CS 70 Discrete Mathematics and Probability Theory
Fall 2015 Satish Rao HW 2

Due Wednesday Sept 9 at 10PM

1. a) Prove that, for any positive integer $n \geq 3$, $n^2 \geq 2n+1$.

   Answer:

   - Base Case ($n = 3$): $3^2 = 9$, $2(3)+1 = 7$, and $9 \geq 7$. It is true.
   - Inductive hypothesis: Assume that $k^2 \geq 2k+1$ is true for some $n = k \geq 3$.
   - Inductive Step: We are trying to prove $(k+1)^2 \geq 2(k+1)+1$. We start with the inductive hypothesis that $k^2 \geq 2k+1$. Adding $2k+1$ to both sides gives $k^2+2k+1 \geq 4k+2$, where the LHS is exactly $(k+1)^2$. For the RHS, we use the fact that $k \geq 3$ to conclude that $2k \geq 1$ and thus $4k+2 = 2k+2k+2 \geq 2k+1+2 = 2k+3$. Now we have $(k+1)^2 \geq 4k+2 \geq 2k+3$, and we complete the proof.

   b) Prove that, for any positive integer $n \geq 4$, $2^n \geq n^2$.

   Answer:

   - Base Case ($n = 4$): $2^4 = 16$, $4^2 = 16$, and $16 \geq 16$. It is true.
   - Inductive hypothesis: Assume that $2^k \geq k^2$ is true for some $n = k \geq 4$
   - Inductive Step: We are trying to prove $2^{k+1} \geq (k+1)^2$.

     $$2^{k+1} = 2 \times 2^k \geq 2 \times k^2 = k^2 + k^2 \geq k^2 + 2k + 1 = (k+1)^2,$$

     where the second step is by the inductive hypothesis, and the fourth step is by Part (a) (note that, in Part (a), the condition $n \geq 3$ covers all cases here). Therefore, since the inductive step is proven true, we complete the proof.

2. Use induction to prove that for all positive integers n, all of the entries in the matrix

   $$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^n$$

   are $\leq 2n$. (Hint: Find a way to strengthen the inductive hypothesis!)

   Answer:

   Before starting the proof, writing out the first few powers reveals a telling pattern:

   $$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^1 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$
   $$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^2 = \begin{pmatrix} 1 & 4 \\ 0 & 1 \end{pmatrix}$$
   $$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^3 = \begin{pmatrix} 1 & 6 \\ 0 & 1 \end{pmatrix}$$

It appears (and we shall soon prove) that the upper left and lower right entries are always 1, the lower left entry is always 0, and the upper right entry is $2n$. We shall take this to by our inductive hypothesis.
Proof: We shall use a proof by induction that the upper left and lower right entries of the matrix are always 1, the lower left entry is always 0, and the upper right entry is $2n$. This will prove that all entries in the matrix are less than or equal to $2n$ for all $n \geq 1$. The base case of $n = 1$ is trivially true. Now suppose that our proposition is true for some $n \geq 1$, meaning

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^n = \begin{pmatrix} 1 & 2n \\ 0 & 1 \end{pmatrix}$$

for some $n \geq 1$. Multiplying both sides of the equation by the original matrix yields

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^{n+1} = \begin{pmatrix} 1 & 2n \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1+0 & 2n+2 \\ 0+0 & 0+1 \end{pmatrix} = \begin{pmatrix} 1 & 2(n+1) \\ 0 & 1 \end{pmatrix}$$

By the principle of induction, our proposition is therefore true for all $n \geq 1$, so all entries in the matrix will be less than or equal to $2n$. ■

3. Prove that for every positive integer k, the following is true:

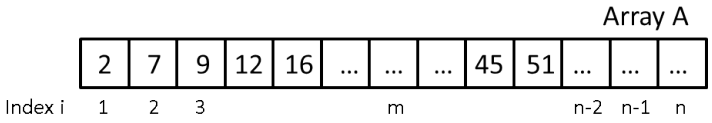
   For every real number $r > 0$, there are only finitely many solutions in positive integers to $\frac{1}{n_1} + \cdots + \frac{1}{n_k} = r$.

   In other words, there exists some number m (that depends on k and r) such that there are at most m ways of choosing a positive integer n_1, and a (possibly different) positive integer n_2, etc., that satisfy the equation.

   Answer: **Claim**: $\forall k \in \mathbf{Z}\ \forall r \in \mathbf{R}\ \big((k > 0 \wedge r > 0) \Rightarrow (\text{There are finitely many solutions to } \frac{1}{n_1} + \cdots + \frac{1}{n_k} = r, n_i \in \mathbf{Z}, n_i > 0)\big)$
   Proof: We will prove this by induction on k. For our base case, $k = 1$. In the base case, iff r can be written as $\frac{1}{n_1}$ when n_1 is a positive integer, then there is exactly one solution, $n_1 = \frac{1}{r}$. If r cannot be written in that form, then there are exactly zero solutions. In all cases, there is a finite number of solutions. For the inductive hypothesis, assume that there are finitely many solutions for some $k \geq 1$ for all r. Each real number r_1 either can or cannot be written as the sum of $k+1$ integers' inverses. If r_1 cannot be written in that form, then there are exactly zero solutions. If r_1 can be written in that form, then the integers' inverses can be ordered. Since r_1 is the sum of $k+1$ integers' inverses, the largest $\frac{1}{n_i}$ must be at least $\frac{r_1}{k+1}$. This means that the smallest n_i must be at most $\frac{k+1}{r_1}$, which means that the smallest n_i has finitely many possible values. For each of the possible smallest n_i values, there is a real number $r_1 - \frac{1}{n_i}$ that can be written as the sum of k integers' inverses in finitely many ways (using the induction hypothesis). This means that there are only finitely many possible solutions for $k+1$ (combining all solutions (finitely many) for each possible smallest n_i values (finitely many)). By the principle of induction, there are finitely many solutions for all k for all r.

4. We have an array A, with sorted values from left to right. There are n values. Two arbitrary numbers are picked from 1 to n, denoting the start and end index of the search range, respectively. Given a value x, we want to use a binary search algorithm to see if this value is contained in the specified range. If x is found in this range, then the program should return the index of where it is in the array. However, if it is not found, it will return 0. An outline of the algorithm goes like this:

   (1) Divide the search range in half and look at $A[m]$ where m is the middle index. If $A[m] = x$, we've found it and the program returns index m.



(2) If not, and $A[m] < x$, search the remaining right part of the array by setting the range from index $(m+1)$ to b. If $A[m] > x$, set the range from a to $(m-1)$.

(3) If the search range is empty, return 0. Otherwise, repeat from step (1).

Part 1. Write pseudocode (be as detailed as you can) of a recursive function that would follow the steps outlined above. No explicit `for` or `while` loops are allowed. **Answer:**

```
global array A
global variable x
function binary_search(x,start,end,A):
  if end < start
   return 0
  middle = (end + start) / 2
  if A[middle] = x
   return middle
  if A[middle] < x
   return binary_search(x,middle+1,end,A)
  if A[middle] > x
   return binary_search(x,start,middle-1,A)
```

`start` is the starting index of the search range, `end` is the ending index of the search range. If the search range has an even number of elements, we take $\lfloor (\texttt{start} + \texttt{end})/2 \rfloor$.

Part 2. We want to know if this pseudocode is doing what it's supposed to, returning a single index if x is in the search range or returning 0 if it's not. Try using induction to prove that your pseudocode is correct. Please outline all the steps in the proof.

Answer: We induct on the difference of `start` and `end`, $d = \texttt{end} - \texttt{start}$.
Base case: We have two base cases, $d = 0$ and $d = 1$.
For $d = 0$, we have a search range with 1 element. Obviously, `start` = `end` = `middle` and if the value is equal to `x`, the program will return this index. If the value is not equal to `x`, there are two cases, `x` $<$ `A[middle]` and `x` $>$ `A[middle]`.

- `x` $<$ `A[middle]`: The ending index for the next step will be `middle-1` = `start-1`, which is smaller than `start`, resulting in a return value of 0.
- `x` $>$ `A[middle]`: Similarly, the starting index for the next step will be `middle+1` = `end+1`, which is bigger than `end`, resulting in a return value of 0.

For $d = 1$, we have a 2-element array. Because we are taking the floor, `middle` = `start`. If `x` = `A[middle]` then the program returns `middle`.

- `x` $<$ `A[middle]`: Returns 0.
- `x` $>$ `A[middle]`: Reduces to $d = 0$ case.

Induction hypothesis: We assume for all arrays with $d \leq n$, we can find the index of `x` or if not, the program returns a 0.

Inductive step: For an array with $d = n+1$, we have two cases when d is even and d is odd. We denote d' as the new difference in the search range indices for the recursive call.

- d is odd: We consider $n = 2,4,6\ldots$. The number of elements in the array is even, and `middle = (start+end-1)/2`. If `A[middle]=x`, the program returns `middle`. Otherwise, if `x < A[middle]`, the recursive call gets a $d' =$ `middle-1-start = (start+end-1)/2-1-start = (end-start-3)/2` $= \frac{n}{2} - 1$, which is clearly $\leq n$, since $-2 \leq n \Rightarrow \frac{n}{2} - 1 \leq n$.
  If `x > A[middle]`, $d' =$ `end-(middle+1) = end-(start+end+1)/2 = (end-start+1)/2` $= \frac{n}{2} + 1$, which is $\leq n$, since $2 \leq n \Rightarrow \frac{n}{2} + 1 \leq n$.
- d is even: We consider $n = 1,3,5\ldots$. The number of elements in the array is odd, and `middle = (start+end)/2`. Similarly to the case above, the program will return `middle` if `A[middle]=x`. Otherwise, if `x < A[middle]`, $d' = \frac{n-1}{2}$, and if `x > A[middle]`, $d' = \frac{n-1}{2}$ (since symmetric). $d' \leq n$ since $-1 \leq n \Rightarrow \frac{n-1}{2} \leq n$.

Thus, by strong induction, we have proved the proposition.

We want to note that in the above, we have written out all the steps in great detail where each step is a basic fact from high school algebra. In a solution, you don't need to write out the steps in that level of detail.

5. You have been asked to assign TAs for the fall semester. Each class has its own method for ranking candidates, and each candidate has their own preferences. An assignment is **unstable** if a class and a candidate prefer each other to their current assignments. Otherwise, it is **stable**.

   Candidate information:

Candidate	CS70 Grade	CS61A Grade	CS61B Grade	Teaching Experience	Overall GPA	Preferences
A	A+	A	A	Yes	3.80	CS70 > CS61A > CS61B
B	A	A	A	No	3.70	CS70 > CS61B > CS61A
C	A	A+	A-	Yes	3.60	CS70 > CS61A > CS61B

   Ranking method:

   - CS70: Rank by CS70 grade. Break ties using teaching experience, then overall GPA.
   - CS61A: Rank by teaching experience. Break ties using CS61A grade, then overall GPA.
   - CS61B: Rank by CS61B grade. Break ties using overall GPA, then teaching experience.

   a) Find a stable assignment.

   b) Can you find another, or is there only one stable assignment (if there is only one, why)?

   CS70 is overenrolled and needs two TAs. There is another candidate.

Candidate	CS70 Grade	CS61A Grade	CS61B Grade	Teaching Experience	Overall GPA	Preference
D	A+	A	A+	No	3.90	CS61A > CS61B > CS70

c) Find a stable assignment.

d) Prove your assignment in Part (c) is stable.

Answer:

(a) Use SMA with the following class rankings and students proposing.

CS70	A > C > B
61A	C > A > B
61B	A > B > C

Day	CS70	CS61A	CS61B
1	A, B, C		
2	A	C	B

Assignment: (CS70, A), (CS61A, C), (CS61B, B).

(b) Run SMA with classes proposing and get the same assignment.

Day	A	B	C
1	70,61B		61A
2	70	61B	61A

For any stable assignment, if a student is paired with a class C', the student must prefer his/her optimal class to C' and prefer C' to his/her pessimal class. When students proposed, SMA outputs the student optimal assignment. When classes proposed, SMA outputs the class optimal assignment, which is the student pessimal assignment. Because the student optimal assignment is the same as the student pessimal assignment, C' can only be the class in the assignment for any student, and thus there can only be one stable assignment.

(c) Use SMA with students proposing and rule that CS70 can hold 2 proposals, with the following class rankings:

CS70	A > D > C > B
CS61A	C > A > D > B
CS61B	D > A > B > C

Day	CS70	CS61A	CS61B
1	A, B, C	D	
2	A, C	D	B

Assignment: (CS70, A and C), (CS61A, D), (CS61B, B).

(d) Follow the stability proof in the lecture note to prove that the assignment is stable. Suppose some TA T in the assignment prefers some class C^* to their assigned class C. We will argue that C^* prefers their TA(s) to T, so there cannot be a rogue couple (a class and a TA prefer each other to their current assignments). Since C^* occurs before C in T's list, he must have proposed to it before he proposed to C. Therefore, according to the algorithm, C^* must have rejected him for somebody it prefers. If C^* is not CS70, the Improvement Lemma shows C^* likes its final TA at least as much as T. If C^* is CS70, then we must now prove an alternate Improvement Lemma to show this.

Prove: If T is rejected by CS70 on the k-th day, then every subsequent day $j \geq k$, CS70 has 2 TAs whom it likes at least as much as T.

- *Base case*: On day k, CS70 rejects T, so it must prefer the two TAs it holds.
- *Induction hypothesis*: Suppose claim is true for $j \geq k$
- *Induction step*: On day $j+1$, by induction hypothesis, CS70 has 2 TAs T' and T'' it prefers to T. Either nobody proposes to CS70, or T''' proposes. If T''' is accepted, then it must be preferred over T' or T'', which are both at least as good as T, so T''' is preferred over T.

After proving the alternate Improvement Lemma, we can claim C^* likes its final TA at least as much as T. Therefore, no TA T can be involved in a rogue couple, and thus the assignment is stable.