COMP361 Exam Doc

PUT ANSWERS IN RED

2017 Exam Topics

1. Definitions
2. Greedy Algorithms
3. Dynamic Programming
4. Approximation & Probabilistic Algorithms (20 marks)
5. Guest lectures its 10 marks, he already mentioned it in the lecture

"there may be any topic up until the lecture by Mark Moir", e.g. linear programming could come up in definitions

First 3 easier, #4 on the harder side, no really hard question like 2016. All relatively solveable.

Definitions Section

This is a laundry list of things which I think might come up. Please add to it!

Sorting

- Formal definition of sorting
- Stable vs unstable sorting
    - https://stackoverflow.com/a/1517824 Stable sorting is where elements that have the same value appear in their original order after sorting. Important if they point to different things.
    - Unstable sorting is where elements with the same value may appear in any order.

Divide & Conquer

- What defines a D&C algorithm?
- Typical structure of a D&C algorithm
    - Divide - Split a problem into smaller, independant problems
    - Solve - Solve each sub-problem
    - Combine - Merge the sub-solutions together to form a solution for the initial problem
- Requirements for a D&C proof
    - Direct Case - Assuming a condition A, prove that a statement R is held
    - Divide - Assuming A, prove that the subproblems share the same assumption for their given problem size.
    - Combine - Assuming A and R_i holds for each subproblem (1..i), show that combining the sub solutions maintains R
- The meaning of each term in the master theorem: curly L, k, f(n)
    - ℓ = number of recursive calls, sometimes shown as ‘a’
    - k = size input is reduced by each recursive call, sometimes shown as ‘b’
    - f(n) = cost of divide + combine
    - T(n)= ℓT(n/k) + f(n) // straight forward if you look at the master theorem

Master Theorem (Simplified Version)

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

Steps to solve master theorem questions:

1. Extract ‘a’, ‘b’, and f(n) from recurrence relation (see meaning of master theorem section for what the variables represent)
2. Determine $n^{logb(a)}$
3. Compare f(n) with $n^{logb(a)}$ asymptotically (is f(n) bigger, smaller, or equal to $n^{logb(a)}$)
4. Determine appropriate master theorem case and apply it

Master Theorem Cases (Informal)

1. If f(n) is smaller than $n^{logb(a)}$and have to express order in terms of f(n),

T(n) ∈ $\Theta(n^{logb(a)})$

2. If f(n) is equal to $n^{logb(a)}$, then T(n) ∈ $\Theta(n^{logb(a)} \log n)$
3. If f(n) is greater than $n^{logb(a)}$, then T(n) ∈ Θ(f(n))

See https://www.youtube.com/watch?v=6CX7s7JnXs0

Greedy Algorithms

- What defines a GA algorithm?
- Conditions for a GA to be suitable
- Requirements for a proof of GA correctness
    - Acceptable solution: capture what is a correct solution
    - Feasible partial solution: every feasible partial solution can be extended to an acceptable solution
    - Optimal substructure: (will need to be able to prove this) Every optimal solution is a composition of optimal partial solutions
    - Greedy choice: Every optimal partial solution may be extended by a greedy choice. Greedy choice being some simple next logical step. Eg. shortest available path.

Dynamic Programming

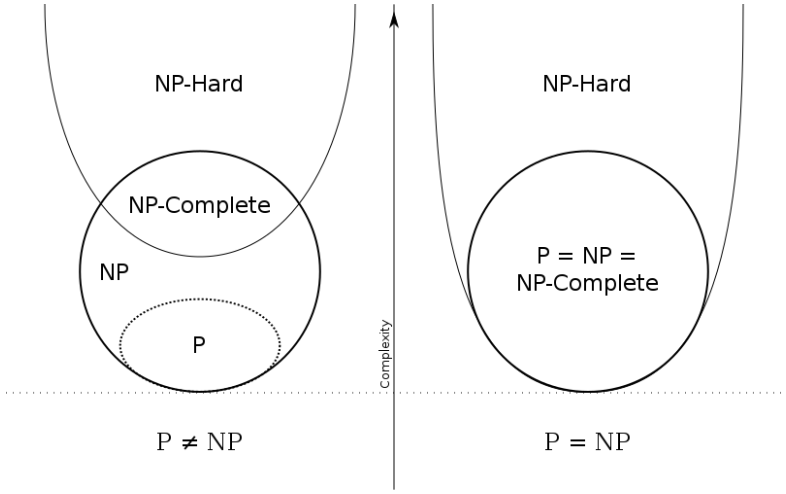
- What defines a DP algorithm?
- Requirements for a proof of DP correctness
    Just optimal substructure?
- Memoization
    - Involves using previously calculated values from a table instead of re-calculating values. Known as caching

Graphs

- Requirements for a proof of GS correctness
- Defn of connected, complete, euclidean graphs
- Defn of cycle, hamiltonian cycle
    - Hamiltonian cycle is a cycle that starts and ends at the same vertex and passes through every other vertex exactly once.
    - A cycle is a path that goes along a graph, starting and ending at the same vertex/node
- Pruning / Bounding?
    - Removing unnecessary paths in a graph. Ie like we enumerated the possibilities in the knapsack, instead of putting more objects, cut off the path when the weight is overflowing

Complexity

- Definition of O, Θ, Ω
  - O Big Oh: Upper bound {f(n)|(∃ d)(aa n)[0≤f(n)≤d.g(n)]}
  - Θ Big Theta: Exact bound {f(n)|(∃ c>0,d)(aa n)[0≤c.g(n)≤f(n)≤d.g(n)]}
  - Ω Big omega: Lower bound {f(n)|(∃ c>0)(aa n)[f(n)≥c.g(n)≥0]}
- Definition of P, NP, NPC, NP-Hard
  - P: Polynomial time problems, able to be SOLVED by a deterministic turing machine in $O(n^k)$ time, for some k. Sometimes considered 'easy' problems.
  - NP:
    - CHECKABLE by a DTM in polynomial time
    - able to be solved by a nondeterministic turing machine in polynomial time (somewhat theoretical since NTMs don't exist)
  - NPComplete: A ∈ NPC if A ∈ NP and for every B ∈ NP $B \leq^P A$
  - NPHard: A ∈ NPH if every B ∈ NP, $B \leq^P A$
    - (or, equivalently, that there exists C ∈ NPC, $C \leq^P A$)
    - "as hard as NPC, but not necessarily in NP"
- What does $A \leq^P B$ mean?
  - A is polynomial-reducible to B. (A and B are problems)
  - that is: inputs of A can be converted to inputs of B in polynomial time; and results from B can be converted to results from A in poly. time
- The venn diagram



- Here's some problem. What minimum complexity Ω does it have?
  - element selection
  - comparison-based sort
- Prove that P!=NP (5 marks and a Nobel Prize in Computer Science).

Linear Programming

- What's the main idea of LP? (i.e. how are problems represented?)

- LP is basically getting a set of values and trying to maximise (or minimise) a certain output
- How does the simplex algorithm work?
- Draw a diagram

Approximation & Probabilistic

- What distinguishes traditional, approxim'n, and prob'stic algorithms from each other?
- Def'n of Numerical, Las Vegas, Monte Carlo, and Sherwood
    - Las Vegas: Always gives a correct answer at some point in time. As the program runs the likelihood of it providing the answer increases.
    - Monte Carlo: always give an answer, but there is some probability of being completely wrong. The probability of an incorrect answer decreases over time.
    - Numerical: give an approximation to the correct answer. As the algorithm runs for longer the answer gets more accurate
    - Sherwood: Variant of the Las Vegas model which attempts to change aspects of the problem to minimise the impact of worst case scenarios.
- Examples of Numerical, Las Vegas, Monte Carlo, and Sherwood

Guest Lectures

- John Lewis talked about a program that could generate any/every image (and therefore video) (there were no slides posted and I didn't take notes)
    - "Describe the main idea of Kolmogorov Complexity" will be the most likely question
- Yi Mei on deep learning
    - Deep Learning: Machine learning algorithms based on learning multiple levels of representation / abstraction.
    - Common tasks include; classification, regression, and clustering.
- Mark Moir on blockchain
    - Agreement on updates to shared data
    - Tamper-proof, immutable record of agreed updates
    - Update governed by flexible, precise rules (smart contracts)
    - No trusted intermediary required
    - Bitcoin/Ethereum

2014 Exam

Questions	Marks
1. Divide and Conquer	[30]
2. Greedy Algorithms	[35]
3. Dynamic Programming	[30]
4. Approximation Algorithms	[25]

Question 1. Divide and Conquer

(a) [4 marks] Use pseudocode to describe the basic structure of a typical divide-and-conquer algorithm. Explain the components of your algorithm.

General algorithm

Suppose:

- $P(n)$ is the problem we are trying to solve, of size n
- **Direct** solves $P(n)$ directly, for sufficiently small n ($n \leq n_0$)
- **divide** divides the problem $P(n)$ into subproblems $P_1(n_1), P_2(n_2), \ldots, P_k(n_k)$ for some constant k
- **combine** combines solutions for $P_i(n_i)$ ($i \in 1..\ell$) to solve $P(n)$, note that $\ell \leq k$ (e.g. in binary search).

divcon(P, n)
  if $n \leq n_0$
    Direct
  else
    divide
    for $i \leftarrow 1$ to ℓ do **divcon**(P_i, n_i)
    combine

(b) [4 marks] State a requirement that an array is sorted which can be used to prove that a sorting algorithm is correct (similar to how we did it in the first few lectures).

I don't think the answer below is what this question asks for. In the first few lectures we went through a technique for proving sorting algorithms are correct using loop invariants and ensuring the loop invariant satisfies at 1. Initialization, 2. Maintenance and 3. Termination of the loop.

~~For all p and q such that p < q, in the sorted list S' we have S'[p] <= S'[q]~~

1. declare an **invariant** condition for each iteration *j*:
    a. elements before j are a **permutation** of original dataset
    b. elements before j in new set are **ordered**
    c. elements after j are unchanged from original
2. Show that the loop invariant holds prior to the first iteration
3. (Induction) Show that if the invariant holds at iteration *j*, it will hold at *j*+1

~~How about this?~~
~~Suppose the problem description P(n) is made up of an assumption A and a requirement R.~~

~~The following conditions are sufficient to ensure divcon is correct:~~
~~Direct~~
- ~~may assume A and n ≤ n0~~
- ~~must establish R~~

~~Divide~~
- ~~may assume A and n > n0~~
- ~~must establish ni < n and Ai for each i ∈ 1..ℓ~~

~~Combine~~
- ~~may assume A and Ri for each i ∈ 1..ℓ~~
- ~~must establish R.~~

Consider the following algorithm that sorts the array **A** between indices **i** and **j** (initially set to the first and last element indices of the array **A**):

```
THIRDS-SORT(A, i, j)
1  if A[i] > A[j]
2      then exchange A[i] ↔ A[j]
3  if i + 1 ≥ j
4      then return
5  k ← ⌊(j − i + 1)/3⌋          // Round down.
6  THIRDS-SORT (A, i, j − k)    // First two-thirds.
7  THIRDS-SORT (A, i + k, j)    // Last two-thirds.
8  THIRDS-SORT (A, i, j − k)    // First two-thirds again.
```

(c) [7 marks] Give the general structure of the proof of correctness of a divide and conquer algorithm, and use it to show that the THIRDS-SORT algorithm above correctly sorts the input array.

Let problem P = assumption A + requirement R + problem size N
In this case: A = nothing, R = the list is sorted, n = j - i
Steps of the process:

solveDirect is correct
- Assume A and $n \leq n_0$ *(the if condition)*
- establish R *(the kernel is here)*

divide is correct
- Assume A and $n > n_0$
  (the else condition)
- establish $n_i < n$ and A_i for each iI
  (the subproblems are smaller, and are valid problems)

combine is correct
- Assume A, A_i, R_i for each i
- establish R

Direct case
if n = 0, the list is a singleton, so it's already sorted: R satisfied
If n = 1:
If a[1] < a[2], already sorted
otherwise, it will swap them, which will make them sorted

Divide
Assume: n >= 1
Therefore: k >= (n + 1)/3, i.e. k >= 1

So problem sizes will be either:
- (j - k) - i < j - i
- j - (i + k) < j - i

Which makes them smaller, as required

Combine
If i...j-k is sorted
And i+k...j is sorted
Then clearly i...j is sorted

(d) [7 marks] Give a recurrence relation for the worst-case running time of THIRDS-SORT and a tight asymptotic (Θ) bound on the worst-case running time.

Analysis

$$T(n) \leq \begin{cases} T_{dir}(n) & n \leq n_0 \\ \ell T\left(\frac{n}{k}\right) + T_{div}(n) + T_{comb}(n) & n > n_0 \end{cases}$$

So here curly_l = 3, k = 1.5 → α = 2.7; divide and combine operations are both o(1)
So by master method, it's in θ(n^2.7)

(e) Compare the worst-case running time of THIRDS-SORT with that of:

(i) [2 marks] insertion sort
n^2
(ii) [2 marks] mergesort
n log n
(iii) [2 marks] heapsort
n log n

(iv) [2 marks] quicksort.
n^2

so it's worse than all of them.

Question 2. Greedy Algorithms

Consider the problem of making change of n cents using the fewest number of coins. Assume that each coin's value is an integer.

(a) [10 marks] Describe a greedy algorithm to make change consisting of 25c, 10c, 5c, and 1c coins. Prove that your algorithm yields an optimal solution. Follow the lecture slides style of presenting and proving greedy algorithms.

PTO for answers :)

(b) [10 marks] Suppose that the available coins are the powers of c, i.e. values are c 0 , c 1 , . . . , c k for some integers c > 1 and k > 1. Show that the greedy algorithm always yields an optimal solution.

(c) [5 marks] Give a set of coin values for which the greedy algorithm does not yield an optimal solution. Your set should include a 1c coin so that there is a solution for every value of n.

(d) [10 marks] Give an O(nk)-time algorithm that makes change of n cents using coins from any given set of k different coin values, assuming that one of the coins is 1c. Show that the time your algorithm takes is as required.

Optimal Substructure: $Opt(n) = \{c\} \cup Opt(n-c)$

s.t. $c = \max(1, 5, 10, 25)$,

$n - c \geq 0$.

and we can see that each optimal solution is composed of solutions to subproblems.

Algorithm:

```
change(N):
  if N=0:
    return {}
  for c in (25, 10, 5, 1):
    if N >= c:
      return {c} + change(N-c)
```

Correctness:

1. It terminates: all loops finite, and recursive calls are to smaller problems (values of N)
2. Acceptable solution: only returns a set consisting of values $c_i \in (25, 10, 5, 1)$, and $\sum c_i = N$. (by last line)
3. Follows optimal substructure: see above.
4. Greedy choice property: Proof by contradiction.

   Let G be our algorithm's solution, and O be the optimal sol^n.

   Suppose $G \neq O$.

   Then let g_i be the first coin $\in G$ and $\notin O$.

   Then we know $g_i > O_i$, so there must exist $o_j \ldots o_k$ that sum to the rest of g_i's value. solution.

Then replace {oi, oj…ok} with gi and you get a better solution than O.

Question 3. Dynamic Programming

A certain string-processing language allows a programmer to break a string into two pieces. Because this operation copies the string, it costs n time units to break a string of n characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks occur can affect the total amount of time used. For example, suppose that the programmer wants to break a 20-character string after characters 2, 8, and 10 (numbering the characters in ascending order from the left-hand end, starting from 1). If she programs the breaks to occur in left-to-right order, then the first break costs 20 time units, the second break costs 18 time units (breaking the string from characters 3 to 20 at character 8), and the third break costs 12 time units, totalling 50 time units. If she programs the breaks to occur in right-to-left order, however, then the first break costs 20 time units, the second break costs 10 time units, and the third break costs 8 time units, totalling 38 time units. In yet another order, she could break first at 8 (costing 20), then break the left piece at 2 (costing 8), and finally the right piece at 10 (costing 12), for a total cost of 40.

(a) [10 marks] Design an algorithm that, given the numbers of characters after which to break, determines a least-cost way to sequence those breaks. More formally, given a string S with n characters and an array L[1 . . . m] containing the breakpoints, compute the lowest cost for a sequence of breaks, along with a sequence of breaks that achieves this cost.

```
Initialise array Cost[N x N]

function Fkghl(S, L, i=0, j=N):
if j − i = 0 return 0

if Cost[i, j] is null:
     Cost[i, j] = min([
                (j − i) + Fkghl(S, L, i, k-1) + Fkghl(S, L, k+1,
          j)
                for k in L if i < k < j
          ])
     //memoised D&C: write values to Cost array as we go

     return Cost[i, j]
```

(b) [10 marks] Prove that this problem has an optimal substructure property.

We have a string S with n characters

Pick two characters $i, j: 0 \leq i \leq j \leq n$

Find the breakpoints between these two characters $L_{i,j} = \{k \in L: i < k < j\}$

Then Cost $C(i,j) = \begin{cases} 0 \text{ if } i = j \\ (j-i) + \min\limits_{k \in L_{i,j}} \left(C(i,k-1) + C(k+1,j)\right) \end{cases}$

Thus solution is composed of solutions to subproblems.

(c) [10 marks] Prove that your algorithm is correct.

a) The algorithm terminates, since all loops are finite
b) The problem has optimal substructure (see above)
c) The algorithm respects this optimal substructure by returning 0 if i = j, and returning the cost of the string's length, plus the minimum (wrt potential split points) of the costs of splitting each substring given each potential split point

Question 4. Approximation Algorithms

Suppose you are given a set of positive integers A = a1, a2, . . . , an and a positive integer B. A subset S ⊆ A is called feasible if the sum of the numbers in S does not exceed B:

$\sum a_i \in S\ a_i \leq B$

The sum of the numbers in S will be called the total sum of S.
You would like to select a feasible subset S of A whose total sum is as large as possible. **Example**. If A = 8, 2, 4 and B = 11, then the optimal solution is the subset S = 8, 2.

(a) [10 marks] Here is an algorithm for this problem.

```
Initially S = ∅
Define T = 0
```

```
For i = 1, 2, . . . , n
     If T + ai ≤ B then
          S ← S ∪ ai
          T ← T + ai
     Endif
Endfor
```

Give an instance in which the total sum of the set S returned by this algorithm is less than half the total sum of some other feasible subset of A.

elements: {1, 10}
B limit: 10

it will pick 1, reject 10, and get a result 10% of the best feasible subset (i.e. {10})

(b) [15 marks] Give a polynomial-time approximation algorithm for this problem with the following guarantee:

• It returns a feasible set S ⊆ A whose total sum is at least half as large as the maximum total sum of any feasible set S 0 ⊆ A.

throw away any elements larger than B (obvsly they can't be part of the solution)
start adding elements a1, a2, … a[i] as before
When you get to the element a[i+1] that would tip us over the limit, then either a[1...i] or a[i+1] is more than half of B. So pick whichever's larger. hooray

Your algorithm should have a running time of at most O(n logn) (note that at most means that a running time of Θ(n) is acceptable).

2015 Exam

Questions	Marks
1. Divide and Conquer	[30]
2. Various Algorithms	[35]
3. Dynamic Programming	[30]
4. Approximation Algorithms	[25]
5. Chips But No Fish	[25]

Question 1. Divide and Conquer

(a) [10 marks] Use the master method to give tight asymptotic bounds for the following recurrences:

1. T(n) = 2T(n/4) + 1 ~~Case 1: alpha is 0.5 so O(sqrt n)~~
2. T(n) = 2T(n/4) + √ n ~~Case 2: same alpha so O(sqrt n log n)~~

I agree but don't you mean Θ (theta) instead of O? Surely you'd lose a few marks for this, important. And I would write T(n) ∈ blah blah to prefix it....

Answer:T(n) ∈ Θ(sqrt n) ; 2. T(n) ∈ Θ (sqrt n n log n)

(b) [10 marks] Describe a typical structure of a Divide and Conquer algorithm and outline a typical approach to proving it correct as was described in the lectures and was used to prove the correctness of merge sort etc.

General algorithm

Suppose:

- $P(n)$ is the problem we are trying to solve, of size n
- **Direct** solves $P(n)$ directly, for sufficiently small n ($n \leq n_0$)
- **divide** divides the problem $P(n)$ into subproblems $P_1(n_1), P_2(n_2), \ldots, P_k(n_k)$ for some constant k
- **combine** combines solutions for $P_i(n_i)$ ($i \in 1..\ell$) to solve $P(n)$, note that $\ell \leq k$ (e.g. in binary search).

divcon(P, n)
  if $n \leq n_0$
    Direct
  else
    divide
    for $i \leftarrow 1$ to ℓ do **divcon**(P_i, n_i)
    combine

Correctness

Suppose the problem description $P(n)$ is made up of an **assumption** A and a **requirement** R.

The following conditions are sufficient to ensure **divcon** is correct:

Direct

- may assume A and $n \leq n_0$
- must establish R

divide

- may assume A and $n > n_0$
- must establish $n_i < n$ and A_i for each $i \in 1..\ell$

combine

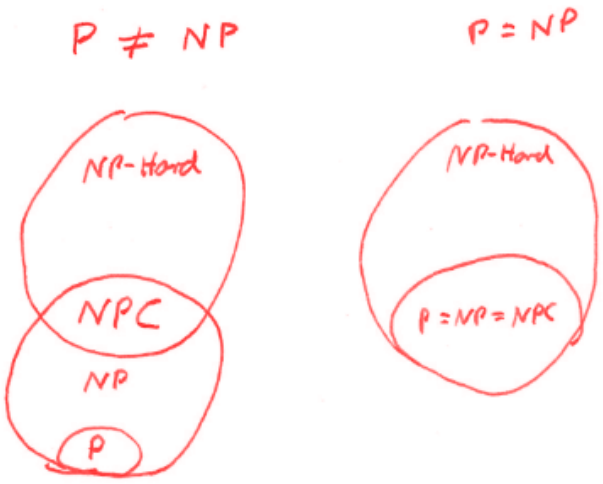
- may assume A and R_i for each $i \in 1..\ell$
- must establish R.

Question 2. Various Algorithms

(a) [10 marks] Pick your favourite lock-free data structure and describe how it works despite the lack of locks etc. Use diagrams as appropriate. See description in CACM article and make sure to include CAS operation and what it does.

not covered

(b) [5 marks] Draw two Venn Diagrams (showing how sets overlap or intersect) representing the sets of P, NP, NP-Hard, and NPC problems. One diagram should assume P = NP and the other diagram should assume P 6= NP. https://en.wikipedia.org/wiki/NP-hardness



(c) [5 marks] What is the difference between Monte Carlo and Las Vegas probabilistic algorithms? See relevant lecture sl

~~Monte Carlo - Probably Correct~~

~~Las Vegas - Probably Fast~~

Monte Carlo - May not be correct, always gives an answer (MC - Maybe Not Correct)

Las Vegas - Always correct, may not give an answer

Question 3. Dynamic Programming

Suppose you're running a lightweight consulting business — just you, two associates, and some rented equipment. Your clients are distributed between Hawkes Bay and New Plymouth, and this leads to the following question.

Each month, you can either run your business from an office in Hawkes Bay (HB) or from an office in New Plymouth (NP). In month i, you'll incur an *operating cost* of H_i if you run the business out of HB; you'll incur an operating cost of N_i if you run the business out of NP. The costs depends on the distribution of client demands for that month.

However, if you run the business out of one city in month i, and then out of the other city in month $i+1$, then you incur a fixed *moving cost* of M to switch base offices.

Given a sequence of n months, a *plan* is a sequence of n locations — each one equal to either HB or NP — such that the i^{th} location indicates the city in which you will be based in the i^{th} month. The *cost* of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

The problem. Given a value for the moving cost M, and sequences of operating costs $H_1, \ldots, H_n$ and $N_1, \ldots, N_n$, find a plan of minimum cost. (Such a plan will be called *optimal*.)

Example. Suppose $n = 4, M = 10$, and the operating costs are given by the following table.

	Month 1	Month 2	Month 3	Month 4
H_i	1	3	20	30
N_i	50	20	2	4

Then the plan of minimum cost would be the sequence of locations $[HB, HB, NP, NP]$, with a total cost of $1 + 3 + 2 + 4 + 10 = 20$, where the final term of 10 is the moving cost of changing locations once.

(a) [5 marks] Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer. *Assume* that *n* = 4 and *M* = 10 as it was in the *example above*.

Give the optimal solution and its cost, as well as what the algorithm finds.

```
for i = 1 to n
      if Hi < Ni
            then output " HB in Month i "
      else
            output " NP in Month i "
end
```

Check to see if I've correctly interpreted:
Instance (I've created this):

Month	1	2	3	4
Hi	25	25	30	25
Ni	30	30	25	30

Algorithm output:
Hb month 1, hb month 2, np month 3, hb month 4
(total cost = 25 + 25 + 25 + 25 + (2*10) = 120)
Correct solution:
Hb month 1, hb month 2, hb month 3, hb month 4
(total cost = 25 + 25 + 25 + 30 +(0*10) = 105)

The provided algorithm is incorrect in this instance as it factors in only savings in month 3, but not the additional location change cost

(b) [5 marks] Give an example of an instance (again with n = 4 and M = 10) in which the optimal plan must move (i.e., change locations) at least three times. *Provide a brief explanation, saying why your example has this property*.
Check to see if I've correctly interpreted:
-Instance (I've created this):

Month	1	2	3	4
Hi	20	25	5	25 (total staying for 4 months straight is 75)
Ni	35	5	25	5 (as above, total = 70)

Optimal Output: hb month 1 np month 2 hb month 3 np month 4
Cost = 20+5+5+5 + (3*10) = 65

It works because moving from city 1 to city 2 suggests that city 2 is at least cost 10 cheaper than city 1. This is the only reason why a move would be justifiable -- if city 2 is 5 cheaper than city 1 we would no move as we still incur cost 10 to move.

In my example this occurs 3 times and we have also considered the cost of staying in a city for 4 months straight, and it holds that moving 3 times is the cheapest option.

(c) [20 marks] Give a pseudocode for an efficient algorithm that takes values for n, M, and sequences of operating costs H1, . . . , Hn and N1, . . . , Nn, and returns the cost of an optimal plan.

Hint: What is the table of intermediate results with optimal substructure property?

~~Obvious if you just think about it algorithm~~

Let the optimal sequence of locations for n months be $O_n = (o_1 \dots o_n), o_i \in \{H, N\}$

Recurrence relation. Plans can either end in HB or NP, so:

$$Opt(n) = \min\left(Opt_{HB}(n), Opt_{NP(n)}\right)$$

$$Opt_{HB}(n) = \min(Opt_{HB}(n-1) + H_n, Opt_{NP}(n-1) + N_n + M)$$

$$Opt_{NP}(n) = \min(Opt_{HB}(n-1) + H_n + M, Opt_{NP}(n-1) + N_n)$$

Question 4. Approximation Algorithms

Suppose you are acting as a consultant for the Port Authority of a small Pacific Rim nation. Their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here is a basic sort of problem they face. A ship arrives, with n containers of weight w1, . . . , wn. Standing on the dock is a set of trucks, each of which can hold K units of weight. (You can assume that K and each wi is an integer.) You can stack multiple containers in each truck, subject to the weight restriction of K; the goal is to minimise the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers 1, 2, 3, . . . into it until you get to a container that would overflow the weight limit. Now declare this truck “loaded” and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks. (a) [5 marks] Give an example of a set of weights, and a value of K, where this algorithm does not use the minimum possible number of trucks.

(a) [5 marks] Give an example of a set of weights, and a value of K, where this algorithm does not use the minimum possible number of trucks.

weights: 1, 2, 1
K = 2

THese will fit into two trucks, but the algorithm will use 3

(b) [15 marks] Shown that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

Suppose the number of trucks used by our greedy algorithm = m.
Divide the trucks used into consecutive groups of two, for a total of m/2 groups. In each group but the last, the total weight of containers must be strictly greater than K (else, the second truck in the group would not have been started).

So if the optimal solution uses O = totalweight/K trucks
then m <= 2*totalweight/K

******** different explanation by someone else below*******

Optimal solution will have:
O, number of optimal trucks = total weight / K
in a worst case scenario (non optimal) we can have up to 2*O trucks
want to prove it is bounded by 2*O using the algorithm

each truck, on average will be at least half full using the algorithm
take a sequence a,b,c,d,e,f,g

if any two consecutive containers, ie a and b, have the property a+b > k
then storing a and b will require at least two trucks, however
as a+b > K then the two trucks will, on average, have greater than 0.5k weight
this holds for any pair of consecutive containers

if a+b <= k then they can fit in one truck and we move on with the algorithm, now considering the pair (a,b) as ONLY ONE container, a*, and the next container in the sequence ie c, as the SECOND container, b*. We will eventually reach a point where a*+b*> K

thus we have T, number of trucks using algorithm, <= totalweight/0.5k
which is certainly not more than double that of totalweight/k

thus the number of trucks used by the algorithm is within a factor of 2.

Question 5. Chips but No Fish

(a) Professor Dale has n supposedly identical integrated-circuit chips that in principle are capable of testing each other. The professor's test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the professor cannot trust the answer of a bad chip. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

(i) [10 marks] Show that if you know that *all the chips* potentially can be bad (but you do not know how many are good or bad), no matter what algorithm the professor will come up with that claims to correctly identify the good chips, there is going to be a way to get the bad chips to conspire to behave in such a way that the algorithm would not work (i.e. not be able to correctly identify the good chips). Explain clearly how you can make the chips behave to defeat any such algorithm.

The strategy for the bad chips is to always say that other bad chips are good and other good chips are bad. This mirrors the strategy used by the good chips, and so, it would be impossible to distinguish

(ii) [10 marks] Consider the problem of finding a single good chip from among n chips, assuming that more than n/2 of the chips are good (or in other words definitely less than half of the chips are bad in any given input). Show that bn/2c pairwise tests are sufficient to reduce the problem to one of less than half the size.

Hint 1: If you put two chips on the jig and the result says that one of them is bad, what would happen if you throw both of the chips away?

Hint 2: If you have a set of pairs of chips each of which is either both good or both bad (you don't necessarily know which though), and you know that the majority of such pairs are both good, then how can you come up with a smaller set of chips with the same property of having majority being good?

You should be able to use the two hints above to come up with an algorithm.

Arbitrarily pair up the chips. Look only at the pairs for which both chips said the other was good. Since we have at least half of the chips being good, we know that there will be at least one such pair which claims the other is good. We also know that at least half of the pairs which claim both are good are actually good. Then, just arbitrarily pick a chip from each pair and let these be the chips that make up the sub-instance of the problem

(iii) [10 marks] Show that the good chips can be identified with Θ(n) pairwise tests, assuming that more than n/2 of the chips are good. Give and solve the recurrence that describes the number of tests.

Once we have identified a single good chip, we can just use it to query every other chip.
Recurrence: $T(n) <= T(n/2) + n/2$
So $T(n) \varepsilon \theta(n)$

2016 Exam

http://ecs.victoria.ac.nz/foswiki/pub/Main/ExamArchiveCOMP361/exam2016-partial-solution.pdf

Questions	Marks
1. Divide and Conquer	[30]
2. Greedy Algorithms	[30]
3. Dynamic Programming	[30]
4. Guest Lectures	[10]
5. Approximation Algorithms (hard)	[20]

Question 1. Divide and Conquer

(a) (10 marks) How many lines, as a function of n (in Θ() form), does the following program print? You may assume n is a power of 2.

Hint: Write a recurrence that represents the number of lines printed and solve it.

```
function f ( n )
     if n > 1:
          print_line ( ‘ ‘ still going ’ ’)
          f ( n /2)
          f ( n /2)
```

T(n) =
1 | if n <= 1
2T(n/2) + 1 | if n > 1

Trivial application of master theorem, (or just intuition) gives theta n.

(b) (10 marks) You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time O(n logn).

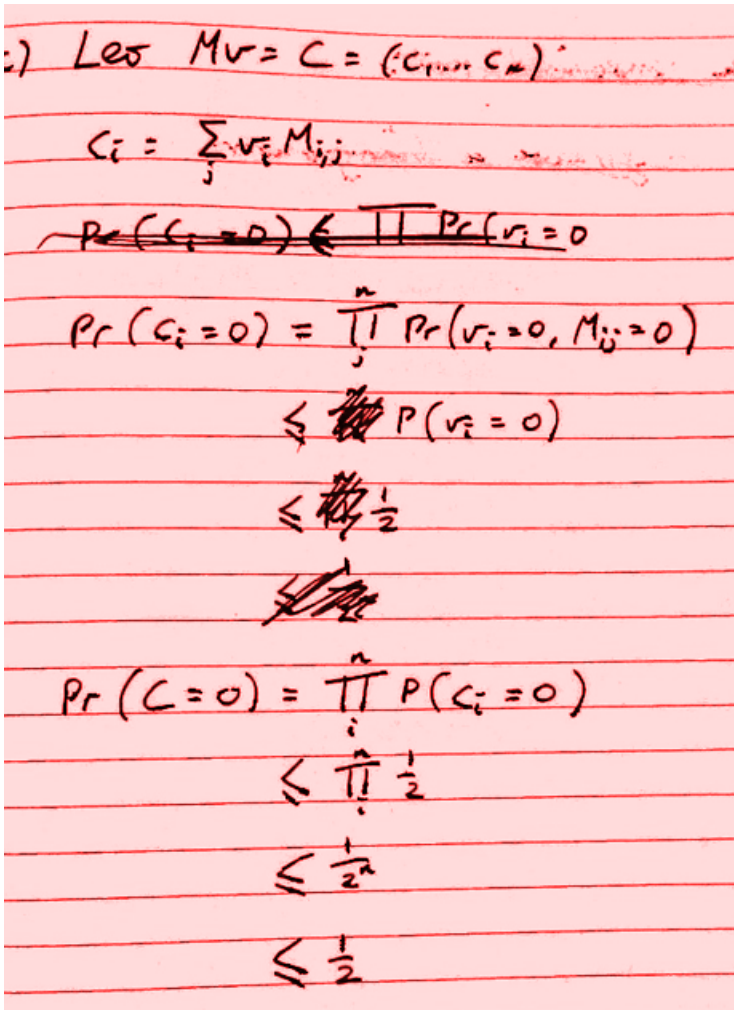
Most simple method, just sort (O(n logn)) and remove duplicates (O(n) or O(1) if you get fancy by doing it while sorting).

Overall the combination of the sorting and reduction equates to O(nlogn) + O(n). Lower order terms are dropped as we consider the asymptotic bounds. Therefore the proposed solution is O(n logn)

(c) (10 marks) (Hard) Suppose you are given matrices A, B, C which are each n by n and you wish to check whether AB = C. You can do this in $O(n^{\log_2 7})$ steps using Strassen's algorithm. In this subquestion we will explore a much faster, $O(n^2)$ randomised test.

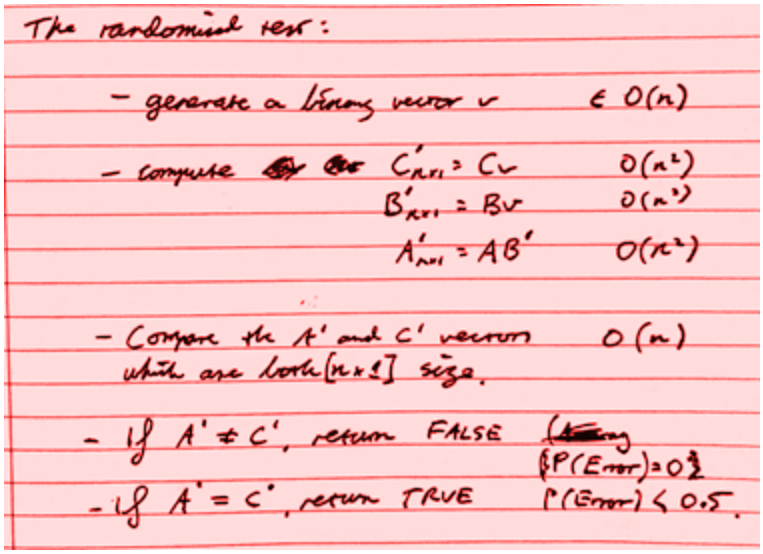
> **(i)** Let en to be 0 or 1 (each with probability 1/2). Prove that if M is a non-zero n by n matrix (i.ev be an n-dimensional vector whose entries are randomly and independently chosen. at least one element is not a zero), then Pr[Mv = 0] ≤ 1/2.

~~Prove by counting argument, show 1-1 correspondence between v's that produce Mv = 0 and Mv /= 0 by transforming between such v's.~~



(ii) Show that Pr[ABv = Cv] ≤ 1/2 if AB /= C. Why does this give an O(n 2) randomised test for checking whether AB = C?

ABv = Cv => ABv - Cv = 0, let M = AB - C => Mv = 0, from above Pr[Mv = 0] ≤ ½.



Computations required: M = AB - C (O(n^2)), Mv = 0 (O(n^2)). You can probably skip the M = AB - C calculation and just apply it to ABv = Cv, since the reverse of the above holds too? Haven't thought about it.

https://www.coursehero.com/file/p1r1sg/d-We-can-save-some-time-by-sorting-the-points-by-y-coordinate-only-once-and/

Question 2. Greedy Algorithms

Here's a problem that occurs in automatic program analysis. For a set of variables x1, . . . , xn, you are given some equality constraints, of the form "xi = xj" and some disequality constraints, of the form "xi 6= xj ." Is it possible to satisfy all of them?

For instance, the constraints

$$x1 = x2,\ x2 = x3,\ x3 = x4,\ x1\ 6=\ x4$$

cannot be satisfied.

(a) (15 marks) Give an efficient algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied.

Hint: One possible option is to consider a graph representation of this problem where each node is a variable (e.g. xi) and an edge represents an “equality constraint”.

~~WTF~~
~~Each variable is a node, edges are equality.~~
~~Iterate over all disequalities, from LHS traverse graph, if reach RHS then contradiction.~~

~~Just use a modified version of kruskal's algorithm~~

create graph representation with nodes being
variables and edges being equality constraints between them.

Do breadth (or depth) first search starting from node representing LHS
variable of disequality **(note: check that both nodes are in the graph first!)**. If we see RHS variable it is not satisfied
otherwise it is satisfied.

(b) (10 marks) Argue (or do a proof sketch) that your algorithm is correct.

If there is an equality between any two nodes x_a and x_z, either directly or via x_b, x_c, x_d etc, then there will be an edge between them. So x_a and x_z will be connected, and the BFS will find one from the other, indicating that the inequality cannot hold.

(c) (5 marks) State the asymptotic cost of your algorithm and justify why it’s correct.

~~O(n), i’ll do it in the morning (or some other time).~~

Add vertices (variables): O(n)
Add edges (equalities): O(m)
DFS from each vertex: O(n + m)
Total number of DFS’s: m

So: $O(mn + m^2)$

Question 3. Dynamic Programming

You are given a string of n characters: s[1 . . . n], which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like

“itwasthebestoftimes...”). You wish to reconstruct the original document using a dictionary, which is available in the form of a Boolean function dict(). For any string w,

$$\texttt{dict}(w) = \begin{cases} \texttt{true} & \text{if } w \text{ is a valid word,} \\ \texttt{false} & \text{otherwise.} \end{cases}$$

(a) (20 marks) Give a dynamic programming algorithm that determines whether the string s[] can be reconstituted as a sequence of valid words. The running time should be at most O(n 2), assuming calls to dict take unit time.

```
int[] dp = new int[s.len];
dp.fill(-1)
dp[0] = 0;
String res;

for(int i = 0; i < s.len; ++i) {
   if(dp[i] != -1) {
      for(int j = i+1; j <= s.len; ++j) {
         int len = j-i;
         string substr = str.substr(i, len);
         if(dict.has(substr)) {
            res += (i ? " " : "") + substr;
            dp[j] = dp[i]+1;
         }
      }
   }
}
hasSolution = dp[dp.len-1] >= 0;
return res;
```

(b) (10 marks) In the event that the string is valid, modify your algorithm to output the corresponding sequence of words.

Question 4. Guest Lectures

(a) (5 marks) State what the Tutte Polynomials are used for as presented in David's guest lecture.

(b) (5 marks) Outline the main steps of the JPEG Encoding Algorithm as discussed in Neil's guest lecture.

Question 5. Approximation Algorithms (Hard)

Recall the knapsack problem from the lectures and assignment 3. There are n items, where the ith item is worth vi dollars and weighs wi grams. We are also given a knapsack that can hold at most W grams. Here, we add the further assumptions that each weight wi is at most W and that the items are indexed in monotonically decreasing order of their values: $v1 \geq v2 \geq \ldots \geq vn$.

In the 0-1 knapsack problem, we wish to find a subset of the items whose total weight is at most W and whose total value is maximum. The fractional knapsack problem is like the 0-1 knapsack problem, except that we are allowed to take a fraction of each item, rather than being restricted to taking either all or none of each item. If we take a fraction xi of item i, where $0 \leq xi \leq 1$, we contribute xiwi to the weight of the knapsack and receive value xivi . Our goal is to develop a polynomial-time approximation algorithm for the 0-1 knapsack problem.

In order to design a polynomial-time algorithm, we consider restricted instances of the 0-1 knapsack problem. Given an instance I of the knapsack problem, we form restricted instances Ij , for $j = 1, 2, \ldots, n$, by removing items $1, 2, \ldots, j - 1$ and requiring the solution to include item j (all of item j in both the fractional and 0-1 knapsack problems). No items are removed in instance I1. For instance Ij , let Pj denote an optimal solution to the 0-1 problem and Qj denote an optimal solution to the fractional problem.

(a) (4 marks) Argue that an optimal solution to instance I of the 0-1 knapsack problem is one of $\{P1, P2, \ldots, Pn\}$.

(b) (4 marks) Prove that we can find an optimal solution Qj to the fractional problem for instance Ij by including item j and then using the greedy algorithm in which at each step, we take as

much as possible of the unchosen item in the set {j + 1, j + 2, . . . , n} with maximum value per gram vi/wi .

(c) (4 marks) Prove that we can always construct an optimal solution Qk to the fractional problem for instance Ij that includes at most one item fractionally. That is, for all items except possibly one, we either include all of the item or none of the item in the knapsack.

(d) (4 marks) Given an optimal solution Qj to the fractional problem for instance Ij , form solution Rj from Qj by deleting any fractional items from Qj . Let v(S) denote the total value of items taken in a solution S. Prove that $v(Rj) \geq v(Qj)/2 \geq v(Pj)/2$.

(e) (4 marks) Give a polynomial-time algorithm that returns a maximum-value solution from the set {R1, R2, . . . , Rn} , and prove that your algorithm is a polynomial-time approximation algorithm for the 0-1 knapsack problem.