CSE 21 – Fall 2018
Solutions to Practice Final

1. Order questions. For each of the following pairs of functions f and g, choose whether $f \in o(g)$, $f \in \Theta(g)$, or $f \in \omega(g)$ (exactly one will be true).

(a) $f(n) = n, g(n) = 2^{\lfloor \log n \rfloor}$

(b) $f(n) = \log(n^4), g(n) = (\log n)^4$.

(c) $f(n) = n^2, g(n) = 1000n^2 + 2^{100}$

(d) $f(n) = 2^{2n}, g(n) = 2^n$.

(e) $f(n) = n!, g(n) = n^n$.

Solution:

(a) First, observe that $n = 2^{\log n}$. So, $2^{\lfloor \log n \rfloor} < 2^{\log n} = n$. This suggests the Θ relationship for these functions, because they are always very close to each other. Let's pick constants to prove this:

$$c_1 \times 2^{\lfloor \log n \rfloor} \leq n \leq c_2 \times 2^{\lfloor \log n \rfloor}$$

By the above, we can set $c_1 = 1$, and $c_2 = 2$. , making the rhs of the above inequality $2^{\lfloor \log n \rfloor + 1}$. This (more than) makes up for the fractional power lost to flooring, and proves $f \in \Theta(g)$.

(b) Again, we first simplify the functions and consider their relationship intuitively. Using properties of logs, $f(n) = \log(n^4) = 4\log(n)$. Since $g = \log(n)^4$, this suggests that eventually $g(n) > f(n)$. We'll use the limit test to confirm this:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{4\log(n)}{\log(n)^4} = \frac{4}{\log(n)^3} = 0$$

Proving that $f \in o(g)$.

(c) Considering the functions informally, we see that the n^2 term will dominate both functions asymptotically. This suggests that they are equivalent, and we should attempt to prove the Θ relationship.

One direction and constant is clear: $n^2 < 1000 \times n + 2^{100}$, so the constant is 1. In the other direction, let's set up and inspect the desided inequality:

$$c(1000n^2 + 2^100) < n^2$$
$$c1000n^2 + c2^100 < n^2$$

Setting $c = 2^{-100}$ and denoting $c \times 1000 < 1$ by ϵ because it is a very small number that is guranteed to be less than 1 with that setting of c:

$$\epsilon n^2 + 1 < n^2$$

Since $\epsilon < 1$, the inequality above holds for any n sufficiently large. We can set $n_0 = 4$, proving $f \in \Theta(g)$.

(d) Comparing the functions informally, we have $2^{2n} > 2^n$ for any n. So this suggests a ω relationship, and the following limit test:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^{2n}}{2^n} = 2^n = \infty$$

proving $f \in \omega(g)$.

(e) Unrolling the definitions, we know that $n! < n^n$ for any n, so this suggests the o relationship. The limit is annoying to evaluate, so we use the definitions directly. The desired inequality is, for *any* c, eventually

$$cn! < n^n$$

This is true because we can divide both sides by $n!$, and eventually $n^n/n! = \frac{n}{n} \cdot \frac{n}{n-1} \cdots \frac{n}{1} > c$ for any c.

2. Analyzing algorithms. What is the order of the standard multiplication algorithm for multiplying two $n \times n$ matrices? Suppose that any two matrix entries can be multiplied together in constant time, and that addition takes constant time.

Solution: To analyze the standard matrix multiplication algorithm, we must first know the standard matrix multiplication algorithms. Let A and B be $n \times n$ matricies. The standard algorithm is:

1. For each i in $1 \ldots n$
   (a) For each j in $1 \ldots n$
      i. do: $[AB]_{i,j} = \sum_{r=1}^{n} A_{i,r} \times B_{r,j}$

Note that we were told that, in this situation, addition and multiplication are constant-time. Looking at the loop structure, we see that the inner-most line of code (line i) is executed once for each pair (i, j). Since the matricies are of dimension $n \times n$, this inner line must be executed n^2 times. So we can obtain the full cost of the algorithm by multiplying the cost of the inner line by n^2 (number of times it is executed).

The summation in the inner line runs from 1 to n, so there are n addition operations. Each addition operation is run on the output of a product, so there are n product operations. Therefore, the total cost of the inner line is $2n$. Multiplying this by the number of times it is executed, we get $2n^3$, which is $O(n^3)$, the asymptotic cost of the standard algorithm for matrix multiplication.

3. Recursive algorithms. Here's a recursive algorithm, that given an n bit number in binary, $x = x_{n-1} \ldots x_0$ and an $m < n$ bit number in binary y, computes $x \mod y$. It uses the following sub-routines: $Add(x, y)$ adds two n bit binary numbers in $O(n)$ time, $AsLarge(x, y)$ returns true if and only if $x \geq y$ and also takes $O(n)$ time, and $Subtract(x, y)$ computes $x - y$ and is also $O(n)$ time. We assume $y \geq 2$ and $n \geq 1$.

Mod ($x = x_{n-1} \ldots x_1 x_0$, y)

1. IF $n = 1$, return x_0.
2. $v \leftarrow Mod(x_{n-1} \ldots x_1,\ y)$.
3. $v \leftarrow Add(v, v)$
4. $v \leftarrow Add(v, x_0)$.
5. IF $AsLarge(v, y)$ THEN $v \leftarrow Subtract(v, y)$.
6. Return v

(a) **Correctness.** Prove by induction on n that $Mod(x, y)$ returns $x \mod y$.
Hints:
- Let $x = 2a + x_0$, where $a = x_{n-1} \ldots x_1$.
- To show that two numbers are the same mod y, show that their difference is a multiple of y.

(b) **Recurrence.** Give a recurrence for $T(n)$, the time this algorithm takes on an input of size n.

(c) **Solving recurrences.** Solve this recurrence to give the Big O time of this algorithm.

Solution:

(a) First, we must prove correctness. Using the hints, we'll let $x = 2a + x_0$, and then we'll re-write a using the division theorem as $a = k_1 y + r_1$.

To prove a recursive algorithm correct, we need to use induction. The **base case** for this algorithm is $n = 1$ (invoking the algorithm on a 1-bit number). In this case, looking at line 1, we see that the algorithm just returns the single bit. This is correct, because a single-bit number can only be 0 or 1, and $0 \mod y = 0$ for any $y \geq 2$, and $1 \mod y = 1$ for any $y \geq 2$. We have this condition on y from the restrictions given on the input.

For the inductive case, we first assume that there is a $c > 1$ such that the *Mod* algorithm listed works for inputs of c bits (that is, for any x of c bits, $Mod(x, y)$ correctly returns $x \mod y$). This is the **inductive hypothesis**. Now, we must use this assumption to show that the algorithm is correct for inputs of $c + 1$ bits.

Note that $a = \lfloor \frac{x}{2} \rfloor$ is the c-bit number that *Mod* is called on during line 2, because we are dropping the least significant bit of x and "shifting" right. This is what allows us to write $x = 2a + x_0$.

First, let x be an arbitrary $c + 1$ bit number. Now, simulate the algorithm:

1. We know that the condition in line 1 will not be true, because $c + 1 > 1$, so the algorithm continues.
2. In line 2, we use the **IH** and the fact that a is c bits to assert that v is assigned the value $a \mod y$, which in our re-expression of a above is r_1.
3. The value in v is doubled.
4. The value in v has x_0 added to it.

After line 4, then, we have that:

$$v = 2r_1 + x_0$$

This algorithm returns v, possibly after subtracting y from it once. Since subtracting a multiple of y (such as $1 \times y$) will not change the value of $v \mod y$, all we need to do is prove that $v \mod y$ and $x \mod y$ are the same. Now we use the second hint: if we

prove that $x - v$ is a multiple of y (that is, $\exists k'$ such that $x - v = k'y$) then we'll have that $v \mod y = x \mod y$ and the algorithm is correct after (possibly) adjusting the range of v to be between 0 and $y - 1$.

We'll show this by expanding the definitions of $x - y$. The key insight here is using the division theorem to rewrite v and a.

$$\begin{aligned} x - v &= (2a + x_0) - (2r_1 + x_0) \\ &= (2(k_1 y + r_1) + x_0) - 2r_1 - x_0 \\ &= 2k_1 y + 2r_1 + x_0 - 2r_1 - x_0 \\ &= 2k_1 y \\ &= k'y \end{aligned}$$

Therefore, by the hint, $x \mod y = v \mod y$. To see why the hint is true, take arbitrary a and b and re-express them using the division theorem for y, and then subtract the re-expressions. The difference is a multiple of y iff a and b have the same remainder when divided by y.

The only thing we need to account for now is the size of v at the end of the algorithm: it could be as large as $2(y - 1)$. The value that we want is a remainder after dividing by y, so it must be between 0 and $y - 1$. Because the v is maximixed at $2(y - 1)$ and because subtracting (or adding) multiples of y does not change a value modulo y, it suffices to subtract off just one y if v is too large.

To see why subtracting multiples of y does not change a value mod y, look again at the division theorem:

Express some a as $a = qy + r$, then subtract ly for some y, getting: $a - ly = (q - l)y + r$. The remainder has not changed, so the value mod y is the same.

(b) Now we write the following recurrence. We are told that $Add(x, y)$, $AsLarge(x, y)$, and $Subtract(x, y)$ all have $O(n)$ time, so from that we can say:

- *Add* has time $k_1 n$
- *AsLarge* has time $k_2 n$
- *Subtract* has time $k_3 n$

For some constants k_i. There is only one recursive call, on line 2. Note that *Add* is called twice. So the entire recurrence is:

$$\begin{aligned} T(n) &= T(n-1) + k_1 n + k_1 n + k_2 n + k_3 n \\ &= T(n-1) + (2k_1 n + k_2 n + k_3 n) \\ &= T(n-1) + kn \end{aligned}$$

For some $k = 2k_1 + k_2 + k_3$. Asymptotically, this is:

$$T(n) = T(n-1) + O(n)$$

With $T(1) = 1$ for the constant work involved in testing for a single bit and returning it.

(c) Now we solve the recurrence to obtain the runtime of the algorithm. Unwinding, look for a pattern:

$$\begin{aligned} T(n) &= T(n-1) + kn \\ &= T(n-2) + k(n-1) + kn \\ &= T(n-3) + k(n-2) + k(n-1) + kn \end{aligned}$$

Recognizing the pattern, we see that at each step of unrolling there is only one recursive term, and one new $k(n-i)$ term introduced into the sum. Eventually, when the term $T(n-(n-1))$ comes up, this will evaluate to 1 and the unrolling/recursion will stop. Therefore, we have:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} ki \\ &< k \sum_{i=1}^{n} i \\ &= k \times \frac{1}{2} \times n(n+1) \end{aligned}$$

is $O(n^2)$. Note that the $n-i$ terms were flipped to i by recognizing that this is the same set of numbers in the summation, just counted "backwards" instead of "forwards." This solves the recurrence and the runtime of the algorithms is therefore $O(n^2)$.

4. Representing problems as graphs. You have a system of n variables representing real numbers, $X_1, \ldots, X_n$. You are given a list of inequalities of the form $X_i < X_j$ for some pairs i and j. You want to know whether you can deduce with certainty from the given information that $X_1 < X_n$.

For example, say n is 4. A possible input is the list of inequalities $X_1 < X_2$, $X_1 < X_3$, $X_4 < X_3$ and $X_3 < X_2$. Does it follow that $X_1 < X_4$?

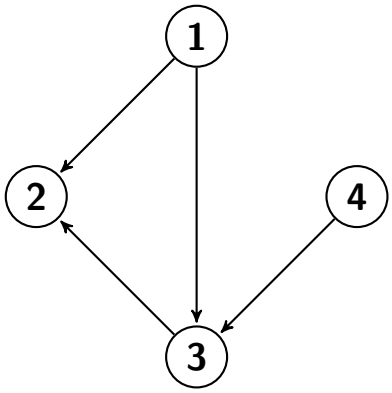
(a) **Defining the Graph.** Give a description of a directed graph that would help solve this problem. Be sure to define both the vertices and edges in terms of the variables and known inequalities.

(b) **Example.** Draw the graph you described for the example above. Does $X_1 < X_4$ follow? Why or why not?

(c) **Graph algorithms.** Say which algorithm from lecture we could use on such a graph to determine whether $X_1 < X_n$ follows from the known inequalities.

Solution:

(a) There will be n vertices labeled $1, 2, \ldots, n$. Connect vertex i to vertex j by a directed edge from i to j if and only if $X_i < X_j$. This helps solve the problem because we can deduce that $X_i < X_j$ if there exists a path from i to j in the graph.

(b) 

It doesn't follow that $X_1 < X_4$ because there is no path from 1 to 4 in the graph.

(c) We could use the Graph Search algorithm to decide whether or not n is reachable from 1. If it is, then $X_1 < X_n$. Otherwise we can't be sure of the relationship between X_1 and X_n.

5. DAGs and trees. A binomial tree is a special kind of rooted tree used for various data structures in computer science. A degree d binomial tree can be defined recursively as follows. A degree 0 binomial tree is a single vertex with no edges. A degree d binomial tree has a root vertex with out-degree d. The first (that is, leftmost) subtree is a degree $d-1$ binomial tree. The second (that is, second to left) subtree is a degree $d-2$ binomial tree. Continue on in this way so that the last (rightmost) subtree is a degree 0 binomial tree.

(a) What is the height of a degree d binomial tree? Prove your result by induction on d.

(b) Write a recurrence for the number of nodes $N(d)$ in a binomial tree of degree d.

(c) Use the guess-and-check method to guess a formula for $N(d)$. Prove that your formula holds by induction on d.

Solution:

(a) A binomial tree of degree d has a height of d. We prove this statement by strong induction on d.

When $d = 0$ the tree is just one node, so it has a height of 0.

Now let $d \geq 0$. Assume as the inductive hypothesis that for all $0 \leq k \leq d$, a binomial tree of degree k has a height of k.

Consider a binomial tree of degree $d+1$. Our root has one child each of degree d,d-1,$\ldots$0. By our induction hypothesis, for each value of k, $0 \leq k \leq d$, the subtree rooted at the child of degree k has a height k. Therefore the maximum height of one of these subtrees is d (choosing the largest value of k). Since our binomial tree of degree $d+1$ has a subtree of height d, then it must have a height of $d+1$.

(b)
$$N(d) = N(d-1) + N(d-2) + \cdots + N(0) + 1, N(0) = 1$$

This comes from counting the number of nodes in each subtree, from left to right, plus one additional root node.

(c) We guess that $N(d) = 2^d$, and we prove the result by strong induction on d.

When $d = 0$ the tree is just one node so $N(0) = 1$, which satisfies the formula since $2^d = 2^0 = 1$.

Now let $d \geq 0$. Assume as the inductive hypothesis that for all $0 \leq k \leq d$, $N(k) = 2^k$.

Then

$$\begin{aligned} N(d+1) &= N(d) + N(d-1) + N(d-2) + \cdots + N(1) + N(0) + 1 \\ &= 2^d + 2^{d-1} + 2^{d-2} + \cdots + 2^1 + 2^0 + 1 \quad \text{(by IH)} \\ &= 2^{k+1} \quad \text{(by a formula for the sum of powers of two, or by geometric series)} \end{aligned}$$

6. Counting. (a) How many 5-card hands can be formed from an ordinary deck of 52 cards if exactly two suits are present in the hand?

(b) In any bit string, the longest consecutive run length is the maximum number of consecutive 1's or consecutive 0's in the string. For example, in the string 1101000111, the longest consecutive run length is 3. How many bit strings of length 10 have a longest consecutive run length of 6?

(c) A software company assigns its summer interns to one of three divisions: design, implementation, and testing. In how many ways can a group of ten interns be assigned to these divisions if each division needs at least one intern?

(d) How many numbers in the interval [1,10000] are divisible by 7, 9, or 11?

Solution:

(a) There are two disjoint possibilities to have exactly two suits:

- 3 cards of one kind + 2 cards of another: The steps involved are:
  (a) Select a kind (k_1) for 3 cards - $\binom{4}{1}$
  (b) Select a kind (k_2) for 2 cards - $\binom{3}{1}$
  (c) Select 3 cards from kind k_1 - $\binom{13}{3}$
  (d) Select 2 cards from kind k_2 - $\binom{13}{2}$
  Total number of ways is $\binom{4}{1} \cdot \binom{3}{1} \cdot \binom{13}{3} \cdot \binom{13}{2}$
- 4 cards of one kind + 1 card of another kind: Using similar counting as previous case, $\binom{4}{1} \cdot \binom{3}{1} \cdot \binom{13}{4} \cdot \binom{13}{1}$

Total number of ways is: $\binom{4}{1} \cdot \binom{3}{1} \cdot \binom{13}{3} \cdot \binom{13}{2} + \binom{4}{1} \cdot \binom{3}{1} \cdot \binom{13}{4} \cdot \binom{13}{1}$

(b) Let $c_1, c_2, c_3, \ldots, c_{10}$ be the 10 spots for 10 digits. We need to fill these spots with 0, 1 such that there are exactly 6 consecutive spots with same number. There are two cases:

- The six consecutive spots are at the beginning or the end. That is, starting location of six consecutive spots is either c_1 or c_5. All the consecutive spots need to take same value and the adjacent spot should take the opposite value. This can be done in two ways. The remaining 3 spots can take any of the two values each (or in 2^3 ways). So, total possibilities is $2 \cdot 2 \cdot 2^3 = 32$
- The six consecutive spots are in the middle. That is, starting location of size consecutive spots is one of c_2, c_3, c_4. All the consecutive spots need to take same value and the two adjacent spots should take the opposite values. This can be done in two ways. The remaining 2 spots can take any of the two values each (or in 2^2 ways). So, total possibilities is $3 \cdot 2 \cdot 2^2 = 24$

So, total number of ways is 56.

(c) We'll use the principle of inclusion-exclusion for this problem.

Each person can be assigned to one of the three divisions, so for 10 people the total number of ways of assigning interns to divisions is 3^{10}.

Now let's count the number of ways that all of them go to just two particular divisions, say design and implementation. We have to exclude this case from our count because testing will be empty. There are two possible assignments for each person, so the number of possibilities is 2^{10}. Similary, we have to account for other two cases where everyone goes to implementation and testing or everyone goes to design and testing. In total, there are $3 \cdot 2^{10}$ ways that the interns can be assigned to only two divisions.

Finally, we must compute the number of ways in which interns are assigned to only one division. There are 3 ways to select which single division will have all interns, and this is the only choice we can make.

Applying the principle of inclusion-exclusion, the number of ways the interns can be assigned to the divisions so that each division has at least 1 intern is given by the total number of possible assignment, minus the number of ways to assign interns to only 2 divisions, plus the number of ways to assign interns to 3 divisions, which is $3^{10} - 3 \cdot 2^{10} + 3$.

(d) Use inclusion-exclusion with three sets:
$A = \{$ numbers in the interval $[1, 10000]$ which are divisible by $7\}$,
$B = \{$ numbers in the interval $[1, 10000]$ which are divisible by $9\}$, and
$C = \{$ numbers in the interval $[1, 10000]$ which are divisible by $11\}$

We have
$$|A| = \left\lfloor \frac{10000}{7} \right\rfloor = 1428,$$
$$|B| = \left\lfloor \frac{10000}{9} \right\rfloor = 1111, \text{ and}$$
$$|C| = \left\lfloor \frac{10000}{11} \right\rfloor = 909.$$
To compute the intersection of two of these sets, notice that since $7, 9$, and 11 have no common factors (they are relatively prime), a number that is divisible by two of $7, 9$, and 11 must be divisible by their product. For example, a number being divisible by 7 and 11 just means it is divisible by 77. So,
$$|A \cap B| = \left\lfloor \frac{10000}{63} \right\rfloor = 158,$$
$$|A \cap C| = \left\lfloor \frac{10000}{77} \right\rfloor = 129, \text{ and}$$
$$|B \cap C| = \left\lfloor \frac{10000}{99} \right\rfloor = 101.$$
Similarly, if a number is divisible by $7, 9$, and 11, it must be divisible by their product 693, and vice versa. Therefore,
$$|A \cap B \cap C| = \left\lfloor \frac{10000}{693} \right\rfloor = 14.$$

Putting this all together, the inclusion-exclusion formula gives:

$$\begin{aligned} |A \cup B \cup C| &= |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C| \\ &= 1428 + 1111 + 909 - 158 - 129 - 101 + 14 \\ &= 3074. \end{aligned}$$

7. Encoding and Decoding A *three coloring* of a graph labels each vertex v with one of three *colors*, say R, B or G, so that the two endpoints of any edge have different colors. Consider an undi-

rected graph which is a single path, i.e., where the vertices are $v_1 ... v_n$, and there is an edge between each v_i and v_{i+1} for $i = 1..n-1$.

(a) In terms of n, how many 3-colorings does this graph have?

(b) In terms of n, how many bits are required to describe such a 3-coloring?

(c) Describe how to encode a 3-coloring of such a graph with n vertices using the number of bits from part (b).

(d) How would your encoding scheme encode the coloring:

$$B, R, B, R, G, R, B, G$$

Solution:

(a) There are three options for the first vertex v_1 in the path. All the remaining vertices can be only be colored in two colors, as the preceding color cannot be reused. As such, there are $3 * 2^{n-1}$ 3-colorings for the graph.

(b) As a general rule, we take the ceiling of the log base 2 of the number of objects we are representing. In this case ,this gives

$$\lceil \log_2(3 * 2^{n-1}) \rceil = \lceil \log_2(3) + \log_2(2^{n-1}) \rceil = \lceil \log_2(3) + n - 1 \rceil = n + 1$$

so it takes $n + 1$ bits to represent a 3-coloring of the given graph.

(c) First, we take a given 3-coloring on a graph of n vertices where there is an edge between each v_i and v_{i+1} for $i = 1..n-1$ and convert it into a ternary string of length n. We denote R as a 0-bit, B as a 1-bit, and G as a 2-bit. The condition that two endpoints of any edge must have different colors implies that corresponding ternary strings of will never have the same symbol appearing twice in a row.

Use two bits to encode the first character in the string (with 00 denoting 0, 01 denoting 1, and 10 denoting 2), and then use one additional bit to encode every subsequent character. This bit specifies the next ternary character among the two that are possible, with 0 denoting the smaller of the two possible ternary characters in that position and 1 denoting the larger. This encoding uses $n + 1$ bits to encode a ternary string of length n, since it uses 2 bits for the first ternary character and 1 bit for each of the remaining $n - 1$ ternary characters. Therefore, it is optimal and uses the number of bits given in part (b).

(d) First, the coloring $BRBRGRBG$ will be converted into the ternary string 10102012. This string will then be encoded as 010001001.

8. Probability. (a) I have 10 shirts, 6 pairs of pants, and 3 jackets. Every day I dress at random, picking one of each category. What is the probability that today I am wearing at least one garment I was wearing yesterday?

(b) A permutation of size n is a rearrangement of the numbers $\{1, 2, \ldots, n\}$ in any order. A *rise* in a permutation occurs when a larger number immediately follows a smaller one. For example, if $n = 5$, the permutation 1 3 2 4 5 has three rises. What is the expected number of rises in a permutation of size n?

(c) There are n teams in a sports league. Over the course of a season, each team plays every other team exactly once. If the outcome of each game is a fair random coin flip, and there are no ties, what is the probability that some team wins all of its games?

(d) Suppose there are 10 people of distinct heights standing in line for free food, arranged in a random order. Use linearity of expectation to calculate the expected number of people who can see all the way to the front of the line.

Solution:

(a) Let us instead compute the complement, since this is a question about whether there exists a match between today's clothing and yesterday's.

Let A be the event that I am wearing all different clothes from yesterday.

We can see that $P(A) = (9/10) * (5/6) * (2/3)$. The reason for this is that today I can wear any of the other garments that I didn't wear yesterday.

Thus the probability of wearing at least one of the same garments as yesterday is $1 - P(A) = 1 - (9/10) * (5/6) * (2/3)$.

(b) We want $E[X]$ where X is the number of rises in the permutation. However, the number of rises in the permutation is the total number of rises that start at each position. So for $i = 1$ to $n - 1$, define X_i to be 1 if there is a rise starting at position i and 0 otherwise. Thus, using linearity of expectation,

$$E[X] = E[X_1 + X_2 + \cdots + X_{n-1}] = E[X_1] + E[X_2] + \cdots + E[X_{n-1}].$$

If we think of a permutation as a random rearrangement of the numbers from 1 to n, no position in a permutation is more likely to have a rise than any other. This means each X_i has the same distribution and the same expected value. With probability $1/2$ there is a rise at position i and with probability $1/2$ there is a fall, so $E[X_i] = 1 * 1/2 + 0 * 1/2 = 1/2$. Thus, $E[X] = (n-1) * 1/2$.

(c) As there n teams in the league, each team plays $n - 1$ games, so for a team to win all of its games it has to win all $n - 1$ games. Every game to be won has a probability of $1/2$.

Define i events, E_i is the event that team i is undefeated. The event that some team is undefeated is $E_1 + E_2 + \cdots + E_n$, so we want $P(E_1 + E_2 + \cdots + E_n)$. But the events E_i are mutually exclusive since we can't have two undefeated teams, which means we can add their probabilities together and instead compute $P(E_1) + P(E_2) + \cdots + P(E_n)$.

The probability that team i wins all of its games is $P(E_i) = (1/2)^{n-1}$. As there are n teams the total probability that some team is undefeated is $n * (1/2)^{n-1}$.

(d) Let X be the random variable that represents the number of people who can see to the front. Suppose each person is numbered $1, 2, \ldots, 10$. For each i from 1 to 10, let X_i be the random variable defined to be $X_i = 1$ if person i can see to the front and $X_i = 0$ otherwise.

For each i from 1 to 10, $E(X_i) = 1/i$ because for person i to see to the front, he must be taller than all the people before him. So the tallest of persons 1 through i must wind up in position i. The probability that the tallest of the first i people winds up in position i is $1/i$, since there are i positions where the tallest person can go, and only one of them is position i. Therefore $P(X_i = 1) = 1/i$ and $P(X_i = 0) = (1 - 1/i)$ for each person i. Therefore the expected value is $E(X_i) = (1)(1/i) + (0)(1 - 1/i) = 1/i$ for each person i.

It follows that $X = \sum_{i=1}^{10} X_i$ and that $E(X) = E\left(\sum_{i=1}^{10} X_i\right) = \sum_{i=1}^{10} E(X_i)$.

Therefore, $E(X) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} = \frac{7381}{2520}$