COMPETITIVE PROGRAMMING CLUB

Membership sign up:

Week 3

JUNIOR EXEC APPLICATIONS OPEN

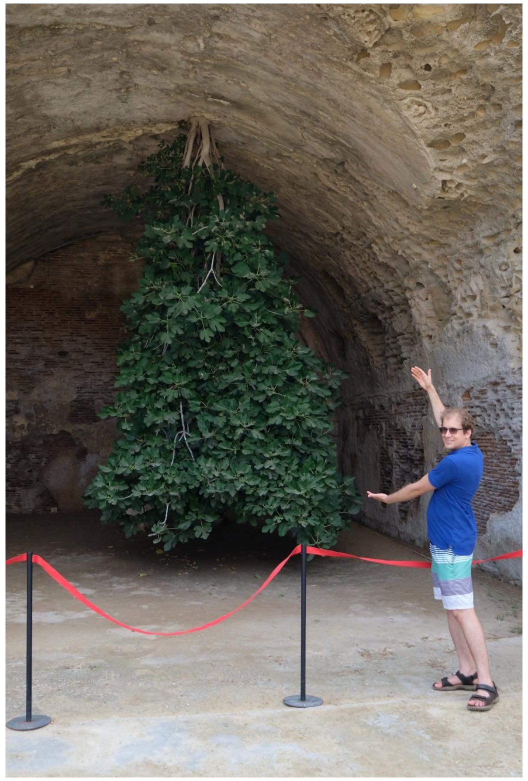
Go to link below or scan QR code to apply.
The deadline is OCTOBER 31st at midnight.

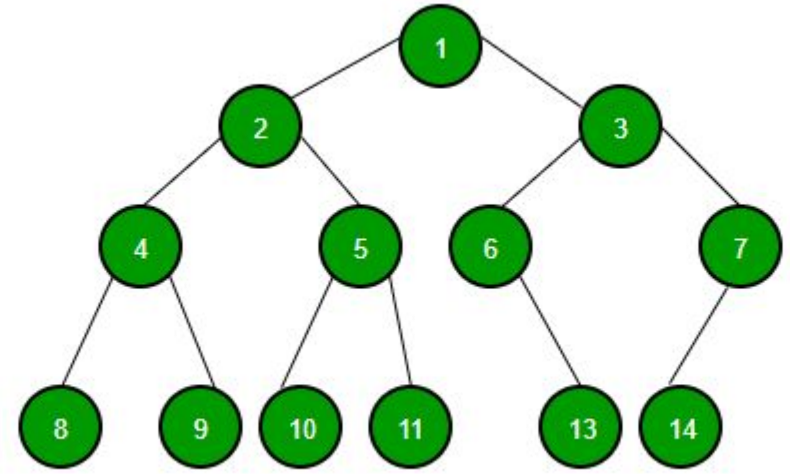
https://forms.gle/8dkGw1Zezfm4BVnRA

SCAN ME

Binary Trees

- Trees in computer science grow from the roots at the top to the leaves in the bottom
    - Just like trees in real life!
- In a binary tree, each internal node branches into (at most) two children
- Can usually represent them internally as a simple list or array
    - Reach left child by doubling index, right child by doubling + 1, and parent by halving index

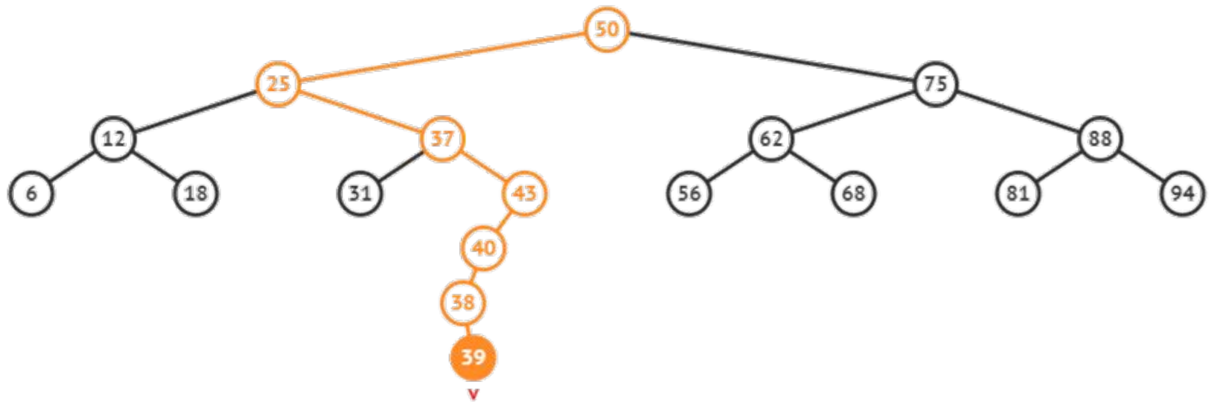




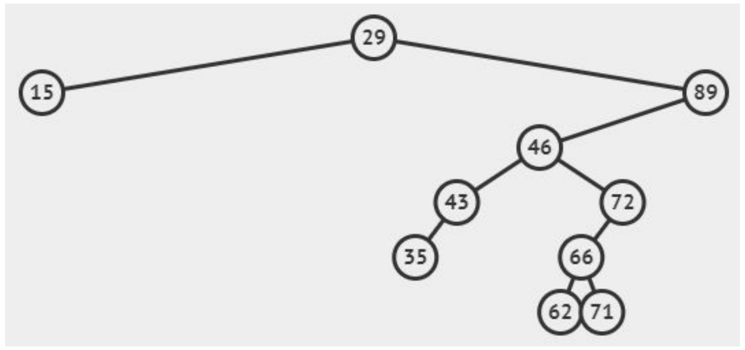
geeksforgeeks.org/binary-tree-data-structure

Binary Search / Divide & Conquer

- We can take the idea of binary trees to solve a problem by repeatedly splitting it in half
- Keep track of a lower and upper bound, and keep checking the midpoint until finding the right number
- Can exponentially reduce the search time for problems if you can tell if a guess is too low or too high!
  - Specifically, O(logN) comparisons for a search space of size N
- To binary search a **sorted array**, use the built-in bisect module in Python or lower_bound / upper_bound in C++

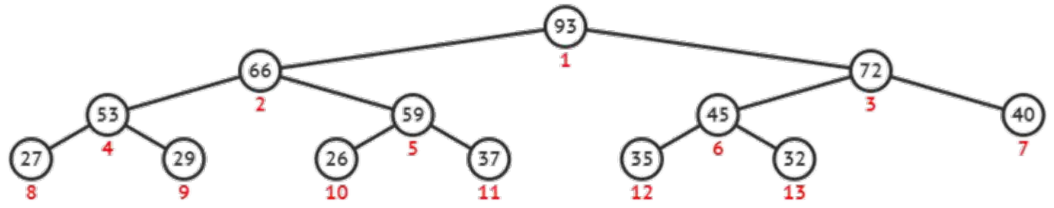


Binary Search Trees



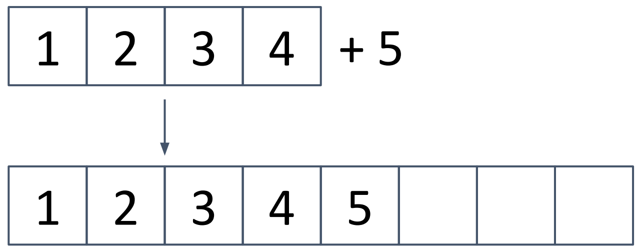
- https://visualgo.net/en/bst
- Keep a set of items in order, and allow extracting either the min or the max in O(logN) time (insertion also takes O(logN))
- set<int> in C++ (or map<int, int> if used as ordered key-value pairs)
  - Again, can replace int with any comparable datatype
- Not built into Python!
- Require rebalancing as elements get added or removed
  - C++ does this automatically with set / map
  - If we often only need the min (or max), consider the somewhat more convenient heaps / priority queues

Application: binary heaps → priority queues



- https://visualgo.net/en/heap
- Keep the max (/min) element at the top, and ensure every node is larger (/smaller) than its children
- Good if you want to **insert arbitrary elements** into a container and **efficiently extract the max (/min) element**
  - Both push (insert) and pop (extract) operations take O(logN) time, where N is the size of the heap / priority queue
- `priority_queue<int>` in C++ (max heap)
  - Can replace `int` with any other comparable data type
- `from heapq import heappush, heappop` in Python (min heap)
  - These operations work on any list of comparable data
- `java.util.PriorityQueue<Integer>`

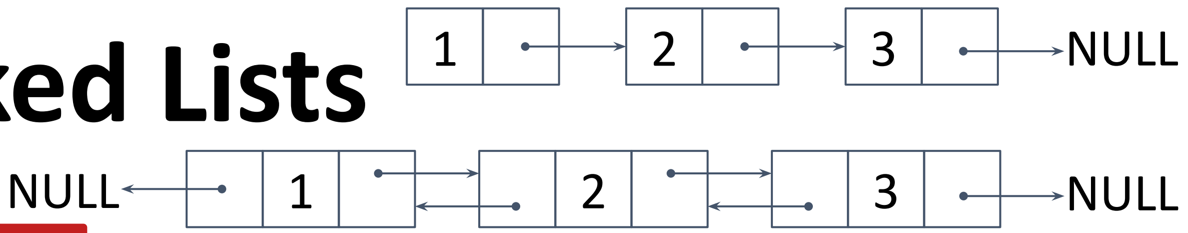
Array Lists



A List implementation based on storing an "internal" static array.
When attempting to add an element that cannot fit in the internal array a new array is created with twice the length and the original contents are copied over.
Operations:

- **insert_back(new_element) ∈ ~O(1):** While in worst case we could have to copy all n elements in the internal array, this is amortized by the fact that the first n operations each took constant time thus each operation can be thought of as being in O(2) which is the same as O(1)
- **insert_front(new_element) ∈ O(n):** We will have to shift all n values in the array over to the right
- **insert(index, new_element) ∈ O(n):** unlike the insert_back we cannot amortize this operation so we will have to shift all (n - index) elements right of the insertion point one space to the right thus O(n)
- **remove_back() ∈ O(1):** Removal from the end requires no shifts so we can do it in O(1)
- **remove_front() ∈ O(n):** Removal from the front requires us to shift all n - 1 values to the left
- **remove(index) ∈ O(n)**: Similar to insert we cannot amortize this operation and will have to shift all (n - index) elements right of the removal point one space to the left
- **get(index) ∈ O(1):** Querying an array by index is in O(1)

Linked Lists/Doubly Linked Lists



A List implementation based on storing nodes each containing a value, a pointer to the next node, and in the case of doubly linked list a pointer to the previous node. Both should store an additional pointer to the first (head) and last (tail) nodes

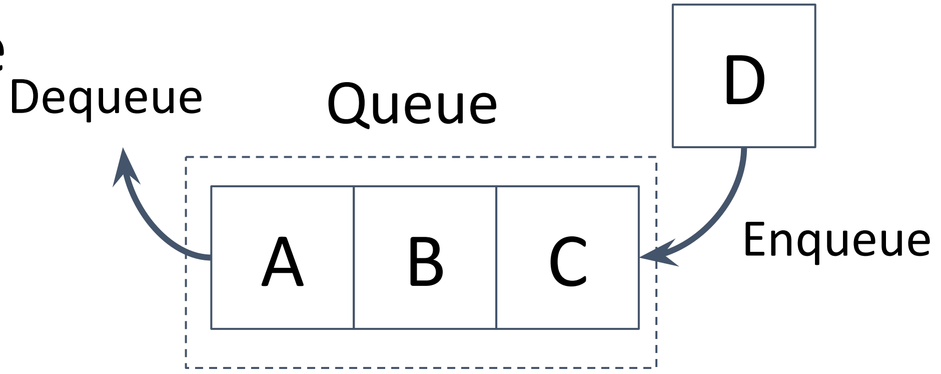
Operations:

- **insert_back(new_element) ∈ O(1):** This is easily accomplished by adding a new node and setting the pointer of the tail node to the new node (and the prev pointer of the new node to the old tail)
- **insert_front(new_element) ∈ O(1):** This is easily accomplished by adding a new node and setting its pointer to the old head node (and the prev pointer of the old head to the new node)
- **insert(index, new_element) ∈ O(index):** We will have to traverse the list node by node until the index is reached then we create a new node and adjust the pointers accordingly
- **remove_back() ∈ O(n)** for singly linked lists or **O(1)** for doubly linked lists**:** For singly linked lists we have to traverse all the way to the second last node and set its pointer to null. For doubly linked lists we can jump right to the end using the tail pointer then traverse backwards once.
- **remove_front() ∈ O(1)** We simply set the head node to be the node following the old head
- **remove(index) ∈ O(index)**: pretty much the same as insert but instead we modify the pointers to skip over the current node.
- **get(index) ∈ O(index):** To query an element by index we will have to traverse the list node by node until the index is reached when we return the value of the current node

(Note: for removal operations in languages without a garbage collector we should also free the node)

Queues

- A queue is a data structure that follows the **First-In-First-Out** principle (FIFO).
- Behaves just like a line aka queue where the first person to arrive gets treated first
- The basic operations for a queue are **enqueue, dequeue,** and **peek.** All are O(1).
- Peeking allows you to look at the first item without removing it from the queue
- Can be implemented easily using a Singly or Doubly linked list (an arraylist can be used but is more complicated)



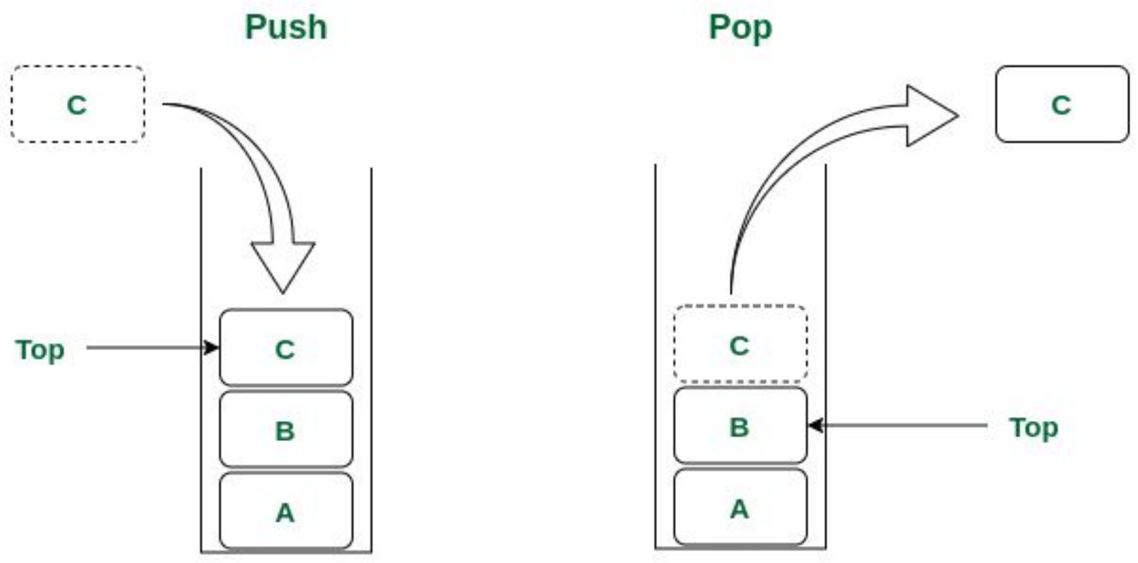
Queues Applications

- **Breadth-First Searches** - for traversing graphs and trees
- **Parallelization** - used to give threads/processes control in turn
- Any situation where you have values that need to be processed in a particular order

Stacks

- A stack is a data structure that follows the **Last-In-First-Out** principle (LIFO).
- Just like a stack of books, the book you put on top will be the first book you'll take off when you want to access them.
- The basic operations for a stack are **push, pop,** and **peek.** All are O(1).
- Peeking allows you to look at the top item without removing it from the stack

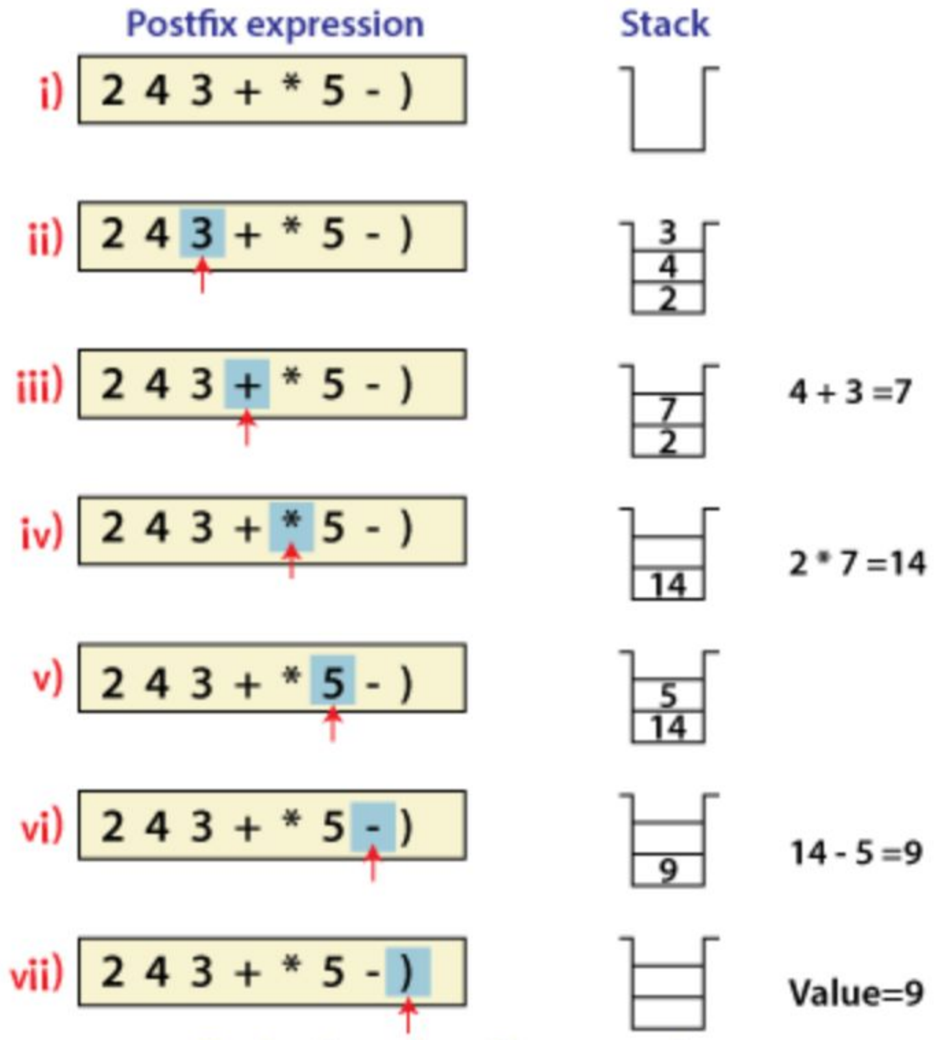
- Can be implemented easily by an ArrayList, Singly Linked List, or Doubly Linked List



Stack Data Structure

Stacks - Advanced Applications

- **Depth-First Searches** - for traversing graphs and trees.
- **Backtracking** - in problems when you need to explore multiple possibilities or find a solution among various options.
- **Memory management** - to allocate or deallocate memory efficiently.
- **Expression evaluation** - for efficiently evaluating expressions in postfix notation, or converting expressions from infix to postfix.

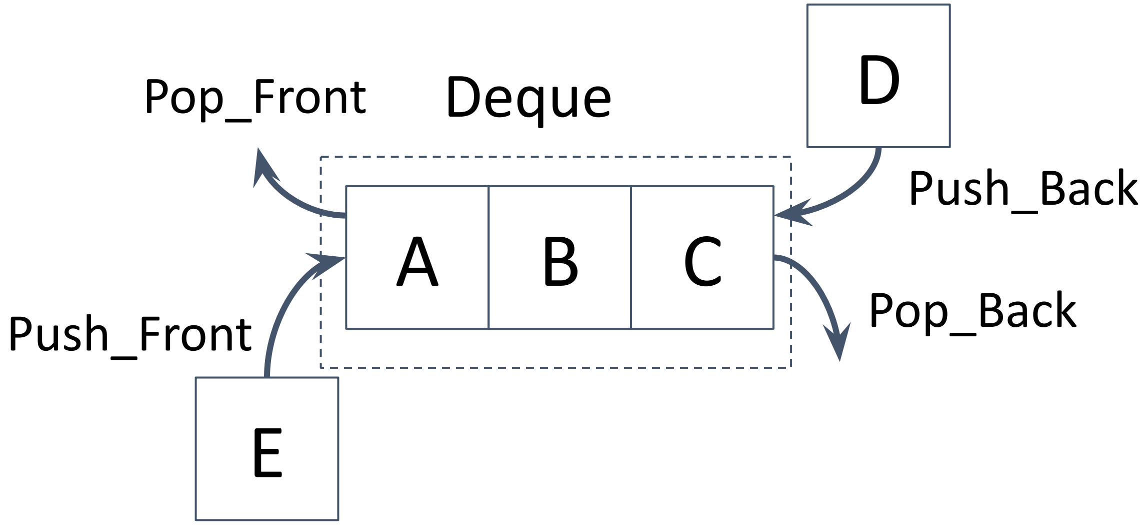


Evaluation of postfix expression

Double Ended Queues (AKA Deques)

- A deque is a data structure that behaves both as a queue and as a stack (kind of)
- Elements can be added at the front **or** back and can be removed from either as well
- The basic operations for a deque are **push_front, push_back, pop_front, pop_back, peek_front,** and **peek_back.** All in O(1)

- Can be implemented easily using a Doubly Linked List



Today’s Contest

https://open.kattis.com/contests/syyayc

(or look up “CPC Fall 2023 Practice Contest Week 3” in the Kattis contest list)

Feel free to ask questions until 7pm, and then throughout the week on Discord!

COMPETITIVE
PROGRAMMING
CLUB