

(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

­­­­



//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

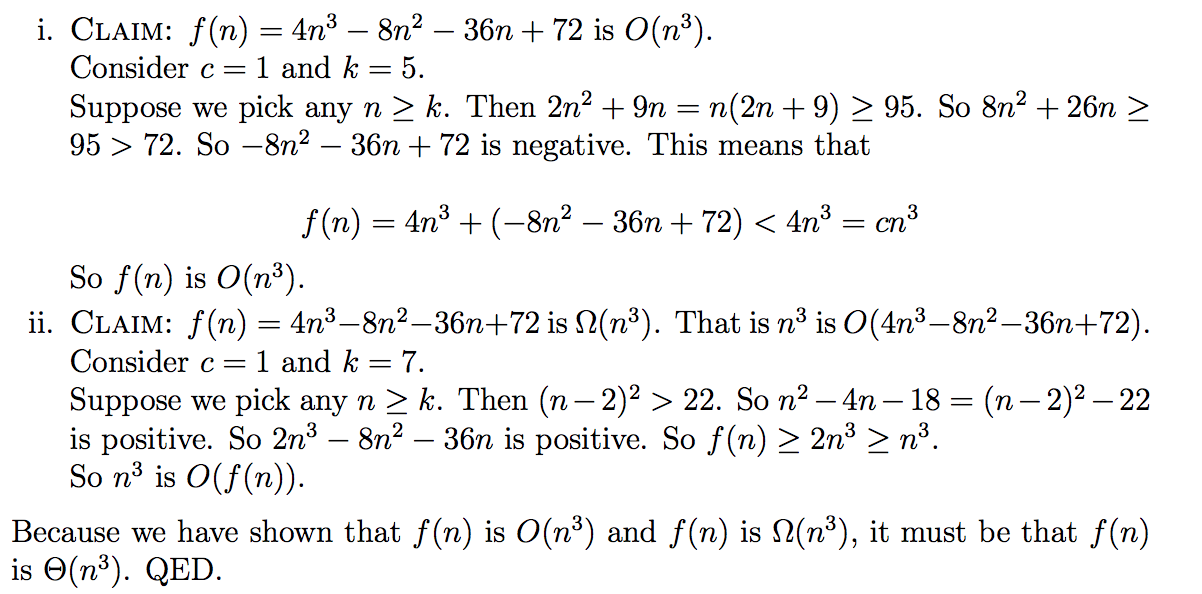
T.push(S.pop())

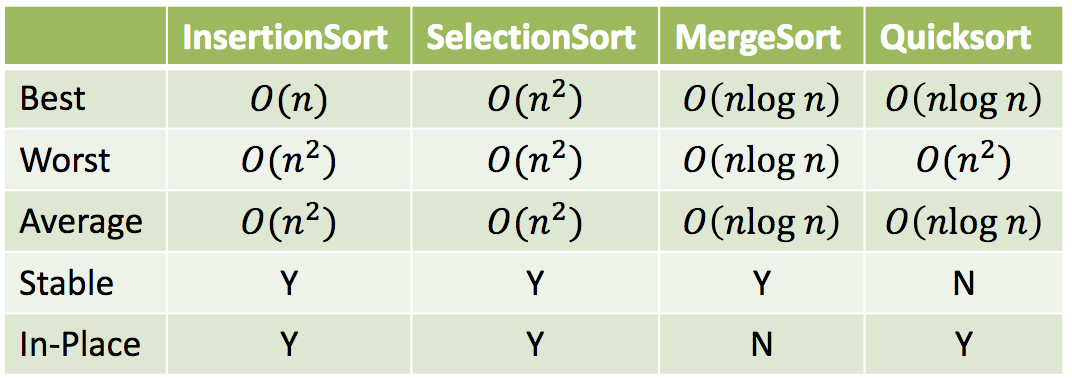
sorted = false

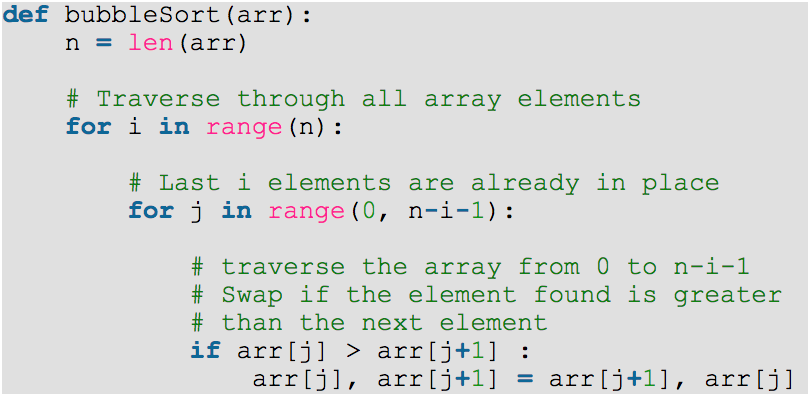
S.push(top)

while not T.empty()

S.push(T.pop())







Bubble Sort Worst: O(n2)

Average: O(n2)

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.

2n-1 has the same order of growth as 2n

For I = 1 to n2

For j=1 to n

Sum++

Has runtime O(n3)

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(n2)

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:

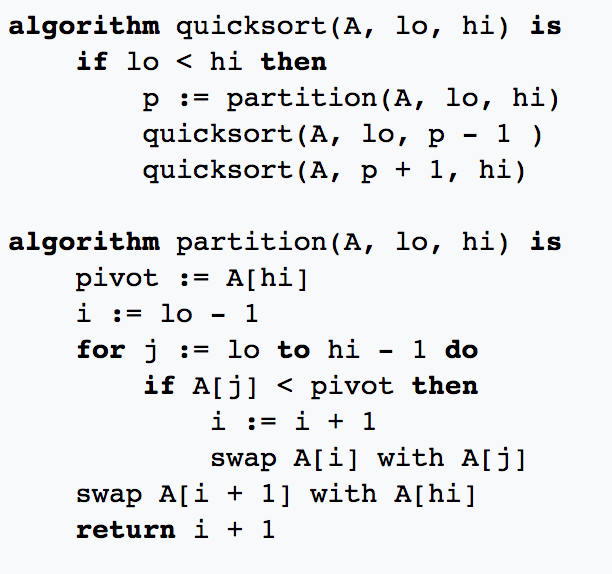
Return 0

Elif T.right is not none

Return 1 + CRC(T.Right)

Else

Return CRC(T.left) + CRC(T.right)



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.



