Egg-dropping problem:

Sorting problem:

loop invariant (insertion sort):

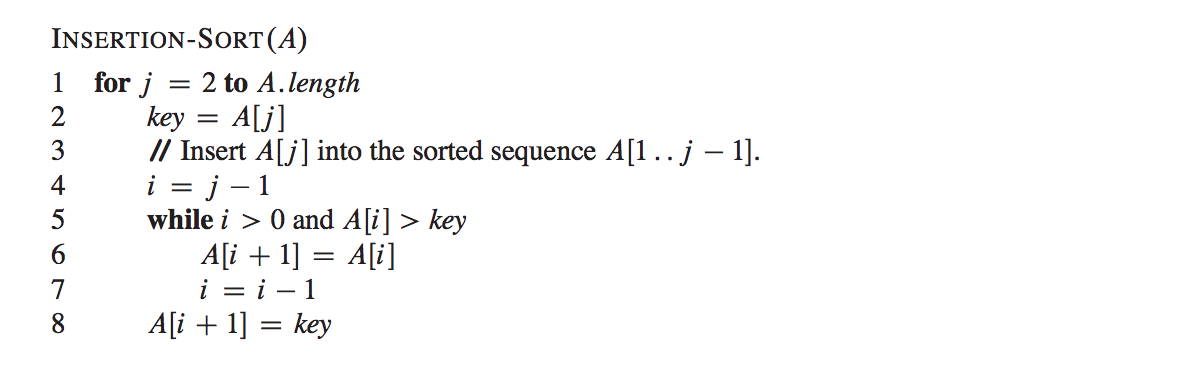
We state these properties of A [j – 1] formally as a loop invariant:

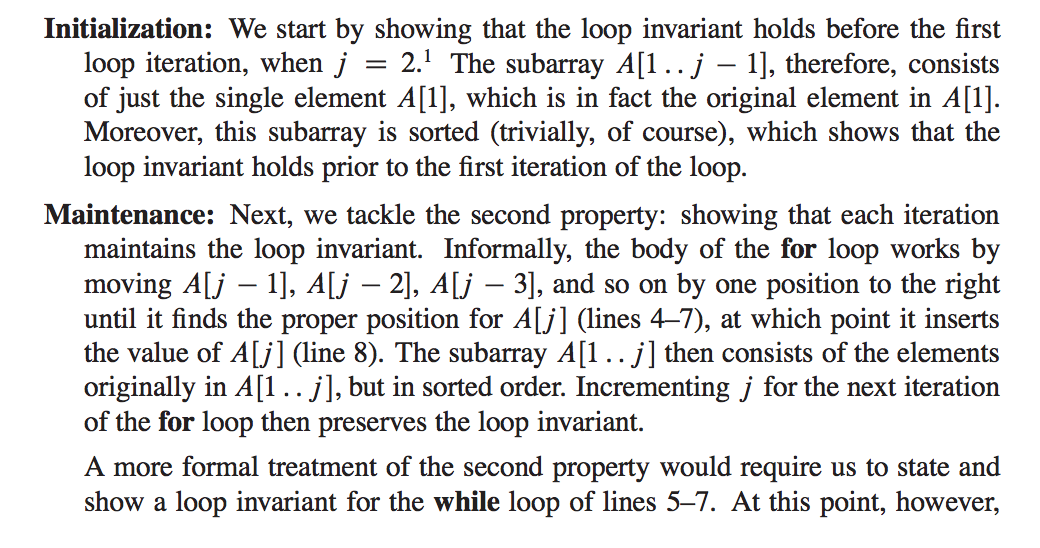
At the start of each iteration of the for loop of lines 1–8, the subarray A [j – 1] consists of the elements originally in A [j – 1], but in sorted order.

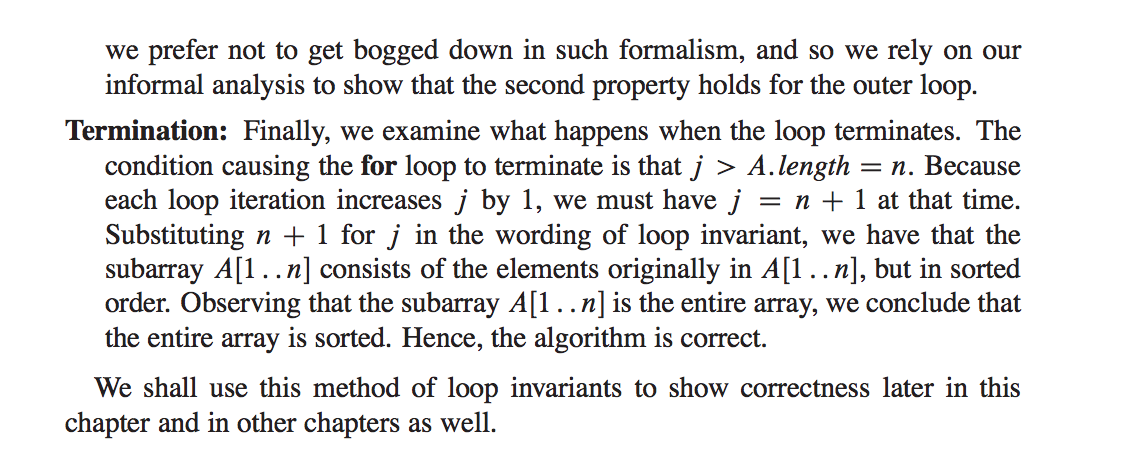
Initialization: It is true prior to the first iteration of the loop. State why the loop is true before first loop.

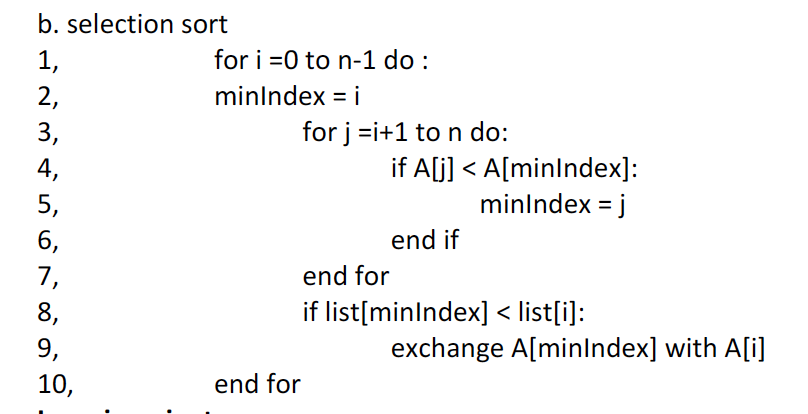
Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.

Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

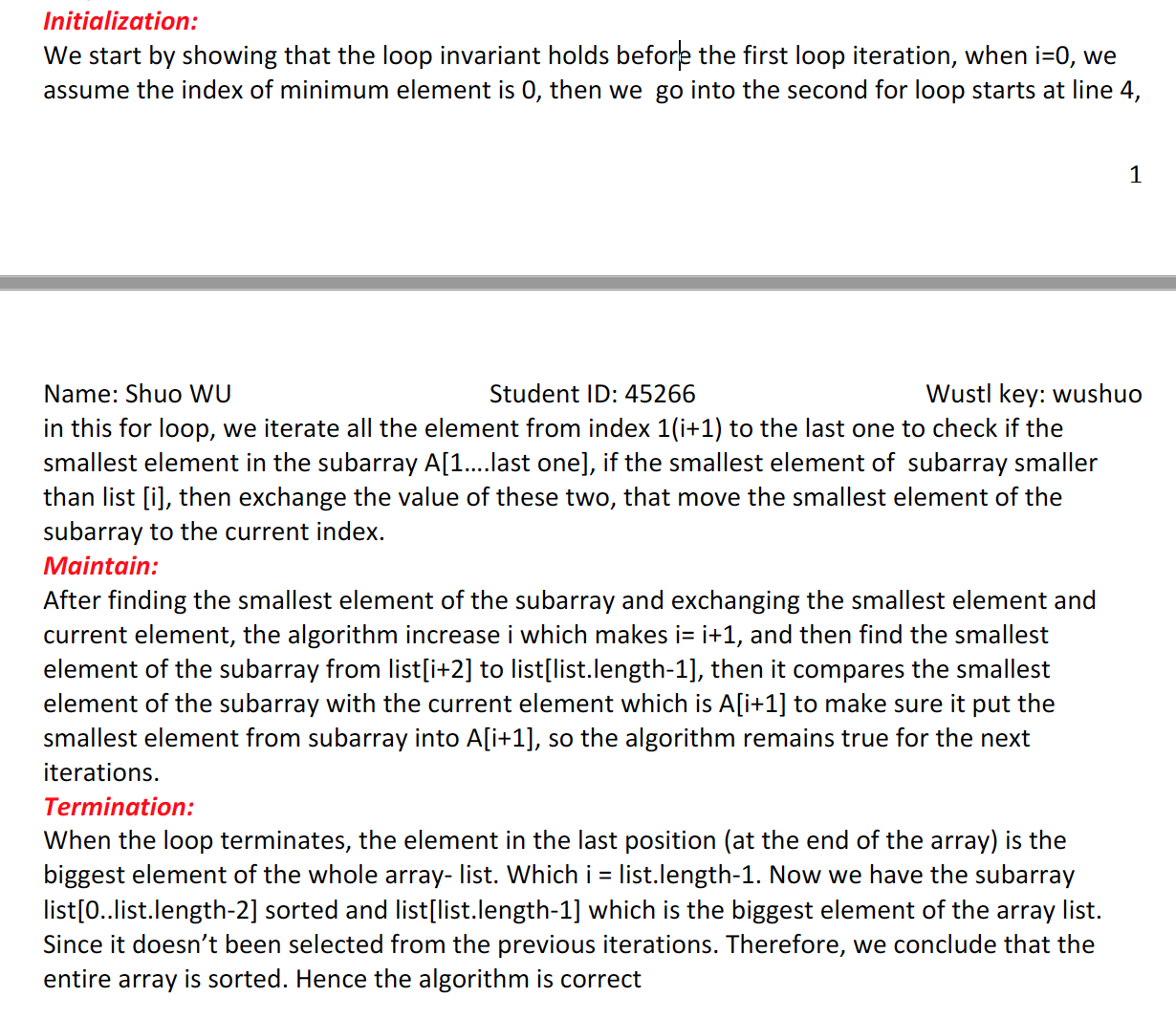






selection sort 

loop invariants:



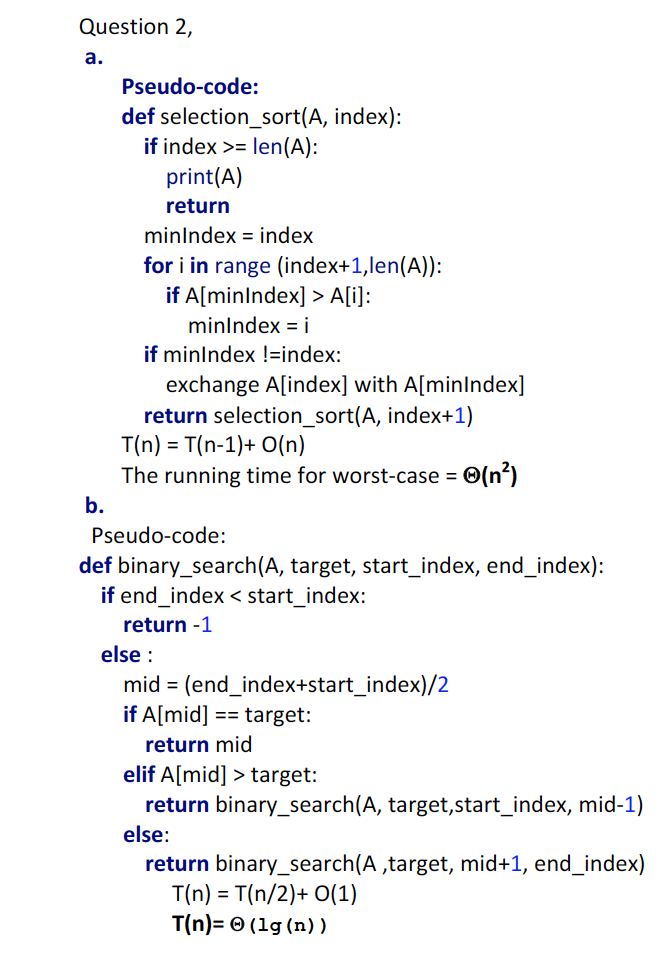
master method:

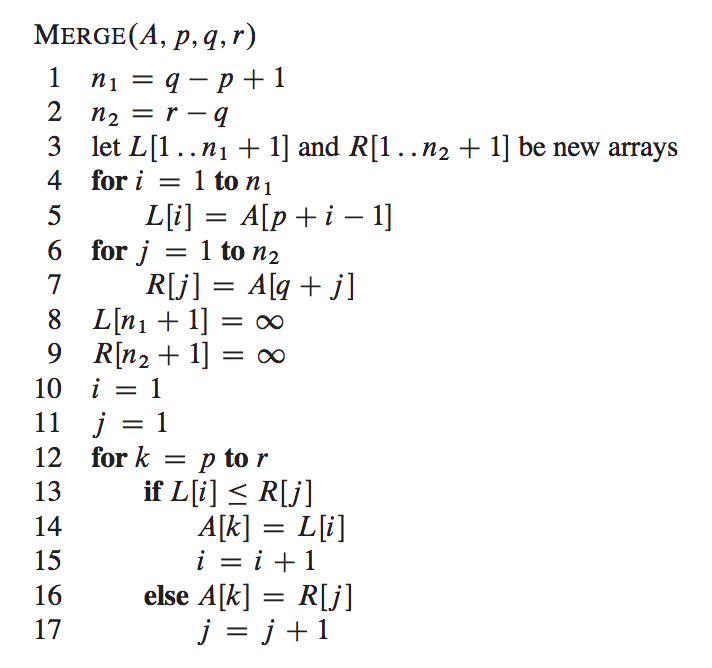
T(n) = aT(n/b)+f(n):

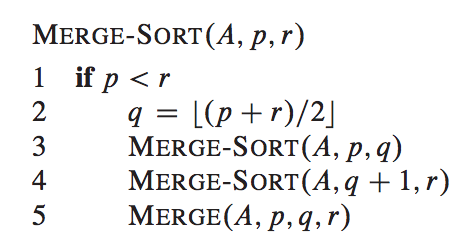
= nlogba

function loop invariant (need to state invariant):

pseudo code for selection-sort and binary search:

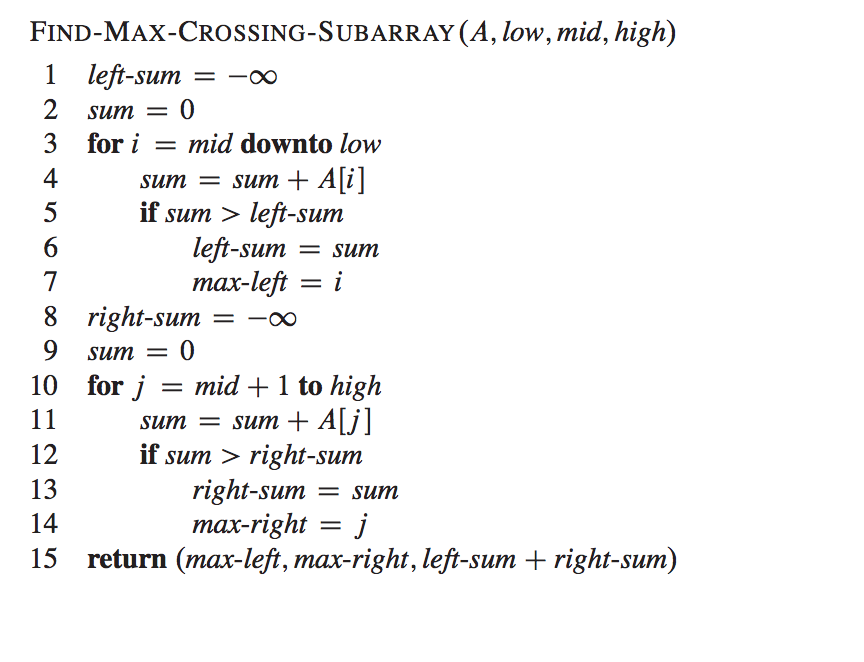


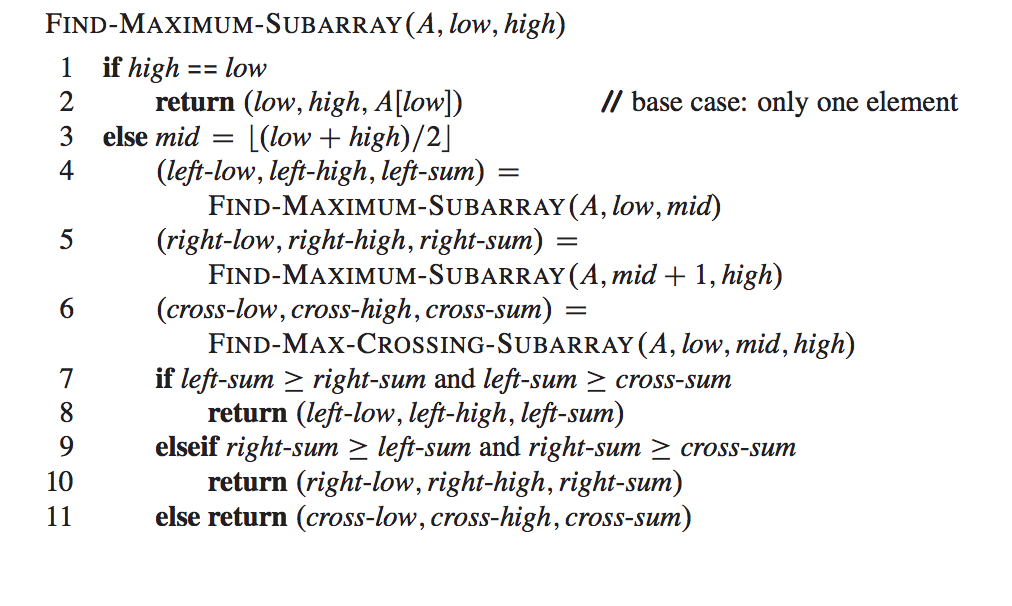




time complexity: T(n) = nlogn

Find-max-subarray☹: find maximum subs array sum: T(n)=2T(n/2)+n





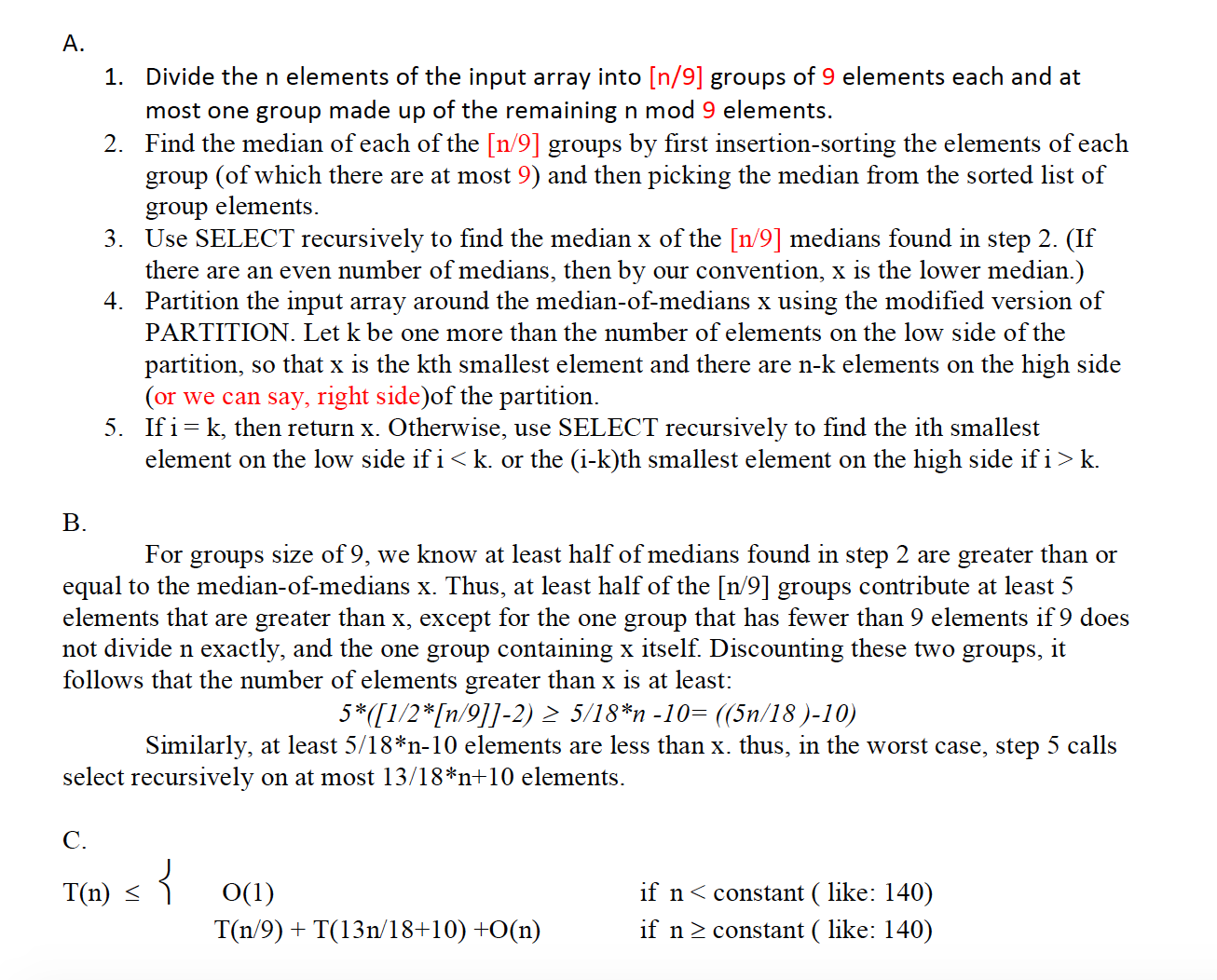
how to deal with: T(n)= T(n-1) +T(n/2) +n?

assume T(n)≤S(n) = 2T(n-1) +n = 2n

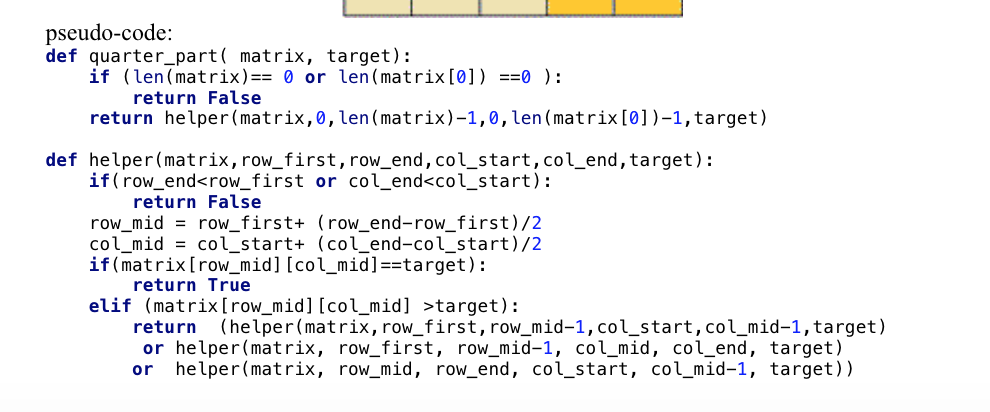
substitution(proof):

T(n)= T(n-1) + T(n/2)+n ≤ 2n-1 +2 n/2 +n≤ 2\*2n-1 =2n

Selection in worst-case linear time:



pseudo-code for finding element in matrix

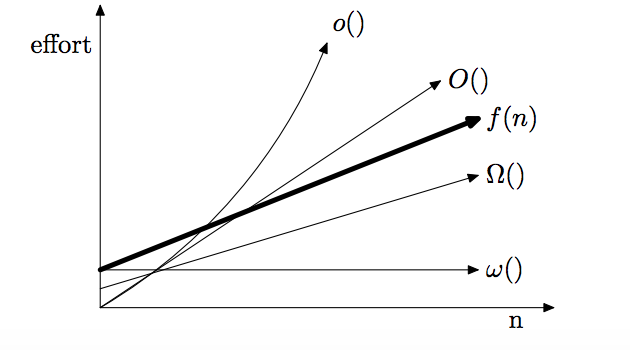


Definition (Little–o, o()), We say that f(n) is o(g(n)) if for any real constant c > 0, there exists an integer constant n0 ≥ 1 such that f(n) < c ∗ g(n) for every integer n ≥ n0.

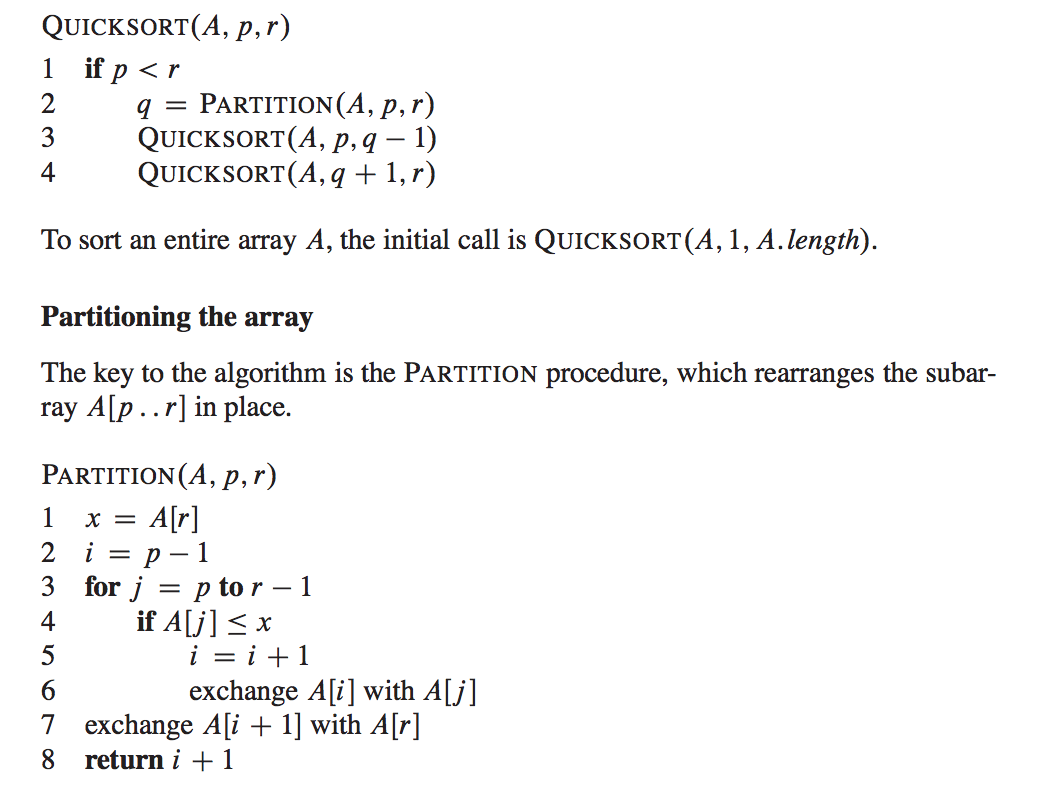
Definition (Little–Omega, ω()):

We say that f(n) is ω(g(n)) if for any real constant c > 0, there exists

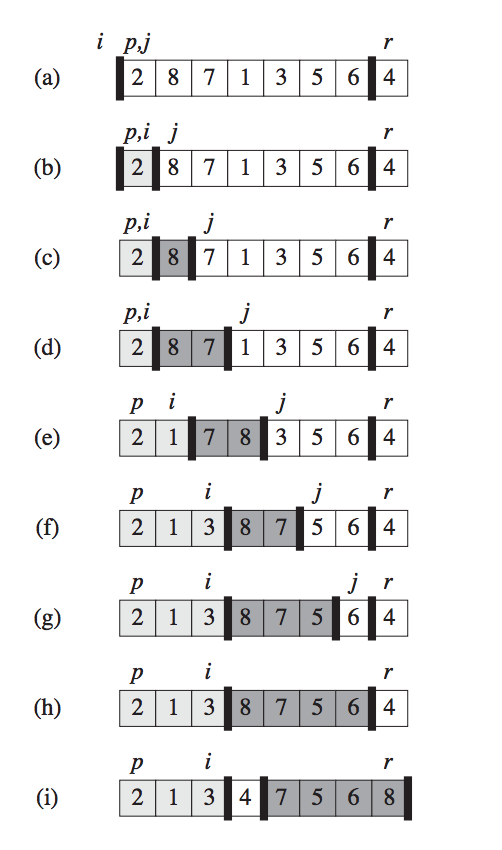
an integer constant n0 ≥ 1 such that f(n) > c · g(n) for every integer n ≥ n0.

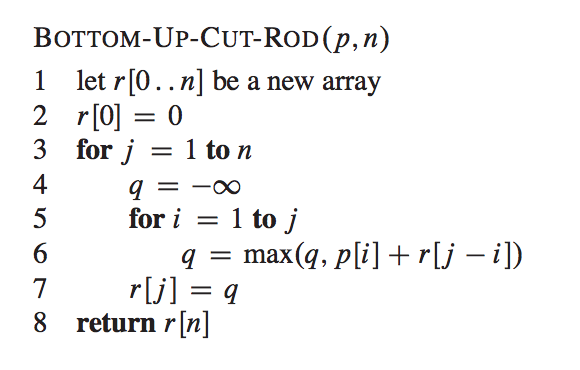


quick sort :



shows how partition works:

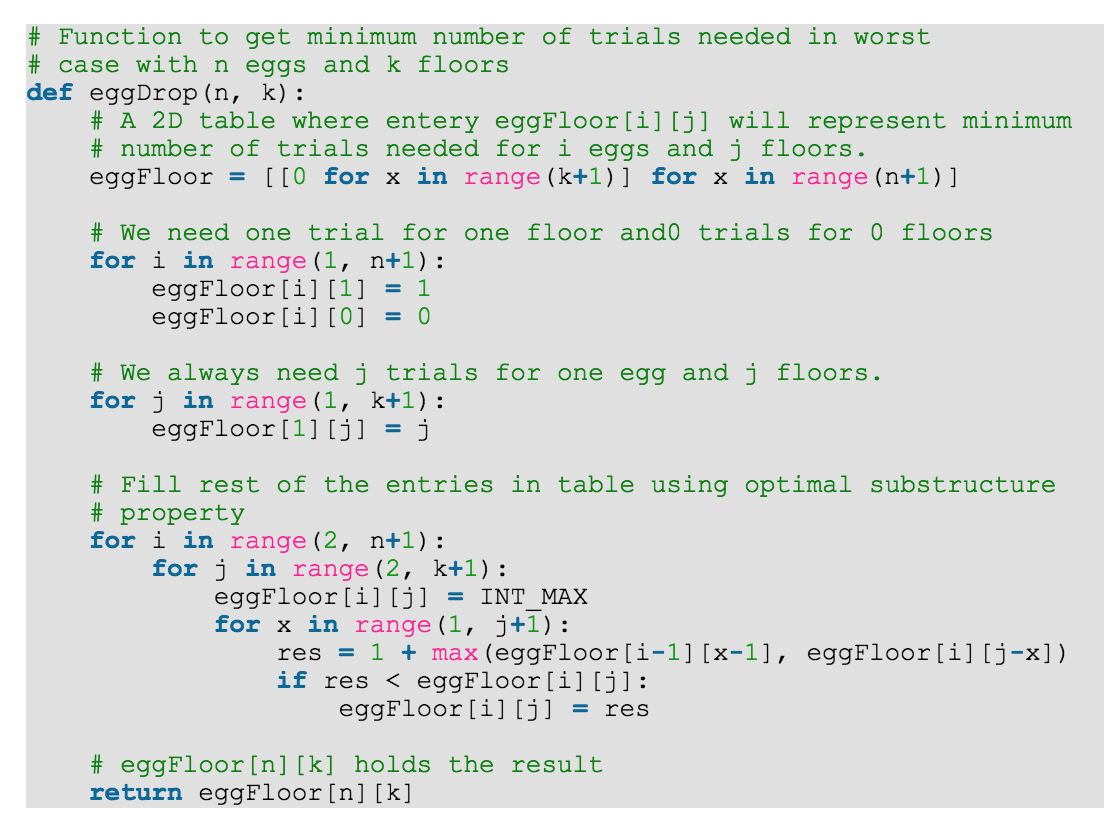
Dynamic-programming:



time complexity is O(n2), when each cut incurs a fixed cost of c, we need to change line 6 into: q= max(q,p[i]+r[j-1]-c) and line 5, for i=1 to j-1, since we might make no cuts,

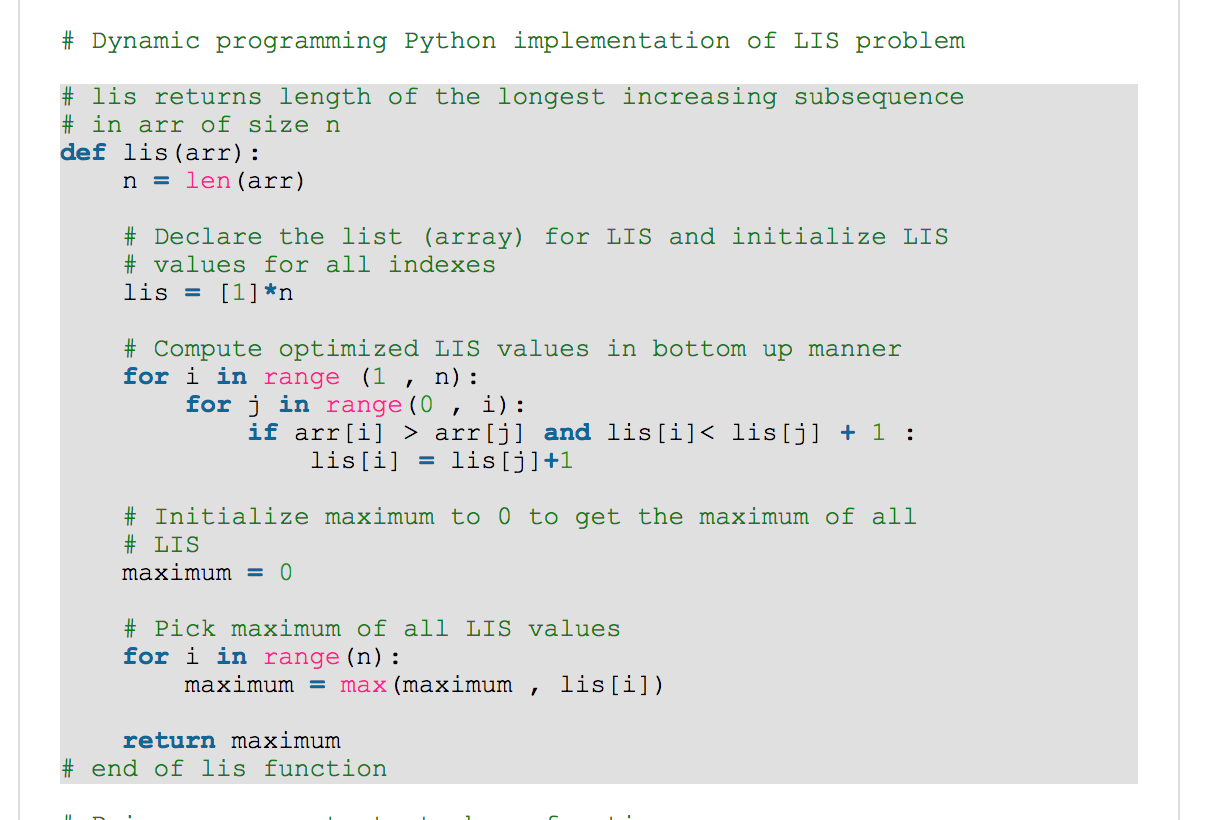
and modify line 4, q = p[j]

Egg-dropping:



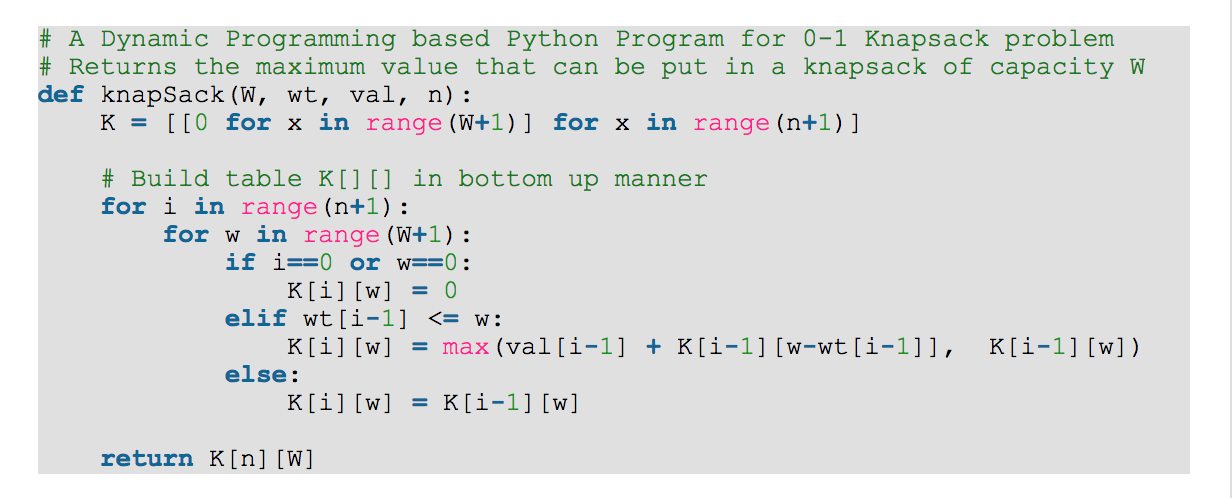
time: T(n) = O(nk2) space = O(nk)

longest increasing subsequence:



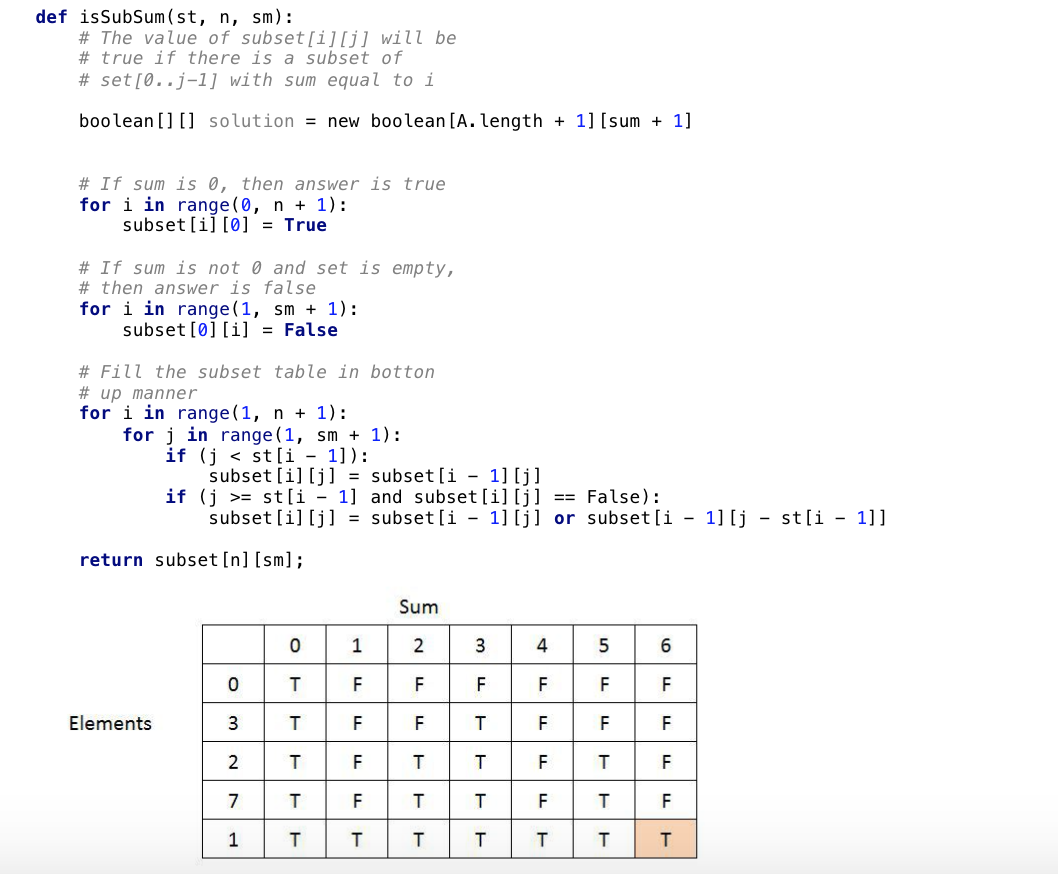
time T(n) = O(n2)

knapsack problem:



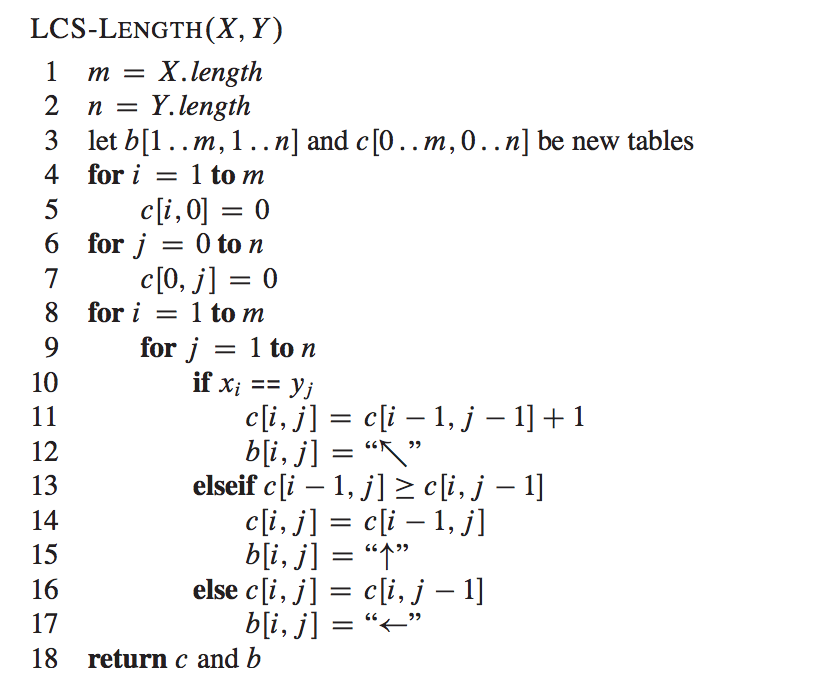
time T(n)=O(nW) n: the number of items and W is the capacity of knapsack.

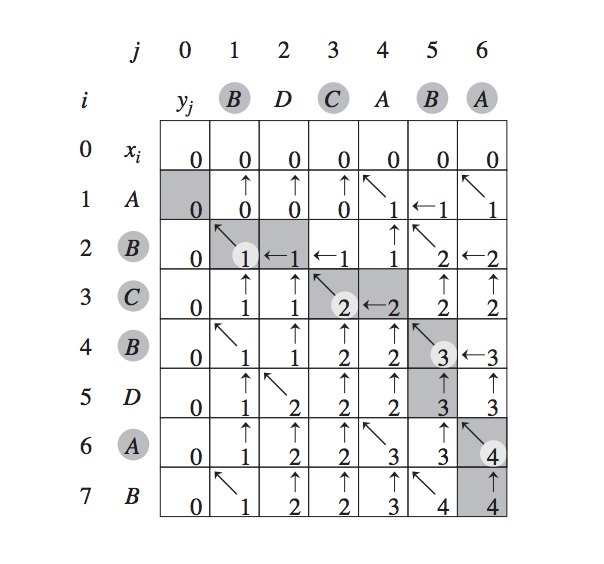
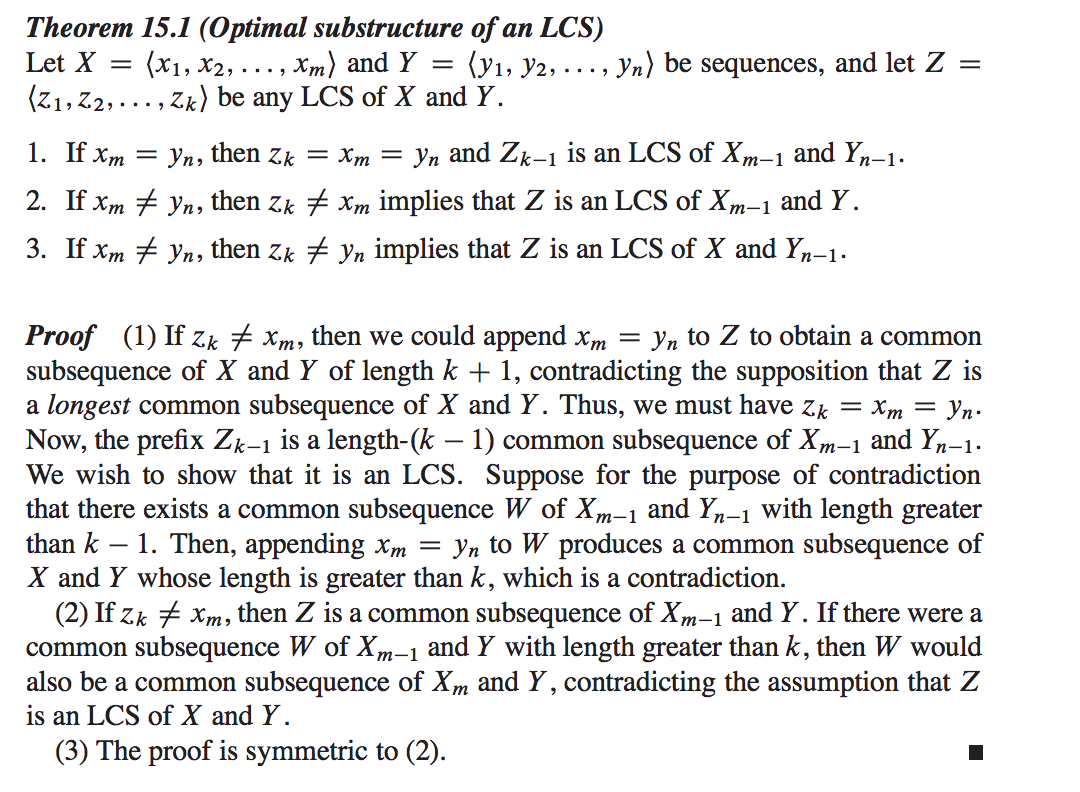
Find sum in a given set:



Time T(n) = O(SUM\*n) where n is elements.

Longest common subsequence:





DYNAMIC\_ACTIVITY\_SELECTOR(S):

initialize c[i,j] = 0

for i <- 1 to n

do for j <- 2 to n

do if i >= j

then c[i,j] <- 0

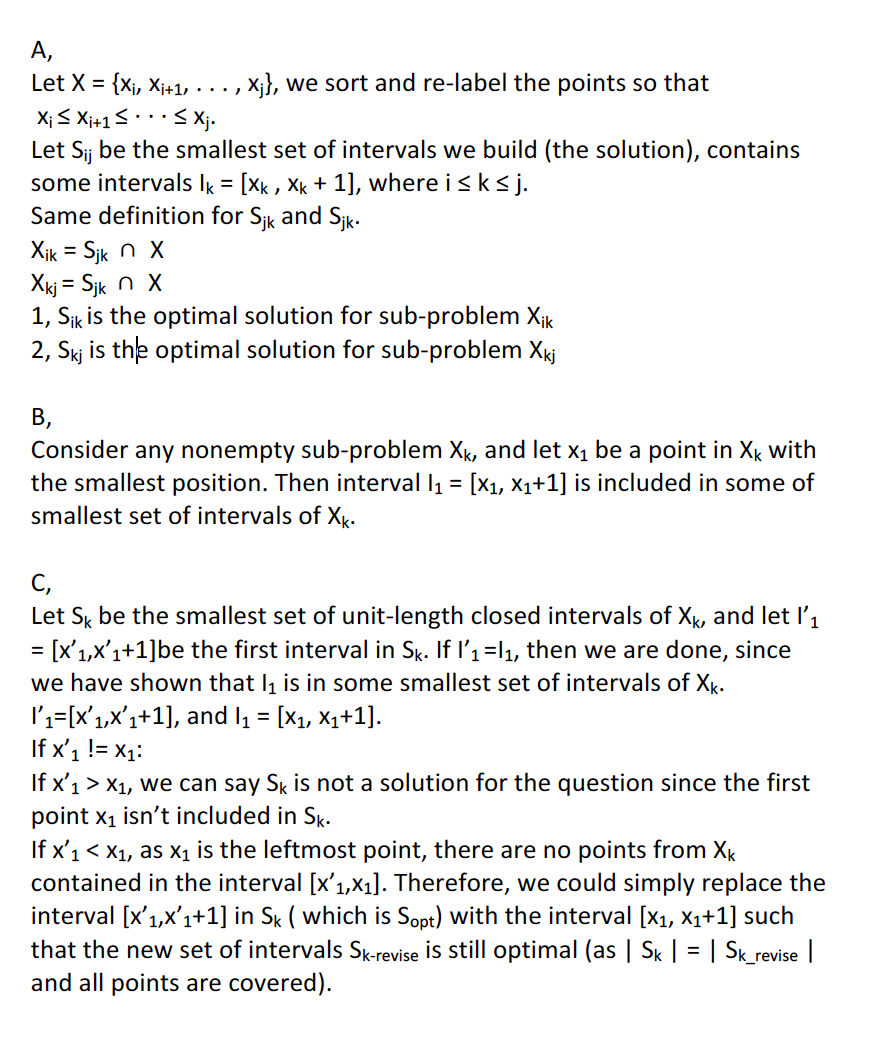
else

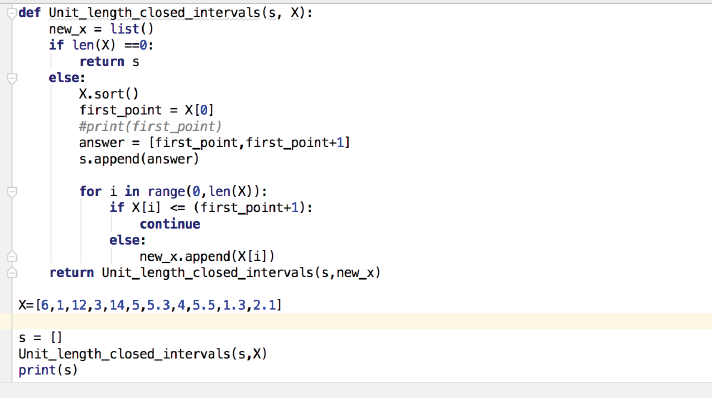
for k <- i+1 to j-1

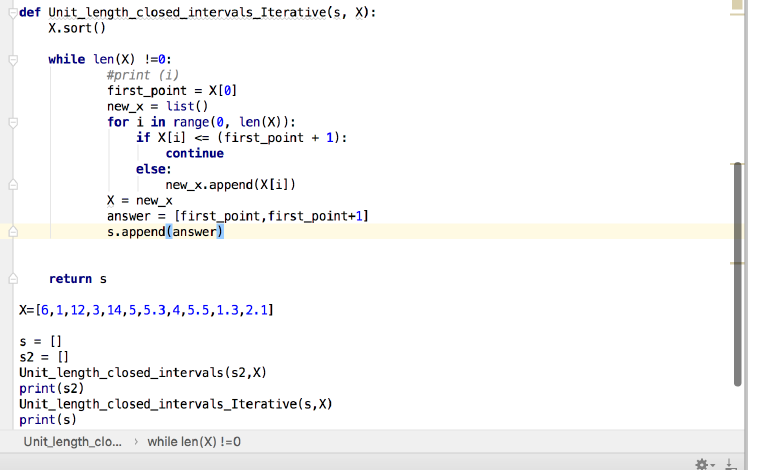
do if c[i,j] < c[i,k] + c[k,j] + 1 then c[i,j] <- c[i,k] + c[k,j] + 1

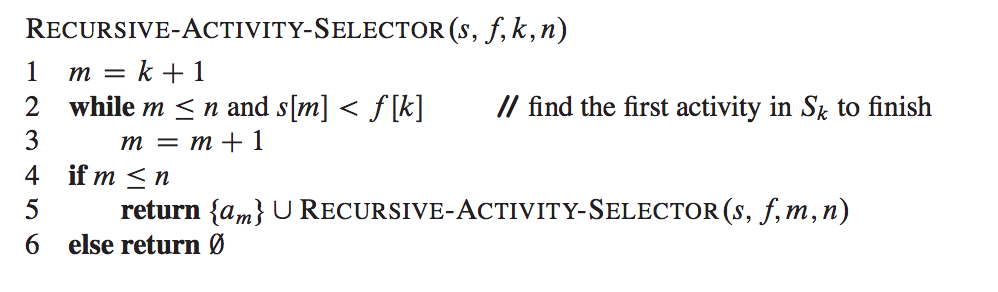
s[i,j] <- k

T(n)= O(n3)

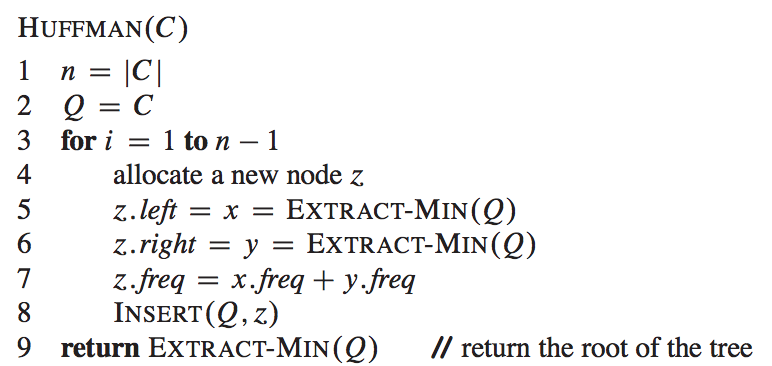


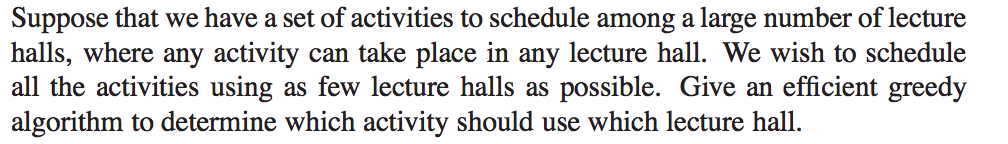


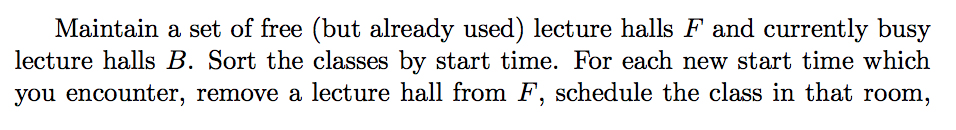


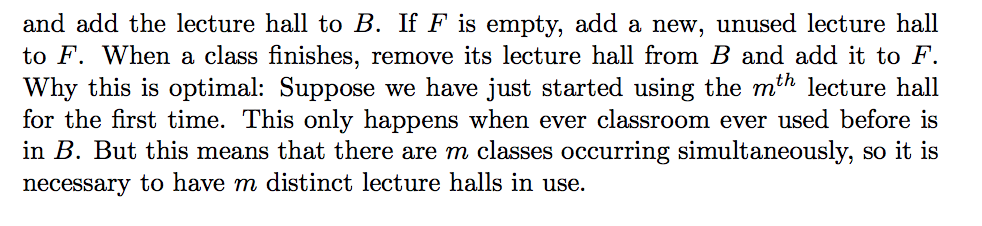


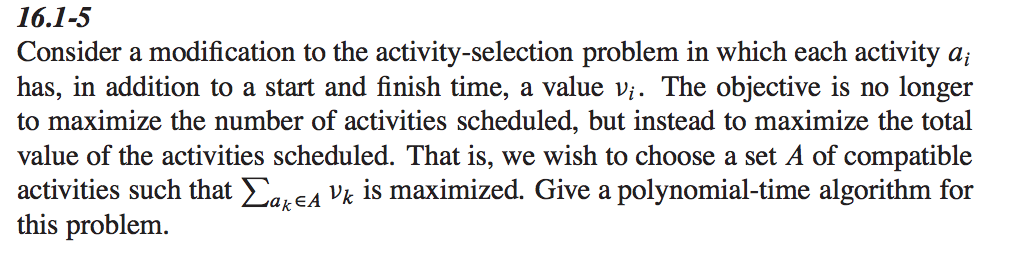
Huffman-coding:

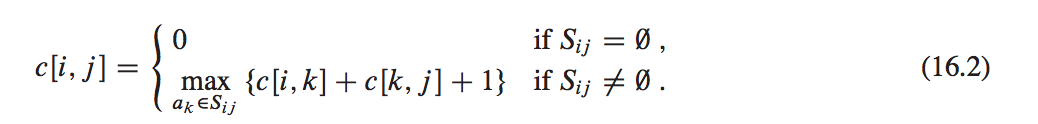


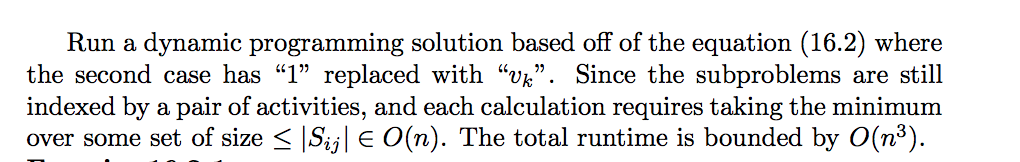












just change value of 1 into Vk in the left corner.

