1. Problem 3-3 part a from the CLRS text. You need not provide justifications for your order.
Due Friday January 12.

2. Problem 3-4. For conjectures that are untrue, explain which reasonable class of functions the statement is true for. For example, you might say "The statement is untrue in general, and here is an example. But the statement is true if the functions are strictly increasing, and here is why.
Due Friday January 12.

3. Consider the following definitions for $f(n, m) = O(g(n, m))$. State which definitions are logically equivalent. Fully justify your answers. Argue about which definition you think is the best one in the context where $f$ and $g$ are run times of algorithms and the input size is monotonically growing in $n$ and $m$ (for example $n$ might be the number of vertices in a graph and $m$ might be the number of edges in a graph).

(a) There exists positive constants $c > 0$, $n_0$, $m_0$ such that for all $(n, m)$ where $n \geq n_0$ and $m \geq m_0$ it is the case that $f(n, m) \leq c g(n, m)$

(b) There exists positive constants $c > 0$, $n_0$, $m_0$ such that for all $(n, m)$ where $n \geq n_0$ or $m \geq m_0$ it is the case that $f(n, m) \leq c g(n, m)$

(c) There exists a constant $c > 0$ such that

$$\lim_{n \to \infty} \lim_{m \to \infty} \frac{f(n, m)}{g(n, m)} < c$$

Note that this means that you first take the limit superior with respect to $m$. The result will be a function of just $n$. You then take the limit superior of this function with respect to $n$. If you don’t know what limit superior means, you can just assume that the limit exists, in which case the limit and limit superior are the same.

(d) There exists a constant $c > 0$ such that

$$\lim_{m \to \infty} \lim_{n \to \infty} \frac{f(n, m)}{g(n, m)} < c$$

Note that this means that you first take the limit superior with respect to $n$. The result will be a function of just $m$. You then take the limit superior of this function with respect to $m$. If you don’t know what limit superior means, you can just assume that the limit exists, in which case the limit and limit superior are the same. Note that on the surface that this definition is different than the last on in that the order that you take the limits is switched.

(e) There exists a constant $c > 0$ such that for all but finitely many pairs $(m, n)$ it is the case that $f(n, m) \leq c g(n, m)$.

Due Friday January 12.
4. Skim/Read section 16.4 in the text. Consider a hereditary set system \( M = (S, \mathcal{I}) \). Assume that for every possible collection of positive weights on the elements of \( S \), it is the case that the natural greedy algorithm (given on page 440 of the CLRS text) returns the maximum weight independent set. Show that \( M \) is a matroid.

Hint: Assume to reach a contradiction that \( M \) is not a matroid. Thus \( M \) must not have the exchange property. Let \( A \) and \( B \) be elements of \( \mathcal{I} \), where \(|A| < |B|\), that do not satisfy the exchange property. Give the elements weights so that the greedy algorithm produces \( A \). Its sufficient to have only 4 different weights. A weight \( w_1 \) for the elements of \( A \cap B \), a weight \( w_2 \) for the elements of \( A - B \), a weight \( w_3 \) for the elements of \( B - A \), and a weight \( w_4 \) for the remaining elements. Ask yourself what property you want of these weights.

Due Wednesday January 17

5. Problem 16-5 part c. You must use an exchange argument.

Due Wednesday January 17

6. Problem 4-3 from the CLRS text except parts d and f. Apply the Master Theorem (Theorem 4.1) whenever applicable. Otherwise, draw the recursive call tree and sum up the costs level by level

Due Friday January 19

7. Problem 4-3 part f. Use induction; You will need 2 inductive proofs, one for the upper bound and one for the lower bound.

Due Friday January 19

8. (a) Read/skim section 30.2 of the text. In particular make sure you note equation 30.8, Theorem 30.7 and equation 30.11. Let \( DFT(u) \) be the discrete Fourier transform of the vector \( u \), and \( DFT^{-1}(u) \) be the inverse discrete Fourier transform of the vector \( u \). The following formulas express \( DFT^{-1} \) in terms of \( DFT \). Prove each formula is correct.

i. \( DFT^{-1}(u) = DFT(\text{reverse}(u))/n. \) Here \( \text{reverse}(u) \) means you reverse the entries of the vector after the first entry, so \( \text{reverse}([1, 2, 3, 4, 5, 6]) = [1, 6, 5, 4, 3, 2]. \)

ii. \( DFT^{-1}(u) = \text{conjugate}(DFT(\text{conjugate}(u)))/n. \) Here \( \text{conjugate}(u) \) means the imaginary component of each complex number in the vector is negated, so \( \text{conjugate}([1 + 2i, 2 - 7i, 3, 4 + i]) = [1 - 2i, 2 + 7i, 3, 4 - i]. \)

iii. \( DFT^{-1}(u) = \text{swap}(DFT(\text{swap}(u)))/n. \) Here \( \text{swap}(u) \) means the imaginary and real components of each complex number in the vector are swapped, so \( \text{swap}([1 + 2i, 2 - 7i, 3, 4 + i]) = [2 + i, -7 + 2i, 3i, 1 + 4i]. \)

(b) Problem 30.2-6 from the CLRS text. Determine the running time of the FFT algorithm if one used the field of integers modulo \( m \) as suggested. You can assume arithmetic operations are implemented in the most naive/natural way that you learned in elementary school.
(c) Problem 30.2-7 from the CLRS text.
Due Monday January 22

9. Problem 8.1-3 from the CLRS text. Due Wednesday January 24

10. Problem 8.1-4 from the CLRS text. Give an adversarial strategy and prove that it is correct. Explain the last sentence of the problem statement from the book, that is, explain why it’s not rigorous/correct to simply combine the lower bounds for the individual subsequences.
Due Wednesday January 24

11. Problem 8-6 from the CLRS text.

(a) For part a and b, it is essentially asking you to consider the adversarial strategy that answers so as to maximize the number of the original ways of merging two sorted lists that are consistent with the answer. You will likely find Stirling’s approximation for $n!$ useful.

(b) For parts c and d, come up with a different adversarial strategy.

(c) Explain why the bound that you get using the method proposed in parts a and b isn’t as good as the bound you get using an adversarial strategy. That is, in what way are you being too generous to the algorithm in parts a and b? In this generosity in the adversarial strategy or in its analysis?
Due Friday January 26

12. Consider the problem of determining whether a collection of real numbers $x_1 \ldots x_n$ is nice. A collection of numbers is nice iff the difference between consecutive numbers in the sorted order is at most 1. So $1.2, 2.7, 1.8$ is nice, but $1.2, 2.9, 1.8$ is not nice since the difference between 1.8 and 2.9 is more than 1. We want to show that every algorithm, that only accesses the input through generalized comparisons, requires $\Omega(n \log n)$ generalized comparisons. A generalized comparison is of the form $x_i - x_j \leq c$, where $c$ is some constant that the algorithm can specify. So if $c = 0$, this is a standard comparison. Examples of generalized comparisons are $x_1 - x_3 \leq 4$ and $x_7 - x_2 \leq -1$.

Hint: This is similar to the lower bound for element uniqueness.
Another Hint: Consider the $(n-1)!$ permutations $\pi$ of $\{1, \ldots, n\}$ where $\pi(1) = 1$, and the corresponding points in $n$ dimensional space. Note that all $(n-1)!$ of these points are nice. Show the midpoint of any pair of these nice points is not nice. Then explain how to use this fact to give an adversarial strategy showing the $\Omega(n \log n)$ lower bound.
Due Friday January 26

13. Consider a setting where you have two computer networking routers $A$ and $B$. Each router has collected a list $L_A$ and $L_B$ of IP source addresses for the packets that have passed through the router that day. An IP address is $n$ bits, and thus there are $2^n$ possible IP addresses. Now the two routers want to communicate via a two-way channel
to whether there was some source that sent a packet through one of the routers, but not the other. So more precisely, at the end of the protocol each router should commit to a bit specifying the answer to this question, and the bits for both routers should be correct. You can assume that each router can send a bit per unit time, and that a bit sent on the channel is guaranteed to arrive on the other end in one time unit. We want to consider protocols for accomplishing this goal.

(a) (warm-up) Consider the following protocol: A sends to B the list of all of the IP source addresses that it has seen; B compares A’s list to its list, and then B sends A a 0 bit if the lists are identical and a 1 bit otherwise. Show that uses protocol about uses $n2^n + 1$ bits in the worst case.

(b) (warm-up) Give a protocol that uses $2^n + O(1)$ bits in the worst case. Another trivial warmup problem.

(c) (warm-up) Show that there is no protocol that can solve this problem without exchanging any bits.

Hint: It’s obvious that this is true. That isn’t the point. The point is to understand what arguments you have to make to make this formally correct.

(d) Show that there is no protocol that can solve this problem that involves A sending one bit to B. And no more bits are exchanged.

Hint: Again its obvious that this is true. Again that isn’t the point. Again the point is to understand what arguments you have to make to make this formally correct. Ask yourself how should the adversarial strategy should decide whether this first bit is a 0 or a 1?

(e) Show that there is no protocol that can solve this problem that involves A sending one bit to B and B replying with one bit to A. And no more bits are exchanged.

Hint: Again its obvious that this is true. Again that isn’t the point. Again the point is to understand what arguments you have to make to make this formally correct.

(f) Prove that every protocol for this problem must send $2^n - O(1)$ bits for its worst case instance. Of course your argument should involve an adversarial argument.

(g) Assume that you have a computer networking router that sees a stream of $k$ IP packets, each with a source IP address consisting of $n$ bits. The router sees a packet, optionally records some information in memory, and then passes the packet on, then sees the next packet, optionally records some information in memory, and then passes that packet on, etc. The routers’s goal is to always know a IP source address that it has seen most frequently to date. (The most obvious way to accomplish this is to keep a count for each IP source address seen to date.)

Show that if $k \leq 2^n$ then every algorithm must use $\Omega(k)$ bits of memory. 

Hint: This is an “easy” consequence of the previous subproblem, provided that you think about it the right way. Assume that you had a method that solved this problem using $o(k)$ bits of memory. Explain how to use this method to get an algorithm for the previous subproblem that uses less than $2^k$ bits of communication.
Due Monday January 29

14. Show that any comparison based algorithm for computing the median of \( n \) numbers (you can assume for simplicity that \( n \) is odd) requires \( 3n/2 - O(1) \) comparisons.

   Hint: Some variation of the lower bound to compute the largest and smallest number will work, but there are additional complications.

Due Wednesday January 31

15. Let \( P \) be a problem. The worst case time complexity of \( P \) is \( O(n^2) \). The worst case time complexity of \( P \) is \( \omega(n \log n) \). Let \( A \) be an algorithm that solves \( P \). For each of the following statements, state whether the statement is logically implied by the above information, and state whether the statement is logically consistent with the above information. Justify your answers.

   (a) \( A \) has worst case time complexity \( O(n^3) \).
   (b) \( A \) has worst case time complexity \( \Omega(n^3) \).
   (c) \( A \) has worst case time complexity \( \Theta(n^3) \).
   (d) \( A \) has worst case time complexity \( O(n \log n) \).
   (e) \( A \) has worst case time complexity \( \Omega(n \log n) \).
   (f) \( A \) has worst case time complexity \( \Theta(n \log n) \).
   (g) \( A \) has worst case time complexity \( O(n) \).
   (h) \( A \) has worst case time complexity \( \Omega(n) \).
   (i) \( A \) has worst case time complexity \( \Theta(n) \).

Due Wednesday January 31

16. Consider the problem of finding the largest \( k \) numbers in sorted order from a list of \( n \) numbers (see problem 9-1) in the text. Consider the following algorithm: you consider the numbers one by one, maintaining an auxiliary data structure of the largest \( k \) numbers seen to date. We get various algorithms depending on what the auxiliary data structure is and how one searches and updates it. For each of the following variations give the worst-case time complexity as a function of \( n \) and \( k \). For each of the following variations give the average-case time complexity as a function of \( n \) and \( k \) under the assumption that each input permutation is equally likely.

   Hint: Use linearity of expectations. These are all similar and easy if you look at them the right way.

   (a) The auxiliary data structure is an ordered list and you use linear search starting from the end that contains the largest number
   (b) The auxiliary data structure is an ordered list and you use linear search starting from the end that contains the smallest number
(c) The auxiliary data structure is a balanced binary search tree and you use standard log time search, insert and delete operations.

(d) The auxiliary data structure is a balanced binary search tree and you use standard log time insert and delete operations, but you start your search from the smallest item in the tree.

Due Wednesday January 31

17. Assume a router sees a stream of IP packets from two different sources. So the router sees a packet, and then can do some minimal computation, then forwards the packet, sees the next packet, etc. The router is trying to determine the similarity of the destination IP addresses for the two different sources while using very little space. Let $A$ be the collection of destination IP addresses for the first source while using very little space, and $B$ be the collection of destination IP addresses for the second source. Assume that we have a uniform hash function $h$ that maps IP addresses to integers, where the range is sufficiently large that the probability of a collision is negligible.

(a) First consider a naive approach. Let $a$ be a random element of $A$ and $b$ a random element $B$. If $A = B$, what is the probability $h(a) = h(b)$? If $A$ and $B$ are disjoint, what is the probability that $h(a) = h(b)$? Calculate the probability that $h(a) = h(b)$ in terms of $|A|$, $|B|$, $|A \cup B|$, and $|A \cap B|$.

(b) Now we turn to something a bit more sophisticated. Let $h_m(A)$ be the minimum integer $k$ such there is an element $x$ of $A$ where $h(x) = k$. Let $h_m(B)$ be the minimum integer $k$ such there is an element $x$ of $B$ where $h(x) = k$. If $A = B$, what is the probability that $h_m(A) = h_m(B)$? If $A$ and $B$ are disjoint, what is the probability that $h_m(A) = h_m(B)$? Remember that we are assuming that the probability of a collision is negligible.

(c) Calculate the probability that $h_m(A) = h_m(B)$ in terms of $|A|$, $|B|$, $|A \cup B|$, and $|A \cap B|$.

(d) Explain how the probability that $h_m(A) = h_m(B)$ is an estimate of the similarity of $A$ and $B$.

(e) Explain how to maintain $h_m(A)$ and $h_m(B)$ using constant space, and constant time per IP packet.

Due Friday February 2

18. Assume you have a source of random bits. So in one time unit, this source will produce one random bit (that is 1 with probability 1/2 independent of other bits). Consider the problem of outputting a random permutation of the integers from 1 to $n$. So each of the $n!$ permutations should be produced with probability exactly $1/n!$.

(a) Give an algorithm to solve this problem and show that the expected time of the algorithm is $O(n \log n)$. This includes both the time that your algorithm takes, plus 1 unit of time for each random bit used.
(b) Now assume that there is a limited source of at most \( n^2 \) random bits. Show that there is no algorithm that can solve the problem using expected time \( O(n^2) \).

Hint: Show the result for \( n=3 \). Why can’t you produce a random permutation of 1, 2, 3 using 9 bits? Then generalize to an arbitrary \( n \).

Further Hint: Often students will say that there two statements are contradictory. Maybe start by understanding why they are not contradictory.

Due Monday February 5

19. Consider the following problem. The input is \( n \) disjoint line segments contained in an \( L \) by \( L \) square \( S \) in the Euclidean plane. The goal is to partition \( S \) into convex polygons so that every polygon intersects at most one line segment. So it is ok for a line segment to be in multiple polygons, but each polygon can intersect at most one line segment.

Consider the following algorithm that starts with the polygon \( S \). Let \( \pi \) be a random permutation of the line segments. While there is a polygon \( P \) that contains more than one line segment, let \( \ell \) be the first line segment in the \( \pi \) order that intersects \( P \). Then cut \( P \) into two polygons using the linear extension of \( \ell \) (so you extend the line segment \( \ell \) into a line and then use that to cut \( P \) into two polygons). Show that the expected number of resulting polygons is \( O(n \log n) \).

Hint: Use linearity of expectations. First ask yourself how the number of polygons is related to the number of times that line segments get cut in the process. Consider to line segments \( u \) and \( v \). Let \( C_{u,v} \) be a 0/1 random variable that is 1 if the linear extension of \( u \) cuts \( v \). Let \( N(u,v) \) denote the number of line segments \( w \) where the linear extension of \( u \) hits \( w \) before hitting \( v \). In other words if you starting walking from \( u \), on \( u \)'s linear extension, towards \( v \), \( N(u,v) \) is how many line segments you cross before hitting \( v \). If you don’t hit \( v \), then \( N(u,v) = +\infty \). What is the relationship between the probability that \( C_{u,v} = 1 \) and \( N(u,v) \)?

Due Wednesday February 7

20. Consider a situation where a router sees over time \( n \) packets with source IP addresses \( x_1, \ldots, x_n \). The router wants to keep track of an estimate of how many packets it has seen from each source IP address, but wants to use less space than the number of IP sources it has seen. Assume that the router has \( t \) independent has functions \( h_1, \ldots, h_t \), each with a range of \([1,k]\), and maintains a table \( T \) of size \( t \) by \( k \). In response to a packet with source IP address \( s \), table entries \( T[j, h_j(s)] \), \( 1 \leq j \leq t \), are incremented. Let \( f_s \) be the number of times that the router has seen a packet with IP source \( s \). The router will use as its estimate \( \hat{f}_s \) of \( f_s \) as \( \min_{j \in [1,t]} T[j, h_j(s)] \). Our goal is this problem is to prove that \( \hat{f}_s \) is a reasonable estimate of \( f_s \). More precisely we want to show that for all \( \epsilon > 0 \) and all \( \delta > 0 \), \( \text{Prob}[\hat{f}_s - f_s \geq \epsilon n] \leq \delta \), provided \( t \) and \( k \) are appropriately selected.

(a) Explain why \( \hat{f}_s \geq f_s \).
(b) Let $Y_{i,j}$ be a random variable that is equal to 1 if $h_i(s) = h_i(x_j)$ and $s \neq x_j$, and 0 otherwise. So this random variable is the excess caused by $x_j$ to the $i$th estimate for $s$. Calculate $E[Y_{i,j}]$.

(c) Let $Y_i = \sum_j Y_{i,j}$. Show using linearity of expectations that the $E[Y_i] = (n-f_s)/k$.

(d) Show that $\text{Prob}[Y_i \geq \epsilon n] \leq (n-f_s)/\epsilon nk$.

(e) Show that if $k = 2/\epsilon$ then $\text{Prob}[Y_i \geq \epsilon n] \leq 1/2$.

(f) Show that if $t = \log 1/\delta$ then $\text{Prob}[\min_i Y_i \geq \epsilon n] \leq \delta$.

(g) Explain why $\hat{f}_s - f_s \leq \min_i Y_i$. Thus $\min_i Y_i$ is an upper bound on the absolute error of estimating $f_s$ by $\hat{f}_s$.

(h) Explain why this establishes the desired result.

Due Friday February 9

21. 11-2 from CLRS.

Due Monday February 12

22. We consider the packet routing problem that we considered in class. See also [http://people.cs.pitt.edu/~kirk/cs2150/LMR.pdf](http://people.cs.pitt.edu/~kirk/cs2150/LMR.pdf)

Let

- $A$ be a yet to be defined randomized algorithm for this problem
- $n$ be the number of paths/packets
- $D$ be the length of the longest path
- $C$ be the maximum number of paths that share any one edge
- $Y_i$ be the time that packet $i$ arrives for algorithm $A$. So $Y_i$ is a random variable.
- $Y = \max_i Y_i$. $Y$ is the makespan for the algorithm.
- Let $F_{e,t}$ be the number of packets waiting to cross edge $e$ at time $t$ using $A$. So $F_{e,t}$ is a random variable.
- Let $F = \max_{e,t} R_{e,t}$. So $F$ is a random variable.

Now we want to analyze $A$ using the following strategy.

(a) Under the assumption that the worst makespan for $A$ is $CD$, show that $E[Y] \leq 20L$ if $\text{Prob}[F \geq L/D] \leq 1/(CDn)^6$.

(b) Explain why $\text{Prob}[F \geq L/D] \leq \sum_e \sum_t \text{Prob}[F_{e,t} \geq L/D]$.

(c) Assume for the moment that the capacity of each edge is infinite instead of 1. Now assume that each packet is delayed by an amount of time that is selected uniformly and independently from the range $[0, C]$ before it starts moving. Once a packet starts moving, it traverse one edge per unit time, reaching its destination within $D$ steps from when it starting moving, and thus its makespan is at most $C+D$. Fix an edge $e$ and a time $t$. Let $x_i$ be the 0/1 random variable that is equal to 1 if packet $i$ cross edge $e$ at time $t$. By renumbering, assume that $i \in [1, C]$. Note that $F_{e,t} = \sum_{i=1}^{C} x_i$. 

i. Explain why \( x_1, x_2, \ldots, x_C \) are Bernoulli trials.

ii. Using the Chernoff bound of your choice from [https://en.wikipedia.org/wiki/Chernoff_bound](https://en.wikipedia.org/wiki/Chernoff_bound) show that 
\[
\Pr[F_{e,t} \geq \frac{L}{D}] \leq (CDn)^{12} \text{ if } L = 10000D \log CDn.
\]

iii. Explain why this means that 
\[
\Pr[F \geq \frac{L}{D}] \leq \frac{1}{(CDn)^6}.
\]

(d) Now explain how to obtain a randomized algorithm \( A \) with the property that 
\( E[Y] \leq 2L \) when the edge capacities are unit.

(e) Now explain how modify \( A \) so that it is still the case that 
\( E[Y] \leq 2L \) when the edge capacities are unit, but additionally \( A \) has the property that it will never unnecessarily delay a packet. So if only one packet wants to use a particular edge at a particular time, that packet must traverse that edge at that time.

Due Wednesday February 14

23. (a) Consider the problem where the input is a sorted array \( A \) containing \( n \) real numbers and a real number \( x \), and the output is an integer \( k \) such that if \( x \) is in \( A \) then it must lie between positions \( k \) and \( k + \sqrt{n} \). Assume that each element of \( A \) and \( x \) are independently and uniformly distributed in the interval \([0, 1]\). Show that the following algorithm solves this problem in \( O(1) \) average case time.

```plaintext
last = x*n
if A[last] < x then
    next = last + sqrt(n)
    while A[next] < x do
        last = next
        next = next + sqrt(n)
    return k = last
else if A[last] > x then
    next = last - sqrt(n)
    while A[next] > x do
        last = next
        next = next - sqrt(n)
    return k = next
```

HINT: Find the Bernoulli trials. Figure out how to think about the outcome of this algorithm in terms of the number of successes/failures in some Bernoulli trials. Use a Chernoff tail bound. See appendix C.5. You can use the result of exercise C.5-6 without proof.

(b) Explain how to use the above algorithm to obtain an algorithm with \( O(\log \log n) \) average case running time for the searching problem (finding the location of the element in \( A \) whose value is closest to \( x \)). Again assume that each element of \( A \) and \( x \) are independently and uniformly distributed in the interval \([0, 1]\).

Due Friday February 16
24. The purpose of this problem is to develop a version of Yao’s technique for Monte Carlo randomized algorithms, within the context of the red and blue jug problem from problem 8-4 in the CLRS text. Assume that if you sorted the jugs by volume, that each permutation is equally likely.

(a) Show that if a deterministic algorithm A always stops in $o(n \log n)$ steps, then the probability that A is correct for large $n$ is less than 1 percent.

(b) Show if there is a distribution of the input on which no deterministic algorithm with running time $A(n)$ is correct with probability $>1$ percent, then there is no Monte Carlo algorithm with running time $A(n)$ that can be correct with probability $>1$ percent.

Hint: Mimic the proof of Yao’s technique/lemma for the case of Las Vegas algorithms. Consider a two dimensional table/matrix $T$, where entry $T(A, I)$ is 1 if algorithm A is correct on input $I$, and 0 otherwise.

(c) Conclude that any Monte Carlo algorithm for this jug problem must have time complexity $\Omega(n \log n)$.

Due Monday February 19

25. Consider the following online problem. You given a sequence of bits $b_1,...b_n$ over time. Each bit is in an envelope. You first see the envelope for $b_1$, then the envelope for $b_2$, .... When you get the $i^{th}$ envelope, you can either look inside to see the bit, or destroy the envelope (in which case you will never know what the bit is). You know a priori that at least $n/2 + 1$ of the bits are 1. You goal is to find an envelope containing a 1 bit. You want to open as few envelopes as possible.

(a) Give a deterministic algorithm that will open at most $n/2 + O(1)$ envelopes.

HINT: This is completely straight-forward.

(b) Give an adversarial strategy to show that every deterministic algorithm must open at least $n/2 - O(1)$ envelopes.

HINT: This is completely straight-forward.

(c) Assume that each of the $n!$ permutations of the inputs is equally likely. Show that there is a deterministic algorithm where the expected number of envelopes that is opens is $O(1)$.

HINT: This is a straight-forward consequence of some facts that we learned about Bernoulli trials.

(d) Give a Monte Carlo algorithm that opens $O(\log n)$ envelopes and has probability of error $\leq 1/n$. Show that the probability of error is this small.

HINT: This is a straight-forward consequence of some facts that we learned about Bernoulli trials.

(e) Show using the version of Yao’s technique for Monte Carlo algorithms that you developed in the last homework assignment to show that every Monte Carlo algorithm must open $\Omega(\log n)$ envelopes if it is to be incorrect with probability $\leq 1/n$. 

10
HINT: This is a straight-forward application of the Yao’s technique for Monte
Carlo algorithms that you developed in the previous homework problem.

(f) Give a Las Vegas algorithm where the expected number of opened envelopes is
$O(n^{1/2})$.

Hint: Take some random guesses for the first half of the envelopes, and then if
you don’t find a 1 bit, give up and do the most obvious thing. See the discussion
of the Birthday paradox in section 5.4.1. You may use facts from the analysis
of the Birthday paradox in the CLRS text or from the wikipedia

Due Wednesday February 21

26. Show that every Las Vegas algorithm for the previous envelope problem must open
$\Omega(n^{1/2})$ envelopes in expectation.

Hint: Use Yao’s technique and the following probability distribution. With probability
half, $\sqrt{n}$ uniformly distributed random bits in envelopes $[1, n/2]$ are set to 1 (the other
bits in the first half of the envelopes are set to 0), bits in the envelopes $[n/2+1, n/2+\sqrt{n}]$
are all set to 0, and the remaining bits are 1. For $k = 0, \ldots, \sqrt{n}$, with probability
$1/(2\sqrt{n})$, the bits in envelopes are distributed according to distribution $D_k$. In $D_k$,
envelopes $[1, \ldots, n/2]$ contain a uniformly distributed random set of $k$ 0’s and $n/2-k$
1’s. Then envelopes $[n/2 + 1, n/2 + \sqrt{n} + 1]$ contain a uniformly distributed set of
$\sqrt{n} - k$ 0’s and $k + 1$ 1’s. The remaining bits are 0.

This is one of the hardest homework problems of the semester. Feel free to consider
this to be extra credit if you like.

Due Wednesday February 21

27. There are three shortest path algorithms covered in chapter 24 (Bellman-Ford, Dijkstra,
and the topological sort algorithm for directed acyclic graphs). For each of the following
problems, pick the most appropriate of these three shortest path algorithms to apply
to obtain an algorithm for the problem. If it is debatable which algorithm is most
appropriate, state the arguments for and against each reasonable candidate for most
appropriate algorithm. This may or may not involve modifying the algorithm slightly.
If you need to modify the algorithm, explain how. You may need to first briefly explain
why the problem is indeed just a shortest path problem in disguise; That is, state how
one obtains the graph, and why the shortest path in this graph corresponds to a
solution to the problem. Give the running time of the resulting algorithm. Recall that
the running time of topological sort algorithm is $O(E)$, the running time of Bellman-Ford
is $O(EV)$, the running time of Dijkstra’s algorithm is $O(E \log V)$ if a standard
heap is used (this can be improved to $O(E + V \log V)$ if a Fibonacci Heap is used, but
assume for this problem that a standard heap is used).

(a) The problem described in 24-2
(b) The problem described in 24-3
(c) The problem described in 24-6
(d) The problem of finding the path where the minimum edge weight is maximized. You need such an algorithm to implement one of the Karp-Edmonds variations on Ford-Fulkerson.

(e) The input is a directed graph \( G \), with a designated source vertex \( s \) and a designated destination vertex \( t \). The edges of \( G \) are labeled with distances, and the vertices of \( G \) are labeled with elevations. The problem is to find the shortest downhill path from \( s \) to \( t \). A path is downhill if each vertex on the path is lower in elevation than the previous vertex on the path.

(f) The input is a list of flights, a source city \( s \), a specified start time \( z \), and a destination city \( t \). Each flight is a 4-tuple consisting of (1) a takeoff time, (2) the city the flight leaves from, (3) a landing time, and (4) the city at which the flight lands. The output is the earliest possible time that one can arrive at destination city \( t \), starting from source city \( s \), and leaving not earlier than time \( z \), by taking a sequence of these flights, subject to condition that the layover in each city has to be at least 1 hour.

Due Friday February 23


Hint: Using exhaustive search plus pruning to design a dynamic programming algorithm. First figure out how to exhaustively generate all feasible solutions as the leaves of some tree. Then construct a pruning rule that you apply to each level of the tree to prune out some subtrees rooted at this level. The natural pruning rule is similar to the one used by Bellman-Ford.

Due Monday February 26

29. There are two options for this problem. Which option you pick depends on your background. If you have not taken CS 1510 from me and I am not your Ph.D. adviser, then you may pick the first option (and and you may consider the second option extra credit). If you have taken CS 1510 from me, or I am your Ph.D. adviser, you must do the second option.

(a) The input is a sequence of integers \( x = x_1, \ldots, x_n \). A subsequence \( y \) of \( x \) is increasing if each number of \( y \) is larger than the previous number in \( y \). The length of subsequence is the number of numbers in the subsequence. The output should be the longest increasing subsequence of \( x \).

Hint: Using exhaustive search plus pruning to design a dynamic programming algorithm. First figure out how to exhaustively generate all feasible solutions as the leaves of some tree. Then construct a pruning rule that you apply to each level of the tree to prune out some subtrees rooted at this level. The natural pruning rule is sort of similar to the one used by Bellman-Ford.

(b) Let \( D[1 \ldots n] \) be an array of digits, each an integer between 0 and 9. An digital subsequence of \( D \) is a sequence of positive integers composed in the usual way.
from disjoint substrings of $D$. For example, 3, 4, 5, 6, 8, 9, 32, 38, 46, 64, 83, 279 is a digital subsequence of the first several digits of $\pi$:

$$3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6, 2, 6, 4, 3, 8, 3, 2, 7, 9$$

The length of a digital subsequence is the number of integers it contains, not the number of digits; the preceding example has length 12. A digital subsequence is increasing if each number is larger than its predecessor. Describe and analyze an efficient algorithm to compute the longest increasing digital subsequence of $D$. Note that it takes $k$ units of time to compare two $k$ digit integers.

For full credit, your algorithm should run in $O(n^4)$ time. Faster algorithms are worth extra credit. The fastest algorithm known for this problem runs in $O(n^2 \log n)$ time. But achieving this bound requires several tricks, both in the algorithm and in its analysis.

Due Monday February 26