Time Complexity of Algorithms: Takeaways

Syntax

- Evaluating the execution time of a Python function:

```
import time
start = time.time()
f(values)
end = time.time()
runtime = end - start
```

- Generating a random integer between *a* and *b*:

```
import random
random.randint(a, b)
```

- Generating a random list with *N* integers between *a* and *b*:

```
import random
def gen_random_list(N, a, b):
    return [random.randint(a, b) for _ in range(N)]
```

Concepts

- Processing speed and data both tend to double every two years. If algorithms remain the same, it can be the case that we are taking more and more time to process data.
- As the amount of data that we want to process increases, the execution time of a Python function becomes almost entirely defined by its slowest part. For this reason, when analyzing the execution time of an algorithm, we focus on this part only.
- Multiplicative constants do not impact the rate of growth of a function. Therefore, to simplify our analysis of the time complexity of algorithms, we can safely drop them.
- Evaluating the time complexity:
    - Count the number of times each line will be executed in the worst case.
    - Add and group terms together.
    - Drop all but the most significant term (the one with the highest exponent).
    - Drop the multiplying constant of the remaining term.

Resources

- Moore's law
- Growth of data
- `time` Python module
- `random` Python module
- Big O notation