COMP 3711/3711H
Collected Tutorial Questions

Version of February 20, 2021

M. J. Golin – HKUST

This document contains a collection of problems, many of which will be covered in the COMP3711 or COMP3771H tutorials.

Please check the tutorial web page to see which problems are assigned to which tutorial session.

We strongly recommend that you try solving the problems yourself BEFORE reading the answers or listening to the TA's presentation.

Note: Be aware that COMP 3711 and COMP3711H teach a very small number of topics slightly differently, e. g., the analysis of randomized Quicksort and the mathematical formulation for Max-Flow. A few tutorial problems on these subjects are oriented towards one or the other of the two classses.

The problems are roughly divided into the following topics

(SS) Sorting and Searching

(R) Randomization

(DC) Divide & Conquer

(GY) Greedy

(GR) Graphs

(DP) Dynamic Programming

(MM) Max Flow & Matchings

(HA) Hashing

Sorting and Searching (SS)

(SS1) $a_1, a_2, \ldots, a_n$ is a sequence that has the following property:
There exists some k such that

$$\forall\ i\ :\ 1 \le i < k, \quad a_i > a_{i+1} \qquad \forall\ i\ :\ k \le i < n, \quad a_i < a_{i+1}.$$

Such a sequence is *unimodal* with unique minimum a_k.
Design an $O(\log n)$ algorithm for finding k.

(SS2) You are given an (implicit) infinite array $A[1, 2, 3.....]$.
and are told that, for some unknown n, the first n items in the array are positive integers sorted in increasing order while, for $i > n$, $A[i] = \infty$.

Give an $O(\log n)$ algorithm for finding the largest non-∞ value in A.

(SS3) Consider the heap implementation of a Priority Queue shown in class that keeps its items in an Array $A[]$.

Let Decrease-Key(i, x) be the operation that compares x to $A[i]$:

If $x \ge A[i]$ it does nothing.
If $x < A[i]$, it sets $A[i] = x$ and, if necessary, fixes $A[]$ so that it remains a Heap.

Show how to implement Decrease-Key(i, x) in $O(\log n)$ time, where n is the number of items in the Heap.

Note: We will use the operation Decrease-Key(i, x) later in the semester.

(SS4) (a) Illustrate how Mergesort would work on input [1,2,3,4,5,6,7,8,9].

(b) Illustrate how Mergesort would work on input [9,8,7,6,5,4,3,2,1].

(c) Illustrate how Quicksort would work on input [1,2,3,4,5,6,7,8].
Assume that the last item in the subarray is always chosen as the pivot.

(d) Illustrate how Quicksort would work on input [8,7,6,5,4,3,2,1].
Assume that the last item in the subarray is always chosen as the pivot.

(SS5) Your input is k sorted lists that need to be merged into one sorted list. The "obvious" solution is to modify the merging procedure from mergesort; at every step, compare the smallest items from each list and move the minimum one among them to the sorted list.

Finding the minimum value requires $O(k)$ time so, if the lists contain n items in *total* the full k-way merge would take $O(nk)$ time.

This can be solved faster.

Design an $O(n \log k)$-time algorithm to merge k sorted lists into one sorted list by making use of priority queues.

Note that each sorted list may contain a different number of elements.

(SS6) The following is the pseudo-code for a procedure known as *BubbleSort* for sorting an array of n integers in ascending order.

```
repeat
    swapped := false;
          for i = 1 to n-1
               if A[i] > A[i+1]  then
                  swap A[i] and A[i+1]
                  swapped := true
until  not(swapped)
```

(a) Prove that Bubble sort correctly sorts its input.

(b) What is the worst-case input for bubble sort? Use it to derive a lower bound on the time complexity of Bubblesort in the worst case.

(SS7) An array $A[1 \dots n]$ is circularly sorted if there exists some $k \in [1 \dots n]$ such that $A[k \dots n]$ concatenated to $A[1 \dots k-1]$ is sorted. As an example

$$A = [30, 40, 55, 10, 18, 24, 27, 28]$$

is circularly sorted with $k = 4$.

How can you modify the binary search algorithm to search for an item in A in $O(\log n)$ time.

(SS8) **Heapify**
In class, we learned how to maintain a min-heap implicitly in an array.
Given that $A[i \dots (j-1)]$ represents an implicit min-heap, we saw how to add $A[j]$ to the min-heap , in $O(\log j)$ time.
This led to an $O(n \log n)$ algorithm for constructing a min-heap from array $A[1, \dots, n]$.

For this problem show how to construct a min-heap from an array $A[1 \dots n]$ in $O(n)$ time.

It might help to visualize the min-heap as a binary tree and not an array.

For simplification, you may assume that $n = 2^{k+1} - 1$ for some k, i.e., the tree is complete.

Hint: Consider "heapifying" the nodes processing them from bottom to top.

(SS9) There are n items in an array. It is easy to see that that their minimum can be found using $n-1$ comparisons and that $n-1$ are actually required. It is also easy to see that finding the max can similarly be done using $n-1$ comparisons with $n-1$ required.

Design an algorithm that finds *both* the minimum and the maximum using at most $\frac{3}{2}n + c$ comparisons, where $c > 0$ can be any constant you want.

Note: Although it is harder to prove, $\frac{3}{2}n + c$ comparisons is actually a lower bound.

(SS10) You are given an input containing n/k lists:
(i) Each list has size k, and
(ii) for $i = 1$ to n/k, the elements in list $i-1$ are all less than all the elements in list i

The simple algorithm to fully sort these items is to sort each list separately and then concatenate the sorted lists together. This uses $\frac{n}{k}O(k \log k) = O(n \log k)$ comparisons.

Show that this is the best possible. That is, show that any comparison-based sorting algorithm to sort the n/k lists into one sorted list with n elements will need to make at least $\Omega(n \log k)$ comparisons.

Note that you can not derive this lower bound by simply combining the lower bounds for the individual lists.

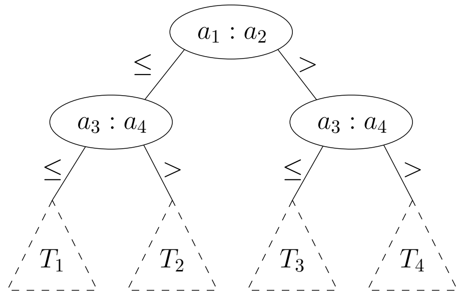
(SS11) **Prove that insertion in a binary search tree requires at least $\Omega(\log n)$ comparisons (in the worst case) per insertion, where n is the number of items in the search tree.**

Hint: What lower bounds have we learned in class? Suppose you built the search tree using insertions. What can you do with it?

(SS12) **Build a Binary Search Tree for the items**
8, 4, 6, 13, 3, 9, 11, 2, 1, 12, 10, 5, 7
and draw the final tree.
Now, delete 3, 9, 4 in order and draw the resulting trees.

(SS13) The figure below shows part of the decision tree for mergesort operating on a list of 4 numbers, a_1, a_2, a_3, a_4.

Expand subtree T_3, i.e., show all the internal (comparison) nodes and leaves in subtree T_3.



(SS14) The maximum item in a set of n real-valued keys is well defined. The maximum item in a set of n 2-dimensional real-valued points is not.

One definition that is used in database theory is that of *skyline vectors*. These are also known as *maximal points* or *maximal vectors*.

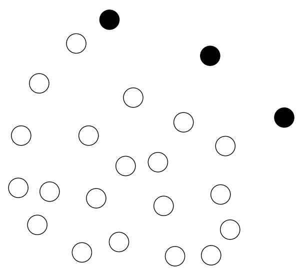
Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of 2-d points where $p_i = (x_i, y_i)$. A point $p \in S$ is a *skyline vector* if no other point is bigger than it in both x and y dimensions.

Formally p_j *dominates* p_i if

$$x_i < x_j \quad \text{and} \quad y_i < y_j.$$

$p = (x, y)$ *is a skyline vector in* S if no p_i in S dominates p.

In the example below, the 3 filled points are the skyline ones.



(a) Give an algorithm that finds the skyline vectors in a set S of n points in $O(n \log n)$ time.

(b) Suppose that the points all have integer coordinates in the range $[1, \ldots, n^2]$. Give an $O(n)$ algorithm for solving the same problem.

(SS15) **AVL Trees**

(a) Construct an AVL tree by inserting the items 134625 in that order.
Next construct another AVL tree on those items by inserting in the order 123456.
Do they have the same height?
Now construct an AVL tree by inserting the items 5362471 in that order and another by inserting 4261357 in that order. Do those two trees have the same height?

(b) What are the minimum and maximum heights for an AVL tree with 88 nodes labelled $1, 2, 3 \ldots, 88$?

(SS16) **Sorting Polynomially Bounded Integers**

In this problem, you are given n integers to sort.

(a) You are told they are all in the range $[0, n^2 - 1]$. How fast can you sort them?

(b) Now you are told they are all in the range $[0, n^t - 1]$ for some fixed t. How fast can you sort them?

(SS17) **Lower Bound on the EPFL of Binary Trees**

The proof (in COMP3711H) that comparison-based sorting algorithms require $\Omega(n \log n)$ comparisons *on average* used the fact that the External Path Length of a binary tree with n leaves is at least $n \log_2 n + O(n)$. Prove this fact.

(SS*1) Extra Problem. *The limits of comparison-based lower bounds*

The purpose of this problem is to illustrate that lower bounds in comparison-based models can completely fail in other models.

Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of integers or real numbers. Let $y_1, y_2, \ldots, y_n$ be the same numbers sorted in increasing order. The *MAX-GAP* of the original set is the value

$$\texttt{Max}_{1 \le i < n}(y_{i+1} - y_i).$$

As an example, if $S = \{3, 12, 16, 7, 13, 1\}$, sorting the items gives $1, 3, 7, 12, 13, 16$ and the MAX-GAP is $12 - 7 = 5$.

Using a more advanced form of the $\Theta(n \log n)$ proof of the lower bound for sorting it can be proven that calculating MAX-GAP requires $\Theta(n \log n)$ operations if only comparisons and algebraic calculations are used. In this problem, we will see that, if the floor (truncate) operator $\lfloor x \rfloor$ can also be used, the problem can be solved using only $O(n)$ operations!

Before reading the solution below, try to see if you can solve it yourself!

- Find y_1 and y_n, the minimum and maximum values in S.
- Let $\Delta = \frac{y_n - y_1}{n-1}$. Let B_i be the half-closed half-open interval defined by

  $$B_i = \Big[y_1 + (i-1)\Delta,\, y_1 + i\Delta\Big)$$

  for $i = 1, 2, \ldots n - 1$ and set $B_n = \{y_n\}$.
- Prune S as follows. For every B_i throw away all items in $S \cap B_i$ except for the smallest and largest. Let S' be the remaining set.
- Find the Max-Gap of S' by sorting S' and running through the items in S' in sorted order. Output this value

Prove that this algorithm outputs the correct answer and show that every step can be implemented in $O(n)$ time (the 3rd step might require the use of the floor function).

(SS*2) **AVL Trees**

Given any specific insertion order on n keys the output is a specific AVL tree. Recall that a tree T is an AVL tree if it satisfies the *local* balance condition at every node. This doesn't a-priori imply that every possible AVL tree can be constructed via insertions.

Suppose T is some tree on n keys that satisfies the AVL balance condition.

Is there always an insertion order that of the keys that builds T? If yes, show one; if no, show a counterexample.

Randomization (R)

(R1) Consider the Hire-Assistant algorithm described in the lecture notes.

Assume that the candidates are presented in a random order.
The analysis in the lecture notes calculated the *Expected* number of hires. For this problem calculate:

(a) the probability that you hire exactly one person.

(b) the probability that you hire exactly n people.

(R2) Use indicator random variables to solve the following, known as the **hat-check problem**.

Each of n customers gives a hat to a hat-check person at a restaurant.

The hat-check person gives the hats back to the customers in a random order.

What is the expected number of customers who get back their own hat?

Note: Replacing hats with homeworks and customers with students gives the following equivalent question: suppose that there are n students in a class who have just submitted their homework. The teacher gives the homeworks back to the students in a random order and asks the students to mark the homework they have been handed. What is the expected number of students who have been asked to mark their own homework?

(R3) **Another Analysis of Randomized Selection** We learned the *Randomized Selection* algorithm and used a geometric series analysis approach to show that it runs in $O(n)$ expected time.

In this problem you will rederive the $O(n)$ time in a different way, using the *Indicator Random Variable* method used to analyze Quicksort in the COMP3711 lectures.

Recall the *Randomized Selection* algorithm to find the **k-th smallest element**.

Pick a random pivot, divide the array into 3 parts:

left subarray, pivot item, right subarray.

We then either stop immediately or recursively solve the problem in the left **OR** the right part of the array.

- As in quicksort, denote the elements in **sorted order** by $z_1, \ldots, z_n$ (so we are searching for z_k)
- We use the same random model for choosing the pivot as for quicksort.

I) Define

$$X_{ij} = \begin{cases} 1 & \text{if } z_i \text{ and } z_j \text{ are compared by the algorithm} \\ 0 & \text{otherwise} \end{cases}$$

Prove the following three facts

(a) $i \le k \le j$: $\Pr[X_{ij} = 1] = 2/(j - i + 1)$.

(b) $i < j < k$: $\Pr[X_{ij} = 1] = 2/(k - i + 1)$.

(c) $k < i < j$: $\Pr[X_{ij} = 1] = 2/(j - k + 1)$.

(II) By the indicator random variable technique, the expected total number of comparisons is

$$\sum_{i<j} E[X_{ij}] = \sum_{i<j} \Pr[X_{ij} = 1]$$

Use this to show that $\sum_{i<j} E[X_{ij}] = O(n)$.

(R4) **Quicksort with repeated elements**

The discussion of the expected running time of randomized quicksort in the COMP3711 lecture notes assumed that all element values are distinct. In this problem, we examine what happens when they are not.

(a) Suppose that all element values are equal. What would randomized quicksort's running time be ?

(b) The PARTITION procedure taught returns an index q such that each element of $A[p...q-1]$ is less than or equal to $A[q]$ and each element of $A[q+1...r]$ is greater than $A[q]$.

Modify PARTITION to produce a new procedure PARTITION$'(A, p, r)$, which permutes the elements of $A[p...r]$ and returns two indices q, t, where $p \leq q \leq t \leq r$, such that

- all elements of $A[q...t]$ are equal,
- each element of $A[p...q-1]$ is less than $A[q]$, and
- each element of $A[t+1...r]$ is greater than $A[q]$

Like PARTITION, your PARTITION$'$ procedure should take $\Theta(r-p)$ time.

(c) Modify the QUICKSORT procedure to produce QUICKSORT$'(A, p, r)$ that calls PARITITION$'$ but then only recurses on $A[p, q-1]$ and $A[t+1, r]$.

Problem (R*3) will discuss how to analyze this new algorithm

(R5) Let $A[1..n]$ be an array of n distinct numbers.
In class we said that if $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an **inversion** of A.

Suppose that the elements of A form a uniform random permutation of $\langle 1, 2, ..., n \rangle$.

Use indicator random variables to compute the expected number of inversions.

(R*1) Extra Problem. You are given n pairs of nuts and bolts

$$(N_1, B_1), (N_2, B_2), \ldots, (N_n, B_n).$$

Each pair is a different size than the others. Someone has unscrewed all of the nuts off of the bolts and mixed them up.

Problem: Match all nuts up with their corresponding bolts.

If we could separately
(i) sort all the bolts by increasing size and then
(ii) sort all the nuts by increasing thread size
$\Rightarrow$ problem would be easily solvable in $O(n \log n)$ time.

After sorting, just match them up in order from smallest to largest. The difficulty is that we can't do this because we can't compare the sizes of two nuts directly or the sizes of two bolts directly.

The only operation available is to try to screw a bolt B into a nut N and then, by seeing whether the bolt

- (a) goes loosely in,
- (b) perfectly fits or
- (c) can't go in at all,

decide whether their thread sizes satisfy

$$\text{(a) } B < N, \quad \text{(b) } B = N \text{ or} \quad \text{(c) } B > n.$$

Design an $O(n \log n)$ time randomized algorithm for matching nuts and bolts.

Hint: Try to modify Quicksort.

(R*2) Extra Problem. **Randomized Binary Search Trees**

- Consider a Binary search tree T on n keys.

  The *depth*, $d(v)$, of v in T is the length of the path from the root of T to v. Note that the depth of the root is 0. The *Path Length of* T, $PL(T)$, is the sum of the depths of all of the nodes of T; $PL(T) = \sum_{v \in T} d(v)$.

  Note that $\frac{1}{n} PL(T)$ is the average depth of a node in the tree. This is also the average time to search for a randomly chosen node in the tree.
- Suppose that every key K_i in a set of n keys has real weight w_i associated with it, with the weights being unique.
- There is a unique binary search tree that can be built on the n keys that also satisfies min-heap order by the weights (Why?).
- Suppose n weights $w_1, w_2, \ldots, w_n$ are chosen independently at random from the unit interval $[0, 1]$ and then sorted. The resulting order is a random permutation of the n items.

A *Treap* or *Randomized Binary Search Tree* on n keys K_i is constructed by choosing n weights w_i independently at random from the unit interval $[0, 1]$ and associating w_i with K_i. The Treap is the unique BST built on the n keys that also satisfies min-heap order on the weights.

(a) Describe how to build T in time $O(n \log n + PL(T))$

(b) If T is the Treap built, prove that the average value of $PL(T)$ is $O(n \log n)$
Hint: consider quicksort

(R*3) **Analyzing Modified Quicksort from Problem (R4)**

Our analysis of QUICKSORT in class assumed that all elements were distinct.

Problem (R4) developed a more sophisticated version of Quicksort that handles repeated items more efficiently.

How would you adjust the analysis (binary tree plus indicator random variables) in the lecture notes to avoid the assumption that all elements are distinct and prove that QUICKSORT$'$ runs in $O(n)$ time.

(R*4) **Recursive Independence in the Probabilistic Analysis of Quicksort**

The probabilistic analysis of Quicksort (in COMP3711H) used the fact that if $\mathbf{A}$ is the set of items in subarray $A[p \ldots r]$ and π a random permutation of $\mathbf{A}$ then

(a) $A[r]$ is equally likely to be any item in $\mathbf{A}$

(b) After running the partition algorithm on $A[p \ldots r]$, the input to the new left and right subproblems are again random permutations.

Prove the correctness of this fact.

Divide & Conquer (DC)

(DC1) Derive asymptotic upper bounds for $T(n)$. Make your bounds as tight as possible.

(a) You may assume n is a power of 2.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + n \qquad \text{if } n > 1 \end{aligned}$$

(b)

$$\begin{aligned} T(1) &= T(2) = 1 \\ T(n) &= T(n-2) + 1 \qquad \text{if } n > 2 \end{aligned}$$

(c) You may assume n is a power of 3.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/3) + n \qquad \text{if } n > 1 \end{aligned}$$

(d) You may assume n is a power of 2.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(n/2) + n \qquad \text{if } n > 1 \end{aligned}$$

(e) You may assume n is a power of 2.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(n/2) + n^2 \qquad \text{if } n > 2 \end{aligned}$$

(f) You may assume n is a power of 2.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + \log_2 n \qquad \text{if } n > 1 \end{aligned}$$

(g) You may assume n is a power of 2.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(n/2) + \log_2 n \qquad \text{if } n > 1 \end{aligned}$$

(DC2) Using the *Master Theorem*, give asymptotic tight bounds for $T(n)$

(a)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(n/4) + n \qquad \text{if } n > 1 \end{aligned}$$

(b)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(n/4) + 1 \qquad \text{if } n > 1 \end{aligned}$$

(c)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(n/2) + n^2 \qquad \text{if } n > 1 \end{aligned}$$

(d)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(n/3) + n^2 \qquad \text{if } n > 1 \end{aligned}$$

(e)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 9T(n/3) + n^2 \qquad \text{if } n > 1 \end{aligned}$$

(f)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 10T(n/3) + n^3 \qquad \text{if } n > 1 \end{aligned}$$

(g)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 99T(n/10) + n^2 \qquad \text{if } n > 1 \end{aligned}$$

(h)

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 101T(n/10) + n^2 \qquad \text{if } n > 1 \end{aligned}$$

(DC3) **Using Black-box median algorithms (modified from CLRS)**
For this problem, you assume that you are given a black-box $O(n)$ time algorithm for finding the median ($\lceil n/2 \rceil$nd) item in a size n array. This means that you can call the algorithm and use its result but can't peer inside of it.

(a) Show how *Quicksort* can be modified to run in $O(n \log n)$ *worst case* time.

(b) Give a simple linear-time algorithm that solves the selection problem for an arbitrary order statistic. That is, given k, your algorithm should find the k'th smallest item.

(c) For n distinct elements $x_1, x_2, \ldots, x_n$ with associated positive weights $w_1, w_2, \ldots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the **weighted (lower) medium** is the element x_k satisfying

$$\sum_{x_i < x_k} w_i < \frac{1}{2} \quad \text{and} \quad \sum_{x_i > x_k} w_i \leq \frac{1}{2}.$$

If the x_i are sorted, then it is easy to solve this problem in $O(n)$ time by just summing up the weights from left to right and walking through the sums until k is found. Show that if the items are *not* sorted you can still solve the problem in linear time using the black box median finding algorithm.

(DC4) **Polynomial Evaluation** The input to this problem is a set of $n+1$ coefficients $a_0, a_1, \ldots, a_n$. Define $A(x) = \sum_{i=0}^{n} a_i x_i$

(a) Given value x, how can you evaluate $A(x)$ using $O(n)$ multiplications and $O(n)$ additions?
Can you evaluate $A(x)$ using at most n multiplications and n additions?

(b) Now suppose that $A(x)$ has at most k non-zero terms. How can you evaluate $A(x)$ using $O(k \log n)$ operations.
Hint. How can you evaluate x^n using $O(\log n)$ operations.

(DC5) **Interpolating Polynomials** The values $A(x_0), A(x_1), \ldots, A(x_n)$, define a unique degree n polynomial having those values. The Langragian interpolation formula for finding the coefficients $a_0, a_1, \ldots, a_n$ of $A(x)$ works by first setting

$$I_i(x) = \prod_{0 \leq j \leq n,\, j \neq i} \frac{x - x_j}{x_i - x_j}$$

and then defining

$$A(x) = \sum_i A(x_i) I_i(x).$$

Show how to use the formula to evaluate the coefficients of $A(x)$ in $O(n^2)$ time.

Hints: Note that the $I_i(x)$ are very similar to each other. Instead of constructing them from scratch consider building their smallest common multiple $P(x)$ and then build $I_i(x)$ via division. First recall how long it takes to divide a degree n polynomial by a degree one polynomial. You can use this procedure as a subroutine.

(DC6) Prove from first principles that

$$\begin{aligned} T(n) &= 1 && \text{if } 1 \le n \le 8 \\ T(n) &= T\left(\left\lfloor \frac{n}{5} \right\rfloor + 1\right) + T(\left(\left\lceil \frac{3n}{4} \right\rceil + 2\right) + n && \text{if } n > 8 \end{aligned}$$

satisfies $T(n) = O(n)$.

Note: Before trying to solve this, review the slides describing the analysis of the deterministic selection problem. The analysis of this recurrence should follow the same technique used there.

(DC7) **More Median of Medians** For this problem you can assume the following fact: $\alpha, \beta \ge 0$, N is a non-negative integer and c, D constants (possibly negative). For $n > N$, if

$$T(n) \le T(\alpha n + c) + T(\beta n + d) + \Theta(n)$$

then

$$T(n) = \begin{cases} O(n) & \text{if } \alpha + \beta < 1 \\ O(n \log n) & \text{if } \alpha + \beta = 1 \\ \Omega(n \log n) & \text{if } \alpha + \beta > 1 \end{cases} .$$

Recall that our deterministic selection algorithm yielded the recurrence

$$T(n) = T(n/5) + T(7n/10 + 6) + \gamma n$$

for some constant γ. The formula above implies $T(n) = O(n)$.

Our algorithm (i) splits the items into sets of 5 elements, (ii) found the median of each set and then (iii) found x, the median of those medians. It then ran *partition* with x as a pivot and recursed on the appropriate subset. From the definition of x we were able to prove that the subarrays created by partition both had at most $7n/10 + 6$ elements, leading to the recurrence relation and hence $O(n)$ running time.

Now suppose that instead of splitting the items into sets of size 5, we split them into sets of size 3 and then ran the algorithm the same way. Would we still get an $O(n)$ time algorithm?

What about if we split into sets of size 7? Sets of size 9?

(DC8) **Divide and Conquer**

You have found a newspaper from the future that tells you the price of a stock over a period of n days next year. This is presented to you as an array $p[1 \dots n]$ where $p[i]$ is the price of the stock on day i.

Design an $O(n \log n)$-time divide-and-conquer algorithm that finds a strategy to make as much money as possible, i.e., it finds a pair i, j with $1 \le i < j \le n$, such that $p[j] - p[i]$ is maximized over all possible such pairs.

If there is no way to make money, i.e., $p[j] - p[i] \le 0$ for all pairs i, j with $1 \le i < j \le n$, your algorithm should return "no way".

(a) Describe your algorithm and explain why it gives the correct answer

(b) Analyze your algorithm to show that it runs in $O(n \log n)$ time.

Note: The purpose of this problem is to provide you practivce with D & C tools. There is also an $O(n)$ time algorithm for solving this problem. See if you can find it.

(DC9) **Modified from CLRS**

Professor Olay is consulting for an oil company, which is planning a large pipeline running east to west through an oil field of n wells. From each well, a spur pipeline is to be connected directly to the main pipeline along a shortest path (either north or south). Given the x, y coordinates of the wells, how should the professor pick the optimal location of the main pipeline (the one that minimizes the total length of the spurs). Show that the optimal location can be determined in linear time.

Hint: Try to solve this using the median finding algorithm.

(DC10) **Modified from CLRS** *Largest i numbers in sorted order.*

Given a set of n numbers, we wish to find the i largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time and analyze the running times of the algorithms in terms on n and i.

(a) Sort the numbers and list the i largest.

(b) Build a max-priority queue (i.e., a heap) from the numbers and call EXTRACT-MIN i times.

(c) Use a selection algorithm to find the ith largest number, partition around that number and sort the i largest numbers.

(DC11) **Finding a "fixed point".**

Input: a sorted array $A[1..n]$ of n **distinct** integers
(Distinct means that there are no repetitions among the integers. The integers can be positive, negative or both).

Design an $O(\log n)$ algorithm to return an **index i such that $A[i] = i$,** if such an i exists. Otherwise, report that no such index exists.

As an example, in the array below, the algorithm would return 4.

i	1	2	3	4	5	6	7	8
A[i]	-3	0	1	4	12	17	20	22

while in the array below this line, the algorithm would return that no such index exists.

i	1	2	3	4	5	6	7	8
A[i]	-3	0	1	7	12	17	20	22

Hint: $O(\log n)$ often denotes some type of binary search. Can you think of any question you might ask that permits you to throw away some constant fraction of the points?

(DC12) **The Majority Problem**

Let $A[1..n]$ be an array of n elements. A *majority element* of A is any element occurring more than $n/2$ times (e.g., if $n = 8$, then a majority element should occur at least 5 times). Your task is to design an algorithm that finds a majority element, or reports that no such element exists.

Suppose that you are not allowed to order the elements; the only way you can access the elements is to check whether two elements are equal or not.

Use standard divide-and-conquer techniques to design an $O(n \log n)$-time algorithm for this problem.

(DC13) **Lagrangian Interpolation Example**

(a) Use Lagrangian Interpolation to construct a degree-2 polynomial $A(x)$ satisfying

$$A(1) = 1, \quad A(2) = 4, \quad A(3) = 9.$$

(b) Use Lagrangian Interpolation to construct a degree-2 polynomial $A(x)$ satisfying

$$A(1) = 1, \quad A(2) = 8, \quad A(3) = 27.$$

(DC*1) **Two for the Price of One Convolutions**

Let $< a_i >$ denote the sequence of real numbers $a_0, a_1, \ldots, a_{n-1}$. The *convolution* of $< a_i >$ and $< b_i >$ both of length n is sequence $< c_i >$ of length $2n-1$ where $c_i = \sum_{j=1}^{i} a_j b^{i-j}$. In class we saw that the FFT could calculate $< c_i >$ in $O(n \log n)$ time.

Now suppose that you are given TWO sequences $< a_i' >$ and $< a_i'' >$ both of length n and $< b_i >$ of length n. Can you see a quick way of calculating the convolution of $< a_i' >$ and $< b_i >$ and the convolution of $< a_i'' >$ and $< b_i >$ without having to run the algorithm twice?

Note: It is not expected that you could solve this yourself without prior knowledge. It is a cute trick that is worth knowing about, though, and understanding it provides a better understanding of what the FFT is really doing.

(DC*2) **An $O(n)$ majority algorithm**

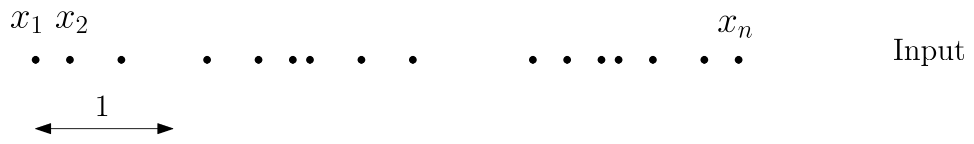
Consider the majority problem defined in problem DC12.

Design an $O(n)$ time algorithm for it.

Greedy (GY)

(GY1) **From CLRS** A Greedy Algorithm.

A *unit-length closed interval* on the real line is an interval $[x, 1+x]$. Describe an $O(n)$ algorithm that, given input set $X = \{x_1, x_2, \ldots, x_n\}$, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct. You should assume that $x_1 < x_2 < \cdots < x_n$.



As an example the points above are given on a line and you are given the length of a 1-unit interval. Show how to place a minimum number of such intervals to cover the points

(GY2) Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

(a) Describe a greedy algorithm to make change consisting of quarters (25 cents), dimes (10), nickels (5), and pennies (1). Prove that your algorithm yields an optimal solution.

(b) Suppose that the available coins are in denominations that are powers of c. i.e. the denominations are $c^0, c^1, ..., c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

(c) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

(GY3) (CLRS–16.2-4) Professor Midas drives an automobile from Newark to Reno along Interstate 80. His car's gas tank, when full, holds enough gas to travel m miles, and his map gives the distance between gas stations on his route. The professor wishes to make as few gas stops as possible along the way. Give an efficient method by which Professor Midas can determine at which gas stations he should stop and prove that your algorithm yields an optimal solution.

(GY4) Huffman Coding
From the book Problems on Algorithms, *by Ian Parberry, Prentice-Hall, 1995.*
Build a Huffman Tree on the frequencies $\{1, 3, 5, 7, 9, 11, 13\}$.
For this problem, follow the rule that if two items are combined in a merge, the smaller one goes to the left subtree (in case of ties *within* a merge you can arbitrarily decide whic goes on the left).
Are there any *ties* in the Huffman Construction process, i.e., are there times when the merge procedure can choose between different choices of items?
How many *different* Huffman Trees can be built on this frequency set?

(GY5) The goal of the interval partitioning problem (i.e., the classroom assignment problem) taught in lecture was to open as few classrooms as possible to accommodate all of the classes. The greedy algorithm taught, sorted the classes by starting time.

It then ran through the classes one at a time; at each step it first checked if there was an available empty classroom. Only if there was no such classroom would it open a new classroom.

Prove that if the classes are sorted by finishing time the algorithm might not give a correct answer.

Note: Proving that something does not work usually means to find a counter-example.

(GY*1) A Huffman Coding Variant

(a) Recall that in each step of Huffman's algorithm, we merge two trees with the lowest frequencies. Show that the frequencies of the two trees merged in the ith step are at least as large as the frequencies of the trees merged in any previous step.

(b) Suppose that you are given the n input characters, already sorted according to their frequencies. Show how you can now construct the Huffman code in $O(n)$ time. (*Hint:* You need to make clever use of the property given in part (a). Instead of using a priority queue, you will find it advantageous to use a simpler data structure.)

(GY*2) Extra problem. Huffman Coding and Megersort.

Recall that Mergesort can be represented as a tree with each internal node corresponding to a merge of two lists.

The weight of a leaf is 1;
the weight of an internal node is he sum of the weights of its two children,
  or equivalently, the number of leaves in its subtrees.

The cost of a single Merge is the number of items being merged, so the cost of Mergesort is the sum of the weights of the tree's internal nodes.

(a) Prove that the cost of Mergesort can be rewritten as the weighted external path length of its associated tree, when all leaves have weight 1

(b) Prove that the recursive Mergesort studied in class has height $h = \lceil \log_2 n \rceil$, with $x = 2^h - n$ leaves on level $h-1$ and $n-x$ leaves on level h.

(c) Show that an optimal Huffman tree for n items, all with the same frequency 1, will have the property that the tree will have height $h = \lceil \log_2 n \rceil$, with $x = 2^h - n$ leaves on level $h-1$ and $n-x$ leaves on level h.

(d) Use the above facts to prove that recursive mergesort is *optimal*, i.e., that there is no other merge pattern for merging n items that has lower total cost.

Graphs (GR)

(GR1) Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges.
Assume that there are no self-loops or duplicated edges.
Answer all questions below as a function of $|V|$, the number of vertices.

a) What is the maximum number of edges in G?
b) What is the maximum number of edges in G if two vertices have degree 0.
c) What is the maximum number of edges that an acyclic graph G can have?
d) What is the minimum number of edges in G if G is a connected graph and contains at least one cycle?
e) What is the minimum possible degree a vertex in a connected graph G can have?
f) What is the maximum length of any simple path in G?

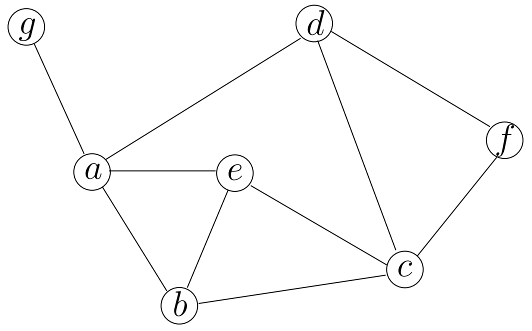
(GR2) Let $G = (V, E)$ be a connected undirected graph. Prove that

$$\log(E) = \Theta(\log V).$$

Note: we implictly use this fact in many of our analyses in class.

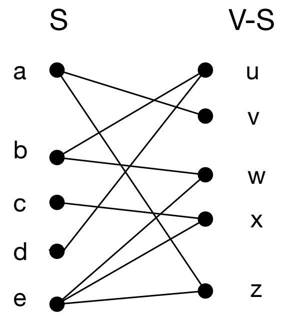
(GR3) The adjacency list representation of a graph G, which has 7 vertices and 10 edges, is:

$a :\rightarrow d, e, b, g$ $\quad$ $b :\rightarrow e, c, a$
$c :\rightarrow f, e, b, d$ $\quad$ $d :\rightarrow c, a, f$
$e :\rightarrow a, c, b$ $\quad$ $f :\rightarrow d, c$
$g :\rightarrow a$



(a) Show the breadth-first search tree that is built by running BFS on graph G with the given adjacency list, using vertex a as the source.
(b) Indicate the edges in G that are NOT in the BFS tree in part (a) by dashed lines.
(c) Show the depth-first search tree that is built by running DFS on graph G with the given adjacency list, using vertex a as the source.
(d) Indicate the edges in G which are NOT in the DFS (c) by dashed lines.

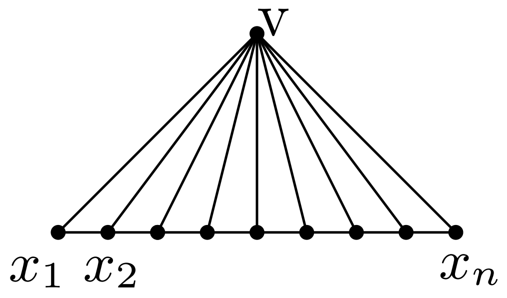
(GR4) An (undirected) graph $G = (V, E)$ is *bipartite* if there exists some $S \subset V$ such that, for every edge $\{u, v\} \in E$, either
(i) $u \in S$, $v \in V - S$ or
(ii) $v \in S$, $u \in V - S$.



Let $G = (V, E)$ be a connected graph. Design an $O(|V| + |E|)$ algorithm that checks whether G is bipartite. *Hint: Run BFS.*

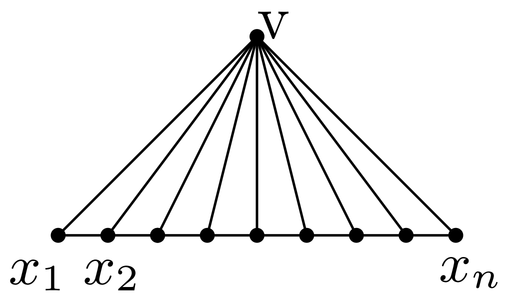
(GR5) In the Fan Graph F_n, node v is connected to all the nodes and the other connections are given by the adjacency lists below.

$$v : x_1, x_2, \ldots, x_n, \qquad x_1 : v, x_2$$
$$x_n : v, x_{n-1} \qquad \forall i \neq 1, n, \quad x_i : v, x_{i-1}, x_{i+1}$$



(a) : Describe the tree that is output when BFS is run on F_n starting from initial vertex v; (ii) initial vertex x_1;
(iii) x_n; (iv) Other x_i.

(b) : Describe the tree that is output when DFS is run on F_n starting from initial vertex v; (ii) initial vertex x_1;
(iii) x_n; (iv) Other x_i.



Now consider the same graph but reorder the adjacency lists as

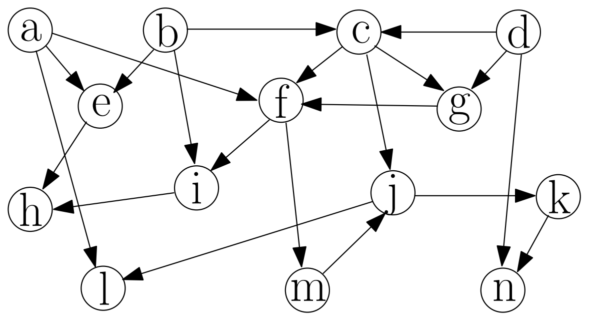
$$\begin{aligned} v &: x_n, x_{n-1}, \ldots, x_1, & x_1 &: x_2, v \\ x_n &: v, x_{n-1} & \forall i \neq 1, n, \quad x_i &: x_{i+1}, x_{i-1}, v \end{aligned}$$

(a') : Describe the tree that is output when BFS is run on F_n starting from initial vertex v; (ii) initial vertex x_1;
(iii) x_n; (iv) Other x_i.

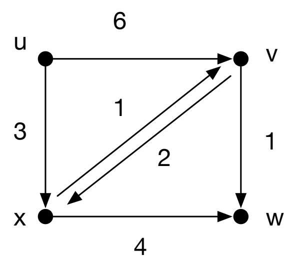
(b') : Describe the tree that is output when DFS is run on F_n starting from initial vertex v; (ii) initial vertex x_1;
(iii) x_n; (iv) Other x_i.

"Describing the tree" means sketching the structure of the tree and also writing down a general formula for $v.p$ (the parent of node v in the BFS/DFS tree) for all v.

(GR6) Give a topological ordering of the following graph.



(GR7) Execute Dijkstra's algorithm on the following digraph, where u is the source vertex.



You need to indicate only the following:

(a) the order in which the vertices are removed from the priority queue.

(b) the final distance values $d[]$ for each vertex.

(c) the different distance values $d[]$ assigned to vertex b, as the algorithm executes.

(GR8) Suppose that instead of using a heap to store the tentative vertex distances, Dijkstra's algorithm just kept an array in which it stored each vertex's tentative distance.

It then finds the next vertex by running through the entire array and choosing the vertex with lowest tentative distance.

What would the algorithm's running time be?

Is this better than our implementation for some graphs?

(GR9) Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why does the correctness proof of Dijkstra's algorithm not go through when negative-weight edges are allowed?

(GR10) Let $G = (V, E)$ be a connected undirected graph in which all edges have weight either 1 or 2. Give an $O(|V| + |E|)$ algorithm to compute a minimum spanning tree of G. Justify the running time of your algorithm. (*Note:* You may either present a new algorithm or just show how to modify an algorithm taught in class.)

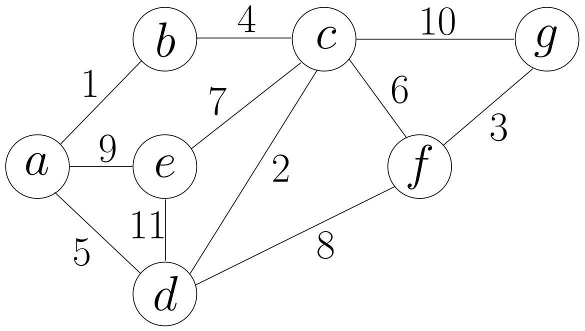
(GR11) Let G be a connected undirected graph with weights on the edges. Assume that all the edge weights are distinct. Prove from first principles that G only has one (unique) MST.

(GR12) Let G be a connected undirected graph with distinct weights on the edges, and let e be an edge of G.

Suppose e is the largest-weight edge in some cycle of G.

Show that e cannot be in the MST of G.

(GR13) Prim's minimum spanning tree algorithm and Dijkstra's shortest path algorithm are very similar, but with crucial differences. Run both algorithms on the following graph, and show the partial MST / shortest path tree after every new edge has been added. The starting vertex for both algorithms is "a".



(GR14) Let $G = (V, E)$ be a weighted graph with non-negative distinct edge weights.

Now replace every weight $w(u, v)$ with its square $(w(u, v))^2$.

(a) Is T still a MST of G with the new weights? Either prove that it is or give a counterexample

(b) Next consider a shortest path $u \to v$ in the original graph. Is this path still a shortest path with the new weights? Either prove that it is or give a counterexample

(GR15) Path Reliability

Suppose that we are given a cable network of n sites connected by duplex communication channels. Unfortunately, the communication channels are not perfect.
The channel between sites u and v is known to fail with (given) probability $f(u, v)$

The probabilities of failure for different channels are known to be mutually independent events.

One of the n sites is the central station and your problem is to compute the most reliable paths from the central station to all other sites (i.e., the paths of lowest failure probabilities from the central station to all other sites).

Design an algorithm for solving this problem, justify its correctness, and analyze its time and space complexities.

(GR16) Let $T = (V, E)$ be a tree and $e = (u, v) \in E$.

Show that removing e from T leaves a graph with exactly two connected components with one component containing u and the other containing v.

(GR17) It is not difficult to see that if e is a minimum weight edge in G then e is always an edge in *some* Minimum Spanning Tree for G.

Prove that if e is a maximum weight edge in G, the corresponding statement is not correct.

This means proving that it is possible that e *does* belong to a MST of G, i.e., provide a counterexample. (Note: It is also possible that e does not belong to any MST for G.)

Dynamic Programming

(DP1) Give an $O(nk)$-time dynamic programming algorithm that makes change using the minimal possible number of coins.

The solution obviously depends upon the country you are in.

Let the local given coin denominations be $d_1 < d_2 < \cdots d_k$, where $d_1 = 1$ (which guarantees that some solution always exists).

(DP2) KFCC is considering opening a series of restaurants along the highway. The n available locations are along a straight line; the distances of these locations from the start of the Highway are given in miles and in increasing order: $m_1, m_2, \ldots, m_n$. The constraints are as follows:

(a) At each location, KFCC may open at most one restaurant.
The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.

(b) Any two restaurants should be at least k miles apart,
where k is a given positive integer.

Give a dynamic programming algorithm that determines a set of locations at which to open restaurants that maximizes the total expected profit earned.

(DP3) Give an $O(n^2)$ time dynamic programming algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers, i.e, each successive number in the subsequence is greater than its predecessor.

For example, if the input sequence is $\langle 5, 24, 8, 17, 12, 45\rangle$, the output should be either $\langle 5, 8, 12, 45\rangle$ or $\langle 5, 8, 17, 45\rangle$.

(DP4) The subset sum problem is: Given a set of n positive integers, $S = \{x_1, x_2, \ldots, x_n\}$ and an integer W determine whether there is a subset $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

For example, if $S = \{4, 2, 8, 9\}$ and $W = 11$, then the answer is "yes" because there is a subset $S' = \{2, 9\}$ whose elements sum to 11. If $W = 7$. the answer is "no".

Give a dynamic programming solution to the subset sum problem that runs in $O(nW)$ time. Justify the correctness and running time of your algorithm.

(DP5) The (Restricted) Max-Sum Problem.

Let A be a sequence of n numbers $a_1, a_2, \ldots, a_n$.

Find a subset S of A that has the maximum sum, provided that, if $a_i \in S$, then $a_{i-1} \notin S$ and $a_{i+1} \notin S$.

Note that, unlike in the previous question, A is a sequence in which order matters (and not an unordered set).

As an example, if $A = 1, 8, 6, 3, 7$, the max possible sum is $S = \{8, 7\}$.

(DP6) Give an $O(nW)$ dynamic programming algorithm for the 0-1 knapsack problem where n is the number of items and W is the max weight that can fit into the knapsack. Recall that the input is i items with given weights $w_1, w_2, \ldots, w_n$ and associated values v_1, $v_2, \ldots, v_n$ and the objective is to choose a set of items with weight $\leq W$ with maximum value.

Now suppose that you are given *two* knapsacks with the same max weight. Give an $O(nW^2)$ dynamic programming algorithm for finding the maximum value of items that can be carried by the two knapsacks.

Note: The one-knapsack problem is taught in the COMP3711 lecture.

(DP7) (Problem from an old exam)

Let $''A'' \to 1$, $``B'' \to 2$, $\ldots$, $``Z'' \to 26$.

Given an encoded message M containing n digits in $1 \ldots 9$, design an $O(1)$-time dynamic programming algorithm to determine the total number of ways to decode M.

As an example, 15243 can be decoded in 4 different ways, "AEBDC", "AEXC", "OBDC", and "OXC".

(DP8) A sequence of numbers $a_1, a_2, \ldots, a_n$ is *oscillating* if

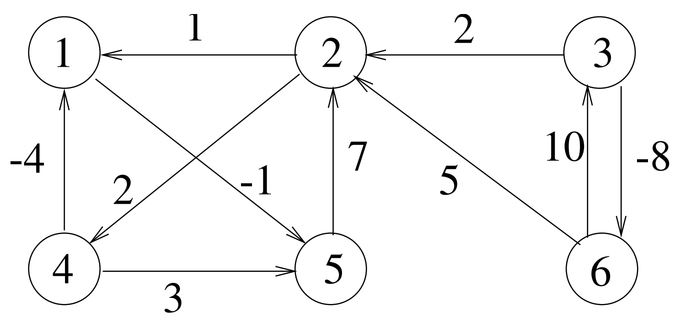
$$a_i < a_{i+1} \text{ for every odd index } i \quad \text{and} \quad a_i > a_{i+1} \text{ for every even index } i.$$

For example, the sequence $2, 7, 1, 8, 2, 6, 1, 8, 3$ is oscillating.

Describe and analyze an efficient dynamic programming algorithm to find a longest oscillating subsequence in a sequence of n integers.

(DP9) Use the DP approach to design an $O(n)$ time algorithm for solving the maximum contiguous subarray problem.

(DP10) Run the Floyd-Warshall algorithm on the weighted, directed graph shown in the figure. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.



(DP11) (CLRS) Give an algorithm that takes as input a directed graph with positive edge weights, and returns the cost of the shortest cycle in the graph (if the graph is acyclic, it should say so). Your algorithm should take time at most $O(n^3)$, where n is the number of vertices in the graph.

(DP12) Assume that all edges have positive weight. Design an algorithm that will, for every pair of vertices, count the *number* of shortest paths between that pair.

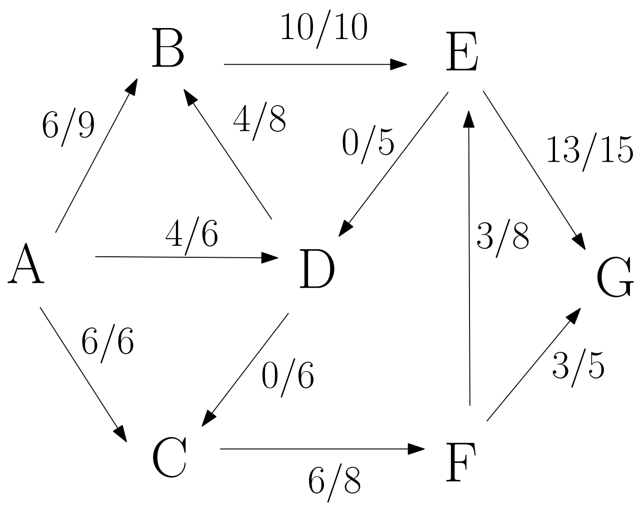
(DP13) In the class notes on the Floyd-Warshall Algorithm we said that it was possible to reduce the space requirement from $O(n^3)$ to $O(n^2)$ by not keeping each of the n $n \times n$ matrices $D^{(i)}$ but instead keeping only ONE matrix and reusing it.

We then wrote the code for doing that.

Why does this space-reduced code work and give the correct answer?

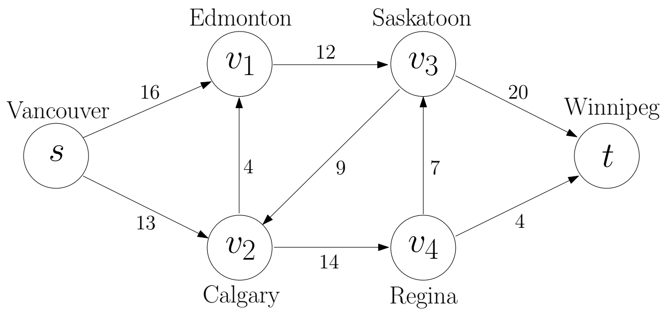
Max Flow and Matchings

(MM1) Consider the given graph with flow values f and capacities c (f/c) as shown. $s = A$ and $t = G$.



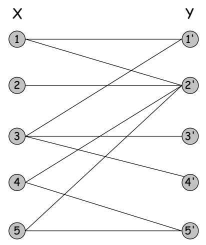
Draw, the residual graph.
Find an augmenting path.
Show the new flow created by adding the augmenting path flow.
Is your new flow optimal?
Prove or disprove.

(MM2) Show the execution of the Edmonds-Karp algorithm on the following flow network.



Recall that the Edmonds-Karp algorithm is to implement Ford-Fulkerson by always choosing a *shortest path* (by number of edges) in the current residual graph.

(MM3) Find a Maximum Bipartite Matching in the graph below using the Max-Flow Method taught in class.



(MM4) Find Find Stable Matchings based on these preference lists.

Man	1st	2nd	3rd	4th
A	c	a	b	d
B	a	d	c	b
C	d	a	b	c
D	d	b	a	c

Woman	1st	2nd	3rd	4th
a	B	D	C	A
b	A	C	D	B
c	B	D	C	A
d	A	D	C	B

(1) What is the Man-Optimal matching?
(2) What is the Woman-Optimal matching?
(3) Are they the same?

(MM5) **Multisource and Multisink**

Note: This problem is only intended for COMP3711. COMP3711H saw this already in class. Also be aware that the mathematical formulations used for the max flow problem are slightly different for COMP3711 and COMP3711H. This was written using the COMP3711 formulations.

Max-Flow as taught in class assumed a single source and single sink.

Suppose the flow network has multiple sources $s_1, s_2, \dots, s_m$ and multiple sinks $t_1, t_2, \dots, t_n$ and the goal is to move as much from all the sources to all of the sinks as possible.

Extend the flow properties and definitions to the multiple-source, multiple-sink problem. Show that any flow in a multiple-source, multiple-sink flow network corresponds to a flow of identical value in the single-source, single-sink network obtained by adding a supersource and a supersink, and vice versa.

(MM6) **An Automaton Pattern-Matching Example**

Set pattern $P = abcab$.

- Construct the transition function $\delta(q, s)$ for the string matching automaton corresponding to P.
- Draw M. (It is not necessary to show the transitions to state 0.)
- Run M on the text T below. For each character, identify the state that M will be in after reading that character.

$$T = a\ c\ a\ b\ c\ a\ b\ c\ a\ b\ a\ b\ c\ a\ b\ c\ a\ b\ c\ a\ b\ a.$$

- Show how this identifies all occurrences of P in T.

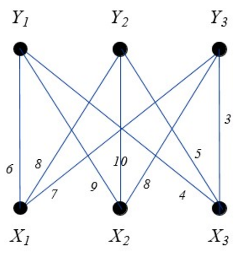
(MM*1) **The Taxi Problem**
Consider a taxi company that has received many reservations. A reservation specifies when and where a taxi needs to be to pick up a passenger and when and where the taxi will drop the passenger off.

The company wants to calculate the minimum number of taxis it will need to service all of those requests. How can it do this?

More specifically, you are given n taxi reservations $r_1, r_2, \ldots, r_n$. For every pair of reservations r_i, r_j you are told if the same taxi can first satisfy reservation r_i and then go on to satisfy reservation r_j. Find the minimum number of taxis needed to satsify all of the reservations.

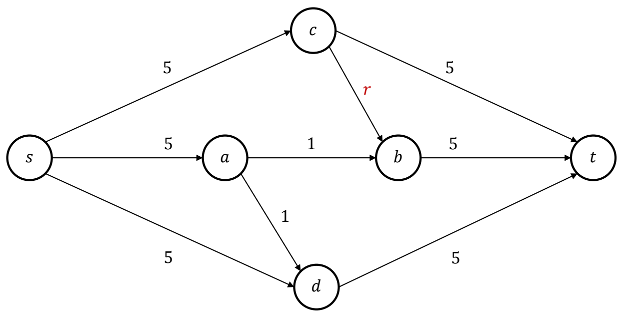
(MM*2) **The Hungarian Algorithm**

Find a max-weight perfect matching in the weighted bipartite graph below:



(MM*3) **Pathological Example of Ford-Fulkerson**

Consider the flow graph



where r is the reciprocal of the Golden Ratio,

$$r = \left(\frac{\sqrt{5}+1}{2} \right)^{-1} = \frac{\sqrt{5}-1}{2} \approx 0.619\ldots.$$

This has a maximum flow with $|f| = 11$.

Show that there exists an infinite sequence of augmenting path steps whose sum converges to a flow with value less than 11.

Hashing

(HA1) **Open Addressing**
Let table size be $m = 15$ (with items indexed from $0 \dots 14$).
Use the hash function $h(x) = (x \bmod 15)$ and linear hashing to hash the items $19, 6, 18, 34, 25, 34$ in that order.
Draw the resulting table.

(HA2) **Universal Hashing**
Recall the universal hash function family defined by

$$h_{a,b}(x) = \Big((ax + b) \bmod p\Big) \bmod m$$

where $a \in Z_p^*$, $b \in Z_p$ and p is a prime with $p \geq U$. Let $p = 17$, $m = 5$. For all $x = 0, 1, \dots, 16$ write the values for $h_{1,0}(x)$. Now write all the values for $h_{2,2}(x)$.