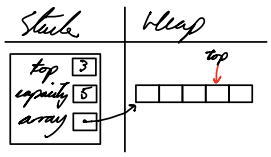


Insert → "Fest" "2 of 'em"

Find("Z125 Pro") → "Z125 Pro" "Fun in the Sun!"

- Sage 9 LIC
  - Smell like a lawnmower
- Z125 Pro
  - Fun in the Sun!
- CB300F
  - For the commuter

- Stack Implementation
  - Track size of array, top of array, and array itself
  - stack.h includes multiple array manipulating functions



ADT Application — Postfix Notation (RPN)

- Every operator follows its operands

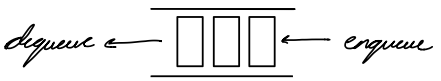
Ex:

infix: $5+((1+2)\times 4)-3$

RPN: $512+4\times+3-$

Queue ADT

- Items enter from the back and are retrieved from the front



Recursion

- A Function calling itself $[T(n) \in O(n)]$

Ex: Fibonacci Series

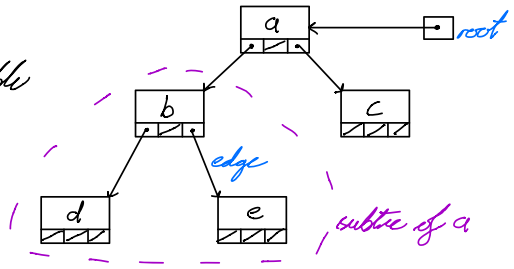
N^{th} number of Fibonacci series = $fib(n) = fib(n-1) + fib(n-2)$

- Many versions can be called in stack depending on distance of n from base case
- Tail Recursion occurs when recursive call is last operation in function

Trees

- Node with pointers to multiple other nodes
- One edge to each node
- Height is number of levels one node is above another
- Depth is number of levels one node is below another

- Balanced if leaves are all approximately same distance from root
- Use recursion to traverse down each possible path



Binary Search Tree (2 branches per node)

- For all nodes in the tree:
  - All nodes in a left subtree have labels less than the root
  - All nodes in a right subtree have labels greater than the root
- Looking for certain value in tree: Worst case = height of tree + 1
- Minimum value in tree
  - From root follow left link until NULL
- Maximum value in tree
  - From root follow right link until NULL

- Insertion
  - Search tree for node where left child is smaller, and right child is greater than target
- Removal
  1) No children → assign NULL to parent pointer
  2) One child → replace node with child
  3) Two children → replace node with node's successor or predecessor
     - The rightmost node of the left subtree is the predecessor
       The leftmost node of the right subtree is the successor

- Efficiency

	Insertion (Worst Case)	Insertion (Average Case)	Removal (Worst Case)	Removal (Average Case)	Search (Worst Case)	Search (Average Case)
Unordered Doubly-linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Ordered array	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(\log n)$
BST	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$

- Priority Queue
  - Organized collection of data to allow fast access to, and removal of, largest or smallest element (highest or lowest priority)

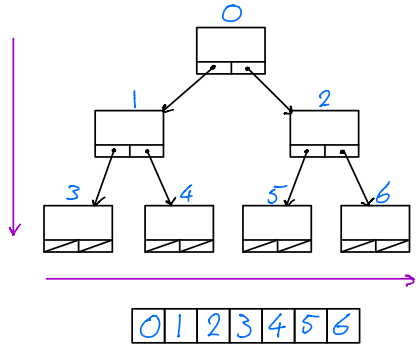
Heap

- The Heap is a Binary Tree that is:
  - Complete: all levels are filled except bottom
  - Partially Ordered: Max heap: value of node is at least value of children
    Min heap: value of node is no greater than value of children
- Can be implemented using arrays
  - Nodes are indexed from top to bottom and left to right

- Array is indexed from 0 to $n-1$
- Each level's nodes are indexed from:

$$2^{level} - 1 \text{ to } 2^{level+1} - 2$$

- Children of node i are:
  Left: $2i+1$ Right: $2i+2$
- Parent of node i is: $\frac{(i-1)}{2}$



- Insertion
  - "Complete" property is followed first, then "Partially Ordered" is fixed
- Removal
  - Replace value with last value in array/furthest right leaf
- Construction
  - Turning an unordered array into a heap; swap node α with largest child for every node on: $0 \leq \alpha \leq \frac{n-1}{2}$ (length of array)
  - Complexity of $O(n)$
- Sorting a Max Heap into an Ascending Array
  - Repeatedly remove root and place at end of array/first unused leaf

Sorting

- Stability: A stable sorting algorithm maintains relative order of records with equal keys
- Selection Sort: Organizes array into ascending order by swapping each unsorted value with smallest unsorted value
  - $\frac{1}{2}n(n-1)$ comparisons
- Insertion Sort: Locate correct position in new array to place value, shift following values in new array by 1
  - $\frac{1}{2}n(n-1)$ comparisons
- Merge Sort: Sort each half of array recursively, then merge halves
  - $n-1$ comparisons
- Quick Sort: Divide array into small and large value sections (partitions), then sort each section
  - $\frac{1}{2}n(n-1)$ comparisons
  - runs better than merge sort ($O(n \log n)$ vs $O(n^2)$)

Hash Tables

- Maps key of data into a hash table (array)'s index

- Deterministic — Use array index prescribed by hash function to check if index is taken
- Uniformity — "Hash" function should use all parts of array
  - "Similar" keys should be mapped to random array indices
- A Collision occurs when two different keys are mapped to same index
  - Open Addressing — look for empty array element following location of collision (probe)
    - Linear Probing — place element at $h(x)+1, h(x)+2, \ldots$
      - can result in clusters of elements
    - Quadratic Probing — place element at $h(x)+1^2, h(x)+2^2$
      - can result in repeating pattern of probes
    - Double Hashing — secondary hash function produces probe sequence