Randomized Algorithms:

1. Consider the randomized algorithm that we discussed in the class for Max-3 SAT. Basically, it worked as follows: for each boolean variable x_i, it tossed a *fair* coin and if it came (say) heads – it set x_i to 1. Else it was set to 0. The availability of a fair coin in fact central to the correctness of the algorithm. Suppose, you did not have a fair coin with you. Instead you have a biased coin (which for eg, gives heads with probability p and tails with probability 1-p with p some value other than 0.5). Design a simple algorithm that allows you to simulate a fair coin from a non fair coin.

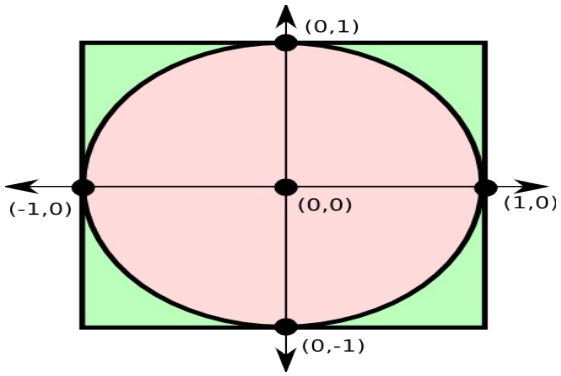
2. Your friend has an unfair coin that comes head with probability p and tails with probability 1-p. She proposes the following betting game. If it comes head, she will give you $10. If it comes tails, you have to give her $5. For what values of p would you accept this bet?

3. We discussed a randomized approximation algorithm for Max 3-SAT with an approximation factor of 7/8. Consider a similar problem called Max k-SAT where each clause has k distinct variables. If you use the same algorithm for Max k-SAT what approximation factor would you get?

4. Here is a cool Monte-Carlo algorithm that we didn't discuss in the class: Draw a square with radius 2 units. In the middle of the square, draw a circle with radius 1. Your figure should look something like this. Now close your eyes and throw 1000 pins (larger the better) into this figure *randomly* (dont try to be smart – just throw randomly). Assuming all your pins fell into the figure, let n1 be the number of figures that fell into the circle. An interesting fact is that (4*n1)/n is approximately equal to Pi!

Of course, if you are busy in the exam week, here is a simple way to code this algorithm. Simply generate many random 2D points (x,y) in the box [-1,1]x[-1,1], and count the number of points within distance the circle (n1) and the number of points in the box total (n). You can approximate Pi as 4*n1 / n.

Why do you think this trick works?



Dynamic Programming:

1. Consider a simple variant of the graph coloring problem. You are given a tree and you wish to color some of the nodes to, say green. Of course, if a node is colored green none of its neighbor can be colored green. Given a tree, design an efficient algorithm to find the maximum number of nodes that could be colored green.

2. Suppose you have elements ranging from 1 to n. How many distinct BSTs could be constructed using them? For eg, n = 2, there are two ways (1 as root and 2 as root). If n=3, there are 5 trees (two trees with 1 as root, one tree with 2 as root and two trees with 3 as root). Design a DP algorithm to count the number of distinct BSTs.

3. Suppose you are given a Directed Acyclic Graph (DAG) with positive weights. Design an efficient algorithm for single source shortest path in graphs. Of course, it has to be much faster than Dijkstra's algorithm.

4. In UTA, you all must have had an experience in ERB where people pressed all floors from 2 to 6 making the elevator ride look like a bus ride with numerous stops. Suppose you work in building with 100 floors all of which are occupied. Imagine what happens when people enter lot of random floors! So you come with a clever idea of batching where the elevator stops at at most k floors in each ride. If your floor is not one of the k, you have to get out and walk to your destination floor.

So here is the abstract problem. You are given an array with n elements. Of course the array can be have duplicate elements and n can be smaller, equal or larger than k. Assume that the elevator always starts at the ground floor and that each number in the array is between 1 and 100. Given this array, decide the k floors in which the elevator stops such that the total number of floors that people have to walk up/down is minimized.

5. Suppose you were given a text document where white spaces and punctuations are removed. For eg, “UTA is in Texas” might look like “ UTAisinTexas”. Suppose you are given a dictionary that can say if a word w is in it in O(1) time. Design an efficient algorithm (hint: O(n^2)) to recover a valid sequence. Of course, if there are multiple possible ones (eg, grammar and gram mar are both valid sequences if the input is grammar).

6. Suppose you have a car. You can do three operations during the beginning of each year. Buy a new one, maintain it or trade it . Buying a new machine costs c_1 dollars. Maintaining the machine during -i-th year of operation takes m_i dollars. For kicks, let us assume that you can use a car for at most 3 years. So only m_1, m_2, m_3 are non negative. Similarly, if you trade your car after i-years, you will get a value of t_i. Of course, you can trade only in the first three years. Given c_1, m_1, m_2_m_3 and t_1, t_2 and t_3, design an efficient strategy to minimize your expenses for the next 5 years. Since you are in Arlington, TX – that largest city in US without public transport - you always need a car.

Miscellaneous

1. [2 pt] Consider an undirected graph with n vertices, and m edges. Assume that the edges are of two types: m_1 red edges and m_2 green edges. Thus $m = m_1 + m_2$. The red edges have weight 1, and the green edges have weight 2. Design and analyze an efficient algorithm to compute **single source shortest paths** in such graphs.
2. [2 pt] Design and analyze an algorithm to find the **second shortest path** between two given vertices u and v in any weighted undirected graph.
3. [2 pt] Design and analyze an algorithm to find the **third shortest path** between two given vertices u and v in any weighted undirected graph.
4. [2 pt] The **diameter** of a weighted undirected graph is defined as the length of the shortest path between the pair of vertices that are the furthest apart. Design and analyze an efficient algorithm to compute the diameter of a graph.
5. Design a small graph of 5 nodes that is not a tree, such that the minimum spanning tree is always the same as the shortest path tree (no matter the starting node)