**# Software Engineering Algorithms 2015**

**1a - Divide and Conquer, Dynamic Programming**

**1ai) Q: Briefly, explain principle of D&C and what are 3 steps applied to solve.**

The principle of D&C is to divide a problem into smaller, simpler, problems of the same structure, the results of which can be combined to solve the original problem.

The three steps applied are the divide step, then conquer step, then the combination step.

**1aii) Q: Subproblems - which property would suggest using DP instead of D&C? Why is DP the preferred technique when this property is present?**

The property is subproblems that overlap, or in other words, a proportion of subproblems that are required more than once. Subproblems share sub-subproblems.

D&C would recompute all of these overlapping problems, leading to added time complexity.

DP would store the values computed, so any subproblem is only computed once, reducing the order of time complexity, which makes DP preferable.

**1aiii) Q: Explain briefly what is *memoization*? How does it help to improve the running time?**

Memoization is a method of saving the results of recursive calls in a tabular form, which is passed as an argument to the recursive function. It helps reduce running time as it means that any recursive calls that have already been computed are saved so do not have to be computed again.

**1b) Approximate string matching**

**1bi) LCS and Levenshtein Question**

LCS= insert and delete

Levenshtein+=replace

**1c) Randomised Quicksort**

i) If the list was already sorted then the last element in the array would be the largest. This would mean that the items would all go to the left of the pivot meaning that the subproblem is still very large, almost as large as the original problem (and vice versa with the smallest element if the list is reverse sorted. Includes the case where all elements are the same). This harms the running time since the division is not helping much. If we randomised the pivot selection, we are less likely to choose the worst pivot (biggest/smallest) since the data in its natural form is more likely than random data to have the extreme values at the end.

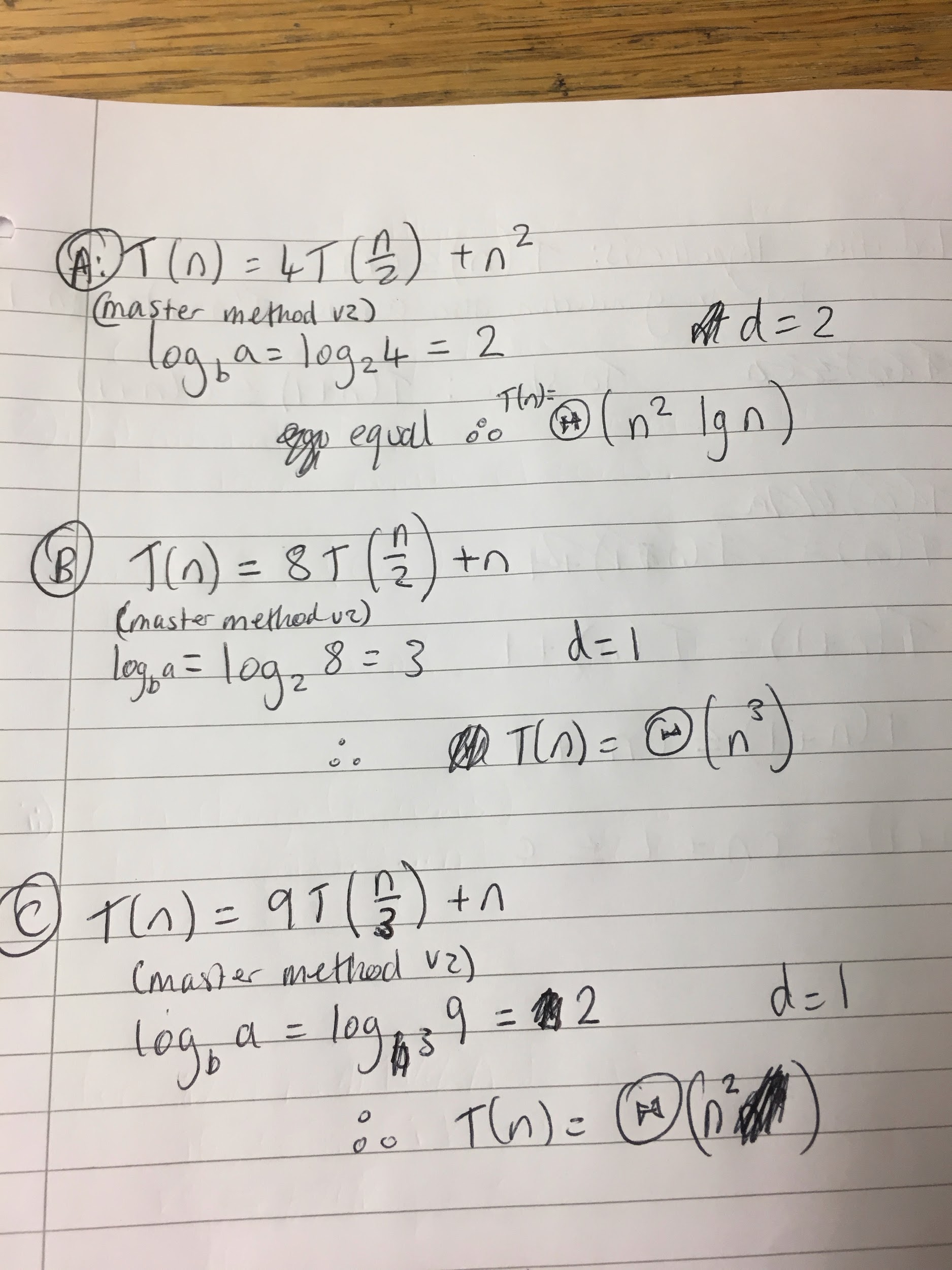
ii) Worst: 0(n2)

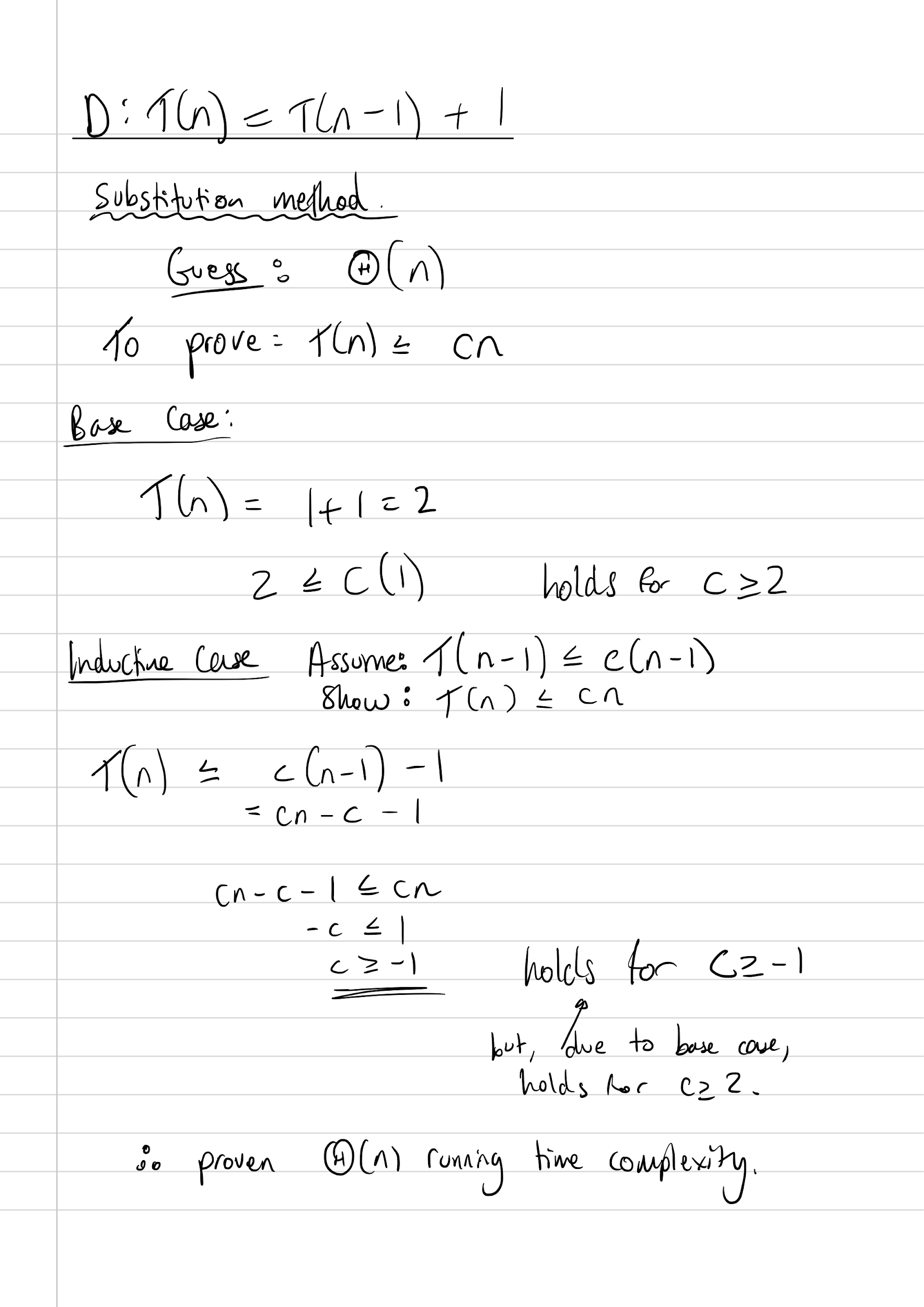
Best: 0(nlogn)

iii) Smaller lists because quicksort has extra overhead for the recursive calls which makes more of a difference than the saving in time from the algorithm itself for small problems.

**1d) Recurrences**

**1di) Write down big O running time complexity of the recurrences and put them in increasing order of complexity.**



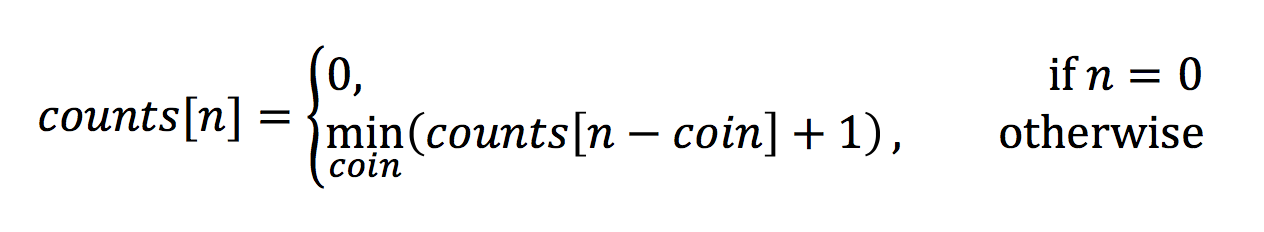


(I think i made an error above, in first line of inductive case should be +1 not -1? Will check properly if have time)

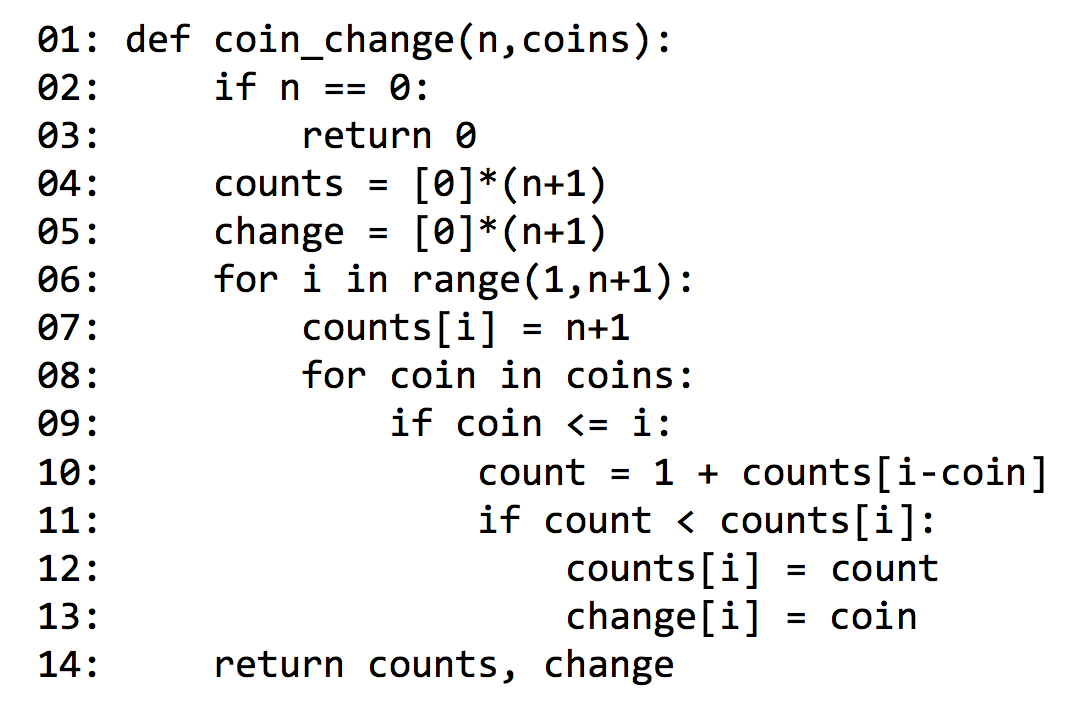
Therefore, order in increasing complexity is: D (n), C (n^2), A (n^2 log n), B (n^3).

**2a) Coin Change Problem**

**2ai) Mathematical Recurrence Relation (in DP solutions slides)**



**2aii) Pseudo-code bottom up coin change algorithm, with a reconstruction mechanism (also in DP solution slides)**



**2aiii) Prove that greedy, picking highest value coin, isn’t always optimal**

E.g. coins = {1,4,9,11} and n = 13.

Greedy chooses 11, 1, 1 c=3

However, optimal is 9,4.

(Would also work with other numbers e.g. {1, 15, 25} and the sum 30.)

**2b) Graph Algorithms**

\*insert fun here\*

i)

Max flow = 7

Minimum cut = {<s, v1>, <v2, v1>, <v2, v4>} or { <v3, t> , <v4, t> }

ii) Final residual network:

Io

