Special types of binary trees

Binary search trees and Heaps
Binary Search Trees (BSTs)

- Write the **inorder** traversal of this tree.

  What do you observe?

- A **search tree** is a tree whose elements are organized to facilitate finding a particular element.

- A **binary search tree** is a binary tree that, for each node \( n \):
  - the left subtree of \( n \) contains elements less than the element stored in \( n \)
  - the right subtree of \( n \) contains elements greater than or equal to the element stored in \( n \)

- Binary search trees must hold **Comparable data**. Why?

- How do you search for an element?
Adding an Element to a BST

Next to add:

Next to add:

Animation:
https://www.cs.usfca.edu/~galles/visualization/BST.html

On what does the shape of a binary search tree depends?
**Degenerate Tree**

- A grossly unbalanced tree, with some long paths

- When does it occur?

- Why is it undesirable?
Removing an Element from a BST

- Removing a target in a BST is not as simple as that for linear data structures
- After removing the element, the resulting tree must still be valid
- What if you remove 88? 51? 60? 86?
After the Root Node is Removed

- Draw Tree (2 valid configurations)
**javafoundations.BinarySearchTree**

```java
package javafoundations;

public interface BinarySearchTree<T extends Comparable<T>> extends BinaryTree<T>{
    // Adds the specified element to the tree.
    public void add(T element);

    // Finds and returns the element in the tree matching the specified target. Overrides the find method of BinaryTree.
    public T find(T target);

    // Returns the minimum value in the binary search tree.
    public T findMin();

    // Returns the maximum value in the binary search tree.
    public T findMax();

    // Removes and returns the specified element from the tree.
    public T remove(T target);
}
```
Let’s take a closer look at the implementation
Heaps and Priority queues
Priority “Queue”
Priority queues, which data structure should we use?

- It’d be great if we can keep the elements sorted.
- Array?
- Linked list?
- Binary Tree? Binary Search Tree?
But then... what kind of tree is this?

- A **minheap** is a
  - complete binary tree
  - each element is *less* than or equal to both of its children

- A minheap keeps the smallest valued element readily available
Heaps, which data structure should we use?
Finding child nodes

Children of H[2] are at H[?] and H[?]
Finding parent nodes

Parent of H[9] is at H[?]
Adding a new element
Heapify-up(H, 15)
Heapify-up(H, 7)
Heapify-up(H, 3)
Algorithm 2.8, page 61

Heapify-up(H,i):
    If $i > 1$ then
        let $j = \text{parent}(i) = \lfloor i/2 \rfloor$
        If $\text{key}[H[i]] < \text{key}[H[j]]$ then
            swap the array entries $H[i]$ and $H[j]$
            Heapify-up(H,j)
        Endif
    Endif
Endif

Why does this work?
Deleting an element
Deleting an element
Heapify-down(H, 3)
Heapify-down(H, 7)
Algorithm 2.9, page 63

Heapify-down(H,i):
Let n = length(H)
If 2i > n then
   Terminate with H unchanged
Else if 2i < n then
   Let left = 2i, and right = 2i + 1
   Let j be the index that minimizes key[H[left]] and key[H[right]]
Else if 2i = n then
   Let j = 2i
Endif
If key[H[j]] < key[H[i]] then
   swap the array entries H[i] and H[j]
   Heapify-down(H,j)
Endif