Chapter 1

Lecture 1

When does it matter what Algorithm you use? There are many Algorithms to go from a given problem to a given solution.

Example.

> To solve multiplication, we can just draw a grid of dimensions nxm and count all the tiles. We can also use the elementary school algorithm and compute in $O(n^2)$ which is goodish . In 2019, an algorithm was made that solves in $O(nlogn)$ time.

Example.

> Adding all integers from 1 to n can be done by individually adding but also by using the formula $\frac{n(n+1)}{2}$ which is much faster

- Small Runtime
- Small Memory
- Correct
- General
- Checkpointing
- Parallelizes well
- Quantum Computable?

A good algorithm for this class has a small runtime. Why do we focus so much on runtime? If the algorithm does not have a small runtime. other qualifications won't matter as much. Usually, faster runtime will check off the other boxes as well.

Worst case asymptotic runtime (Big O)

Definition: Asymptotic

We don't care about what algorithm we use when the input is small. It is probably efficient enough to use brute force at that point.

Definition: Worst Case

Average case sounds amazing, but it doesn't work when we do not know the distribution of the data. We would otherwise be assuming uniform distribution. It is good to use average case in the instance when we do know this variable, otherwise we stick with worst case.

What is a good worst case asymptotic runtime?

$$O(polytime)$$

For this class, an efficient algorithm is simply polynomial time. The order goes $O(logn)$ $O(n)$ $O(nlogn)$ $O(n^k)$ $O(2^n)$ $O(n!)$, but this is not techically true for all cases as coefficient constants could be extremely small or large. However, no algorithms really have such large coefficients. $O(n^{100})$ is technically polytime but is just as bad as $O(2^n)$. In this class the worst we will work with is $O(n^4)$.

Example.

> Primality, given an integer x ¿ 2. Assume that divisibility is O(1). Is this O(n)? Is this linear time?

```
for k = 2 to x - 1
    Test if x|k
    If yes return not prime
return prime
```

Linear time means $O(n)$, where n is the size of the input. More specifically, it is the number of bits to write down input. So the answer would depend on if it is bits or literal input. The anwer is $O(x)$ but not $O(n)$ The size of the input is $log_2 x$ because of the number of bits, meaning that in terms of the input, the algorithm is

$$O(2^{log_2 x}) = O(x) = O(2^n).$$

Therefore, the algorithm is not linear.

1.1 When do we use bits?

If the input is in numbers, the input size is $\sum logn$ of the inputs. Usually, this won't matter because big O will be in terms of a variable in the input.