CS 6150: HW0 – Introduction and background

Submission date: Friday, August 22, 2025, 11:59 PM

This assignment has 6 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Big oh and running times	10	
Square vs. Multiply	5	
Graph basics	8	
Background: Probability	12	
Tossing coins	7	
Array Sums	8	
Total:	50	

Question 1: Big oh and running times ..... **[10]**

(a) **[4]** Write down the following functions in big-oh notation:

1. $f(n) = n^2 + 5n + 20$.

$$\mathbf{O(n^2)}$$

2. $g(n) = \frac{1}{n^2} + \frac{2}{n}$.

$$\mathbf{O(\frac{1}{n})}$$

(b) **[6]** Consider the following algorithm to compute the GCD of two positive integers a, b. Suppose a, b are numbers that are both at most n. Give a bound on the running time of $\textsc{Gcd}(a, b)$. (You need to give a formal proof for your claim.)

Algorithm 1 $\textsc{Gcd}(a, b)$

```
if (a < b) return Gcd(b, a);
if (b = 0) return a;
return Gcd(b, a%b); (Recall: a%b is the remainder when a is divided by b)
```

Claim: The runtime of the algorithm in the worst case is O(log(n))

Obsv: It is known that the worst case input for the euclidean algorithm is two fibonacci numbers follwed by one another such that $a > b$

Obsv: When doing the modulo operation if your quotient is 1 youo will have the most iterations out of any other quoteint. This is because when doing modulo the value returned is r in the following equation: $a = q * b + r$ s.t. $0 \leq r < b$. The value of r is the largest when the value of q is the smallest.

Obsv: With the logic above it is easy to think that trying to find numbers with a quotient of 0 would be ideal. This is the truth but in the euclidean algorithm you can only have this happen once because after it happens the values are swapped

Obsv: The worst case of the algorithm happens when we are able to have a quoteint of 1 at every single modulo operation. The only way we can do this is if we have both a and b be fibonacci numbers

Obsv: The runtime of this worst case has to be $log(n)$. This is because when using fibonacci numbers you will always cut your number in half. This is inherint to how the fibonacci sequence works. The $f_(k-1)$ value always makes up more than half of the f_k value. We are also told that both a and b are of size N

Therefore: The runtime of the euclidean algorithm in the worst case is $O(log(n))$

Question 2: Square vs. Multiply ..... **[5]**

Suppose I tell you that there is an algorithm that can square any n digit number in time $O(n \log n)$, for all $n \geq 1$. Then, prove that there is an algorithm that can find the product of *any two* n digit numbers in time $O(n \log n)$. [*Hint:* think of using the squaring algorithm as a subroutine to find the product.]

Claim: The algorithm shown below can compute the product of any two n digit numbers in time O(nlog(n))

Algorithm 2 multiply(a, b)

```
result = (a + b)^2 − (a − b)^2;
return result ÷4;
```

Obsv: To show that our algorithm is correct all we have to do is expand the value of "result and work through the algebra

$$(a+b)^2 = a^2 + 2ab - b^2$$
$$(a-b)^2 = a^2 - 2ab + b^2$$
$$(a+b)^2 - (a-b)^2 = 4ab$$
$$4ab/4 = ab$$

Obsv: Now that correctness has been proven we can look at the running time. We know that the addition and subtraction of two numbers is consant so this doesn't have any effect on our final run time. We were also given the runtime of our squaring algorithm as $O(n \log n)$. We also know that because 4 is a power of two we can do constant time division by simply shifting the bits.

Therefore: Our final runtime of the algorithm can be expressed as $2n \log n$ which in big o notation would simply be $O(n \log n)$

Question 3: Graph basics ..... **[8]**

Let G be a *simple*[1] undirected graph. Prove that there are at least two vertices that have the same degree. **Claim: There are at least two vertices that have the same degree for a simple undirected graph**

Assumption: Two vertices must exist in the graph from the problem definition

Obsv: Each vertice can have degree $[0, (n-1)]$ vertices. This is because the graph is simple (no cycles and each vertice is connected to each other vertice at most 1 time. This would mean there would be n unique degrees but we realize that if a vertice has $n-1$ degrees (connected to every other element) then it isn't possible to have a vertice with degree 0. So we can either have $[1, (n-1)]$ or $[0, (n-2)]$. This means we have $n-2$ possible "unique" degrees)

Obsv: With the observation above we can apply the pigeonhole principle. Because we have n vertices and $n-1$ "unqiue" degrees, at least two vertices must share the same degree

Question 4: Background: Probability ..... **[12]**

(a) **[3]** Suppose we toss a fair coin k times. What is the probability that we see heads precisely once? **To solve this we realize that there will be k "slots" one for each coin flip. For each toss we have a 1/2 probability of getting what we want (heads or tails) After that we simply need to decide which "slot" we want the heads to be in using a combination**

$$(\frac{\mathbf{1}}{\mathbf{2}})^{\mathbf{k}} * \mathbf{k}$$

(b) **[4]** Suppose we have k different boxes, and suppose that every box is colored uniformly at random with one of k colors (independently of the other boxes). What is the probability that all the boxes get distinct colors? **To solve this problem we can think about how many valid choices we have at each**

$$\frac{\mathbf{k!}}{\mathbf{k^k}}$$

(c) **[5]** Suppose we repeatedly throw a fair die (with 6 faces). What is the expected number of throws needed to see a '1'? How many throws are needed to ensure that a '1' is seen with probability $> 99/100$? **The first part of this question requires us to reliaze that this is a geoemtric random variable (success: we see a 1, failure: we see anything else) The expected value of a gemoetric series can be found from $\frac{1}{p(success)} = \frac{1}{1/6} = 6$ The expected number of throws needed to see a 1 are 6 because the probability of seeing a 1 are $\frac{1}{6}$ We can use**

[1] I.e., there are no self loops or multiple edges between any pair of vertices.

the complement to solve. No 1 appears in n throws. For each independent trial the probability of success is $\frac{5}{6}$. This means that the probability of seeing no 1's over n trials is $\frac{5}{6}^n$. We are solving with the complement so we can take $1 - \frac{5}{6}^n > \frac{99}{100}$. Simplifying yields $0.01 > \frac{5}{6}^n$. Take the natural log of each side and solve for n to get $n > 26$

Question 5: Tossing coins ..... **[7]**

Suppose we have two coins, one of which is *fair* (i.e. prob[heads] = prob[tails] = 1/2), and another of which is slightly biased. More specifically, the second coin has prob[heads] = 0.51. Suppose we toss the coins N times, and let H_1 and H_2 be the number of heads observed (respectively).

(a) **[3]** Intuitively, how large must N be, so that we have $H_2 > H_1$ with "reasonable certainty"?

(b) **[2]** Suppose we pick $N = 25$. What is the expected value of $H_2 - H_1$?

(c) **[2]** Can you use this to conclude that the probability of the event $(H_2 - H_1 \geq 1)$ is small? [It's OK if you cannot answer this part of the problem.]

Question 6: Array Sums ..... **[8]**

Given an array $A[1 \ldots n]$ of integers, find if there exist indices i, j, k such that $A[i] + A[j] + A[k] = 0$. Can you find an algorithm with running time $o(n^3)$? [NOTE: this is the little-oh notation, i.e., the algorithm should run in time $< cn^3$, for any constant c, as $n \to \infty$.] [*Hint:* aim for an algorithm with running time $O(n^2 \log n)$.]