Complexity

Measuring Complexity

1. Just Run a program and see
2. Count the number of instructions it would require
3. Come up with an approximate technique
   - We have to implement an algorithm
   - It might take different amounts for different inputs
   - It will probably be faster on faster computers

Experiment 1:

```python
def count2sum(a):
    N = len(a)

    count = 0
    for i in range(N):
        for j in range(N):
            if a[i] + a[j] == 0:
                count += 1
    return count
```

Experiment 2:

```python
def counteven(N):
    count = 0
    for i in range(1, N + 1):
        if i % 2 == 0:
            count += 1
    return count
```

Experiment 3:

```python
def sum3(a):
    N = len(a)

    count = 0
    for i in range(N):
        for j in range(i + 1, N):
            for k in range(j + 1, N):
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    return count
```

Experiment 4:

```
def counteven(N):
    # Half the numbers will be even
    # The other half are odd
    count = N//2
    return count
```

Classifications

- We can classify algorithms based on their performance
  1. Quadratic
     - Runtime proportional to N^2
  2. Linear
     - Runtime proportional to N
  3. Cubic
     - Runtime proportional to N^3
  4. Linear
     - Runtime is proportional to 1

Constant Factors

- Generally constant factors are not important, that is one quadratic algorithm could be twice as fast as another
  - They are both still quadtratic
  - We are interested in the performance relative to N

Lower terms

- The runtime could be proportionals to N^2 + 4N + 20
  - We ignore the N and the 20 becuase there is N^2 which is going to dominate as N gets large
  - Just take the highest power as it is what will dictate the runtime as N increases

---

Big O Notation

- Big O Notation is used to classify running times
  - O(N^2) - Quadratic
  - O(N^3) - Cubic
  - O(N) - Linear
  - O(1) - Constant

Worst Case

- In theory, Big O Notation is for the worst case, so saying an algorithm is O(N^3) means it will not be worse than O(N^3)
  - You could say that an O(1) algorith is O(N^3)
  - You could not say that an O(N^3) algorithm is O(1)

Algorithm Classes

O-Notation	Description	Speed	n = 1000	N = 1000000
O(1)	Constant	Instant	10^-9 secs	10^-9 secs
O(n)	Linear	Fast	10^-6 secs	10^-3 secs
O(N^2)	Quadratic	Slow for Large N	10^-3 secs	10^3 secs
O(N^3)	Cubic	Slow for medium N	1 second	10^9 secs

Examining Code

- It's easier if to not have to run an experiment everytime you run a program
- Generally, everytime we go over an input, the runtime is multiplied by N.
  - If we examine the input once, the algorithm will be O(N)
  - If for each item in the input, we check every other item in the input, then the algorithm is O(N^2)

Logarithmic and Exponential

- There are two more important classes of algorithm
  1. Logarithmic
     - The input is split in half every time (e.g. Binary Search)
  2. Exponential
     - The runtime doubles each time N increases by 1

Logarithmic

- Very Fast
  - Sample values:

N	Value
1000	10
1000000	20
1000000000	30

The log of a number rises very slowly, Doing a binary search on an array of 1000000 elements is only 20 times longer than a binary search of an array of 1 element.

N*Log(N)

- Nlog(N), O(N log N), N log(N)
  - Linearithmic
    - very close to linear

Exponential

- 2^N
  - The runtime doubles each time N is increased by 1

N	Value
10	1000
20	1000000
30	1000000000

- When N is 30, the operation takes the order of a second. When N is 40 it takes 1000 seconds (15 minutes)
- N = 50 requires 1000 * 15 minutes (approx 250 hours or 10 days)

O-Notation	Description	Speed	~Time for n=1000	~time for n=1000000
O(1)	Constant	Instant	10^-9 secs	10^-9 secs
O(logN)	Logarithmic	Very Fast	10^-8 secs	10^-8 secs
O(N)	Linear	Fast	10^-6 secs	10^-3 secs
O(NlogN)	Linearithmic	Fast	10^-5 secs	10^-3 secs
O(N^2)	Quadratic	Slow for Large N	10^-3 secs	10^3 secs
O(N^3)	Cubic	Slow for Medium N	1 second	10^9 secs
O(2^N)	Exponential	Extremely Slow	10^301 secs	-> ∞

Types of Data

- The runtime complexity can depend on the actual data
  - Not just the size on the dataset (N)
  - In this case we can talk about best case
  - Or worst case
  - Or average case
- Worst case is usually considered
