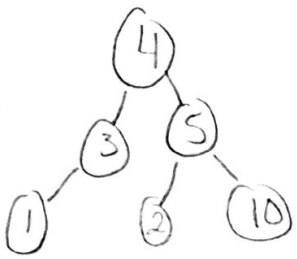
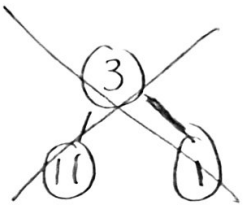
Binary Search Tree Thought process

- Each node has at most two children
- Right child is greater than, where the left child is less than

Ex)





A poorly structured tree

To be safe sort the Array before searching

* However insertion of nodes matter

Eg) 1, 2, 3, 4, 5

Unbalanced tree

(Somewhat) Balanced Tree

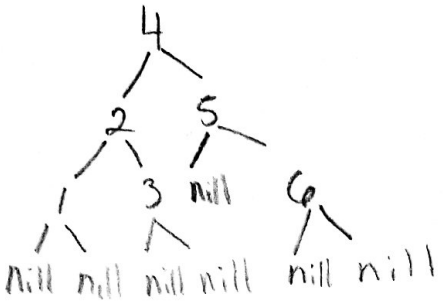
this what we want to achieve

Work flow

Numpy to generate Random Array → Reuse Heap Sort Algorithm → (1) Create Tree → (2) Populate tree with previous constraints in mind → Binary Search Tree Algorithm

How do I know when to stop recursing the function?

Answer:



* When both children return nill, which is a result of being out of index

Input has inconsistent length?

* This functions index check allows us to have varying lengths

Pseudocode

Array To Tree (arr, first, last)

Assumes array is sorted

middle = (first - last)/2 + first

left = Array To Tree (arr, first, middle)

right = Array To Tree (arr, middle, last)

if not Nill

add to tree

if left and Right == nill and with in index

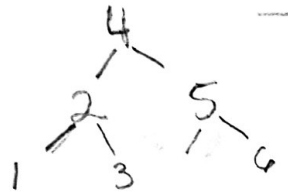
Return

Array to binary Tree

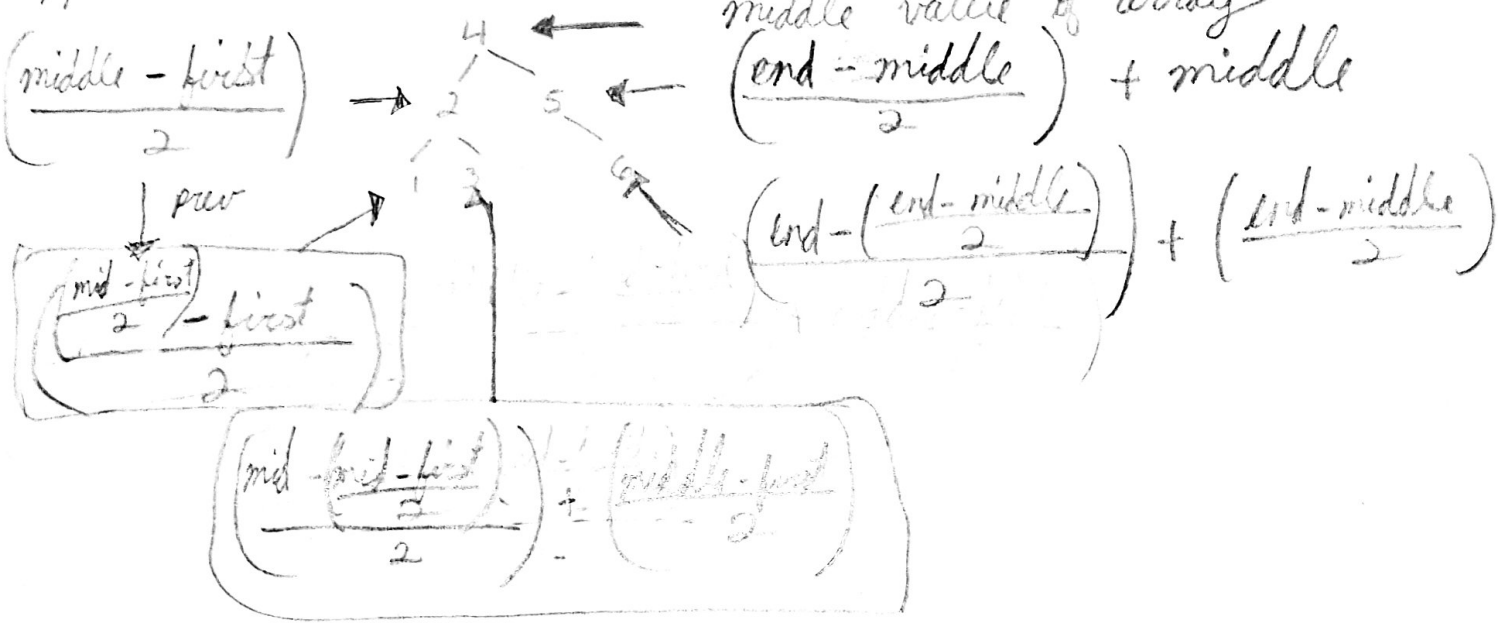
Input: Sorted Array

1	2	3	4	5	6
0	1	2	3	4	5

Output: Balanced Tree



Approach



Concerns:

What if the length of my input changes often?

How do I know when to stop repeating the function

Binary Search

Problem: Given an sorted array. We want to implement binary search algorithm to find index of element in array, if present.

Constraint(s):

① Array must be sorted in Ascending order

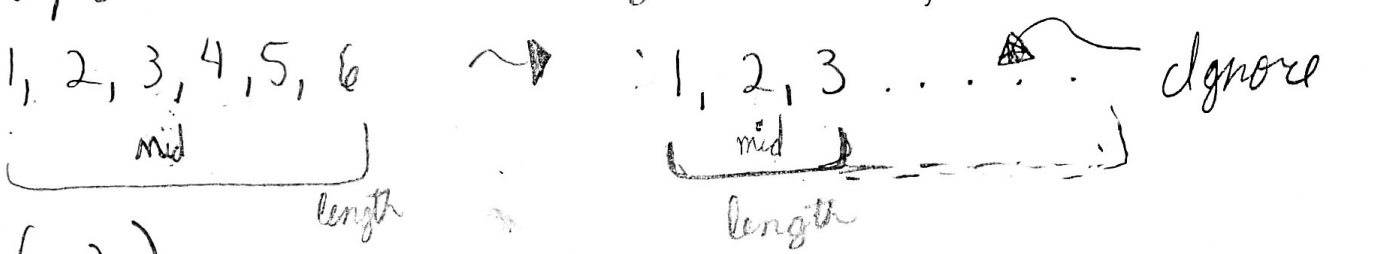
1, 2, 3, 4, 5, 6

Ex Query...

input = 1

(3)

array[mid] > 1; So move to the left



(2)

array[(new) mid] > 1; So move left

If array[(new new) mid] = 1

return index

Else return Not in True... or Nil.