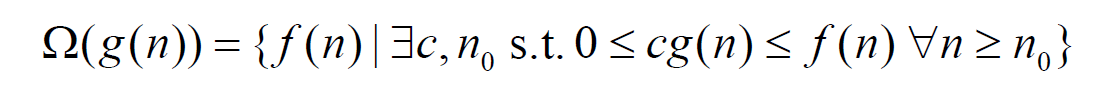
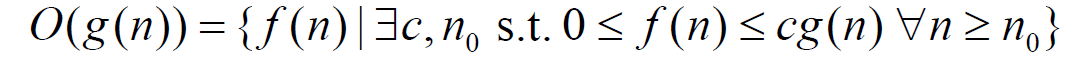
1. Find the median of an unsorted array of n elements with the time O (n) in average case. Divide input into group of size 5: O(n) => Find the medians of each group: O(n) => Find the median of all pivots as medians found: n/5 => Complexity O(n).
2. Divide and Conquer vs. Dynamic Programming: Both techniques split their input into parts, find subsolutions to the parts, and synthesize larger solutions from smaller ones. Divide and Conquer splits its input at prespecified deterministic points (e.g., always in the middle). Dynamic Programming splits its input at every possible split points rather than at pre-specified points. After trying all split points, it determines which split point is optimal.
3. Greedy vs. Dynamic Programming:

* Both techniques are optimization techniques, and both build solutions from a collection of choices of individual elements. The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices. Dynamic programming computes its solution bottom up by synthesizing them from smaller subsolutions, and by trying many possibilities and choices before it arrives at the optimal set of choices. There is no a priori litmus test by which one can tell if the Greedy method will lead to an optimal solution. By contrast, there is a litmus test for Dynamic Programming, called [The Principle of Optimality](http://www.seas.gwu.edu/~ayoussef/cs212/dynamicprog.html#poo)

1. Longest common subsequence (LCS) problem: The longest common subsequence (LCS) problem is to find the Upper bound, lower bound, supremum

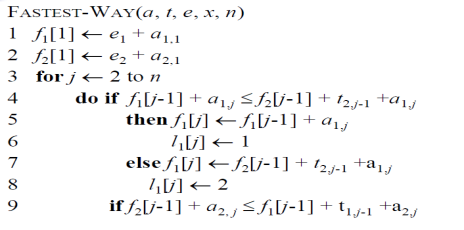




Supremum is least upper bound.

Simple cycle is a closed walk with no repetitions of vertices and edges allowed, other than the repetition of the starting and ending vertex.

1. Heap and binary Tree: a binary heap guarantees that the nodes on higher level are larger than the nodes on lower level. Binary Tree guarantees that the nodes on right are larger than the nodes on left.
2. A randomized algorithm is an [algorithm](http://en.wikipedia.org/wiki/Algorithm) which employs a degree of [randomness](http://en.wikipedia.org/wiki/Randomness) as part of its logic. The algorithm typically uses [uniformly random](http://en.wikipedia.org/wiki/Uniform_distribution_(discrete)) bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits.  Quicksort is a divide and conquer algorithm. DMP: Characterize the structure of an optimal solution. Recursively define the value of an optimal solution. Compute the value of an optimal solution in a bottom up fashion. Construct an optimal solution from computed information.



1. The master method

Let a >= 1and b > 1 be constants, let f (n) be a function, and T(n) be defined on the nonnegative integers by the recurrence T(n) = aT(n / b) + f (n ) where we interpret n / b mean either or

+ If for some constant ɛ > 0, then

+ If then

+ If for some constant ɛ > 0 and if for some constant c < 1 and all sufficiently large n, then .

1. Randomized select method

RANDOMIZED\_SELECT(A,p,r,i)

if p = r

then return A[p]

q <- RANDOMIZED\_ PARTITION( A, p,r )

k = q - p +1

if i = k ► the pivot value is the answer

then return A[q]

elseif i < k

then return RANDOMIZED\_SELECT(A,p,q-1,i)

else return RANDOMIZED\_SELECT(A,q+1,r,i-k)

1. Find both the minimum and the maximum (If n is odd, then we perform 3⌊n/2⌋ comparisons. If n is even, we perform 1 initial comparison followed by 3(n - 2)/2 comparisons, for a total of 3n/2 – 2. Thus, the total number of comparisons is at most 3 ⌊n/2⌋.

n=len(a)

#if odd mark 1st element as min and max

if n%2==1: min=a[0] max=a[0] current=1

#if even length 1st find the min and max of 1st 2 elements and mark it.

if n%2==0 and n>0: if a[0] <= a[1]: min=a[0] max=a[1] else: max=a[0] min=a[1] current=2

for i in range(current,n,2):

#compare two adjacent elements to find min and max

if a[current]<=a[current+1]: minindex=current maxindex=current+1

else: minindex=current+1 maxindex=current

#update min if a[minindex]<min: min=a[minindex]

#update max if a[maxindex]>max: max=a[maxindex]

print min print max