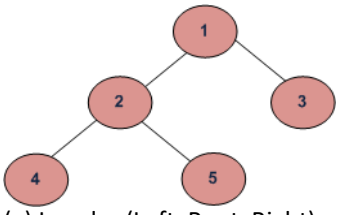
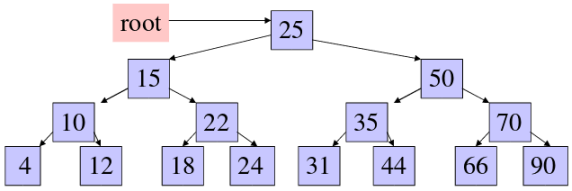


(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25




```
//sorts O(n²) time
//smallest item goes to top of S
def SortStack(S):
    T = Stack()
    sorted = false
    while not sorted:
        sorted = true
        top = S.pop()
        if top < S.peek()
            T.push(top)
            top = S.pop()
        else
            T.push(S.pop())
            sorted = false
        S.push(top)
        while not T.empty()
            S.push(T.pop())
```

i. CLAIM: $f(n) = 4n^3 - 8n^2 - 36n + 72$ is $O(n^3)$.
Consider $c = 1$ and $k = 5$.
Suppose we pick any $n \geq k$. Then $2n^2 + 9n = n(2n + 9) \geq 95$. So $8n^2 + 26n \geq 95 > 72$. So $-8n^2 - 36n + 72$ is negative. This means that

$$f(n) = 4n^3 + (-8n^2 - 36n + 72) < 4n^3 = cn^3$$

So $f(n)$ is $O(n^3)$.

ii. CLAIM: $f(n) = 4n^3 - 8n^2 - 36n + 72$ is $\Omega(n^3)$. That is n^3 is $O(4n^3 - 8n^2 - 36n + 72)$.
Consider $c = 1$ and $k = 7$.
Suppose we pick any $n \geq k$. Then $(n-2)^2 > 22$. So $n^2 - 4n - 18 = (n-2)^2 - 22$ is positive. So $2n^3 - 8n^2 - 36n$ is positive. So $f(n) \geq 2n^3 \geq n^3$.
So n^3 is $O(f(n))$.

Because we have shown that $f(n)$ is $O(n^3)$ and $f(n)$ is $\Omega(n^3)$, it must be that $f(n)$ is $\Theta(n^3)$. QED.

	InsertionSort	SelectionSort	MergeSort	Quicksort
Best	$O(n)$	$O(n^2)$	$O(n\log n)$	$O(n\log n)$
Worst	$O(n^2)$	$O(n^2)$	$O(n\log n)$	$O(n^2)$
Average	$O(n^2)$	$O(n^2)$	$O(n\log n)$	$O(n\log n)$
Stable	Y	Y	Y	N
In-Place	Y	Y	N	Y

```
def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

Bubble Sort Worst: $O(n^2)$
Average: $O(n^2)$
Best: $O(n)$

The best (fastest avg) **stable** sort is Merge Sort
Other stable sorts are Selection, Bubble, and Insertion. Quick sort is **not** stable.

Worst case number of items checked in sorted list of 64 elements is 7 (64, 32, 16, 8, 4, 2, 1) Unsorted would be 64.

$$\frac{n(n+1)}{2} \in \Theta(n^2)$$

2^{n-1} has the same order of growth as 2^n

```
For I = 1 to n²
     For j=1 to n
          Sum++
```

Has runtime $O(n^3)$

Height is number of branches from root to leaf
Ex: top left tree has height of 2

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

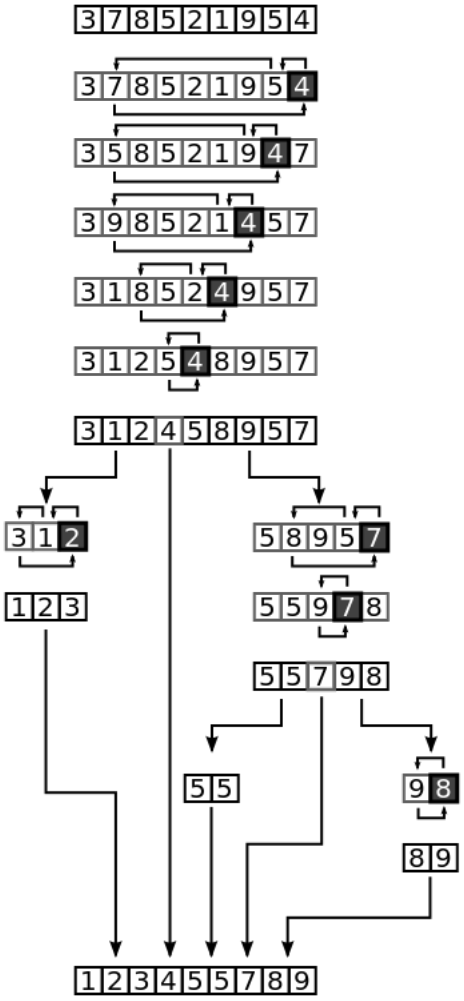
Runtime of Insertion Sort when elements are initially in descending order is $O(n^2)$

BST insertion: put element in correct place, split in half, then move stuff. (first of second half goes to root node contents)

```
//CRC – Count Right Children
Def CRC(T):
	If T is none:
		Return 0
	Elif T.right is not none
		Return 1 + CRC(T.Right)
	Else
		Return CRC(T.left) + CRC(T.right)
```

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1 )
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo - 1
    for j := lo to hi - 1 do
        if A[j] < pivot then
            i := i + 1
            swap A[i] with A[j]
    swap A[i + 1] with A[hi]
    return i + 1
```



Full example of quicksort on a random set of numbers. The shaded element is the pivot. It is always chosen as the last element of the partition. However, always choosing the last element in the partition as the pivot in this way results in poor performance ($O(n^2)$) on *already sorted* arrays, or arrays of identical elements. Since sub-arrays of sorted / identical elements crop up a lot towards the end of a sorting procedure on a large set, versions of the quicksort algorithm that choose the pivot as the middle element run much more quickly than the algorithm described in this diagram on large sets of numbers.