Analysis of algorithms

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 2

Measuring performance

- Example of validating SIM cards against Aadhaar data
  - Naive approach takes thousands of years
  - Smarter solution takes a few minutes

Measuring performance

- Example of validating SIM cards against Aadhaar data
  - Naive approach takes thousands of years
  - Smarter solution takes a few minutes
- Two main resources of interest
  - Running time — how long the algorithm takes
  - Space — memory requirement

Measuring performance

- Example of validating SIM cards against Aadhaar data
  - Naive approach takes thousands of years
  - Smarter solution takes a few minutes
- Two main resources of interest
  - Running time — how long the algorithm takes
  - Space — memory requirement
- Time depends on processing power
  - Impossible to change for given hardware
  - Enhancing hardware has only a limited impact at a practical level

Measuring performance

- Example of validating SIM cards against Aadhaar data
  - Naive approach takes thousands of years
  - Smarter solution takes a few minutes
- Two main resources of interest
  - Running time — how long the algorithm takes
  - Space — memory requirement
- Time depends on processing power
  - Impossible to change for given hardware
  - Enhancing hardware has only a limited impact at a practical level
- Storage is limited by available memory
  - Easier to configure, augment

Measuring performance

- Example of validating SIM cards against Aadhaar data
  - Naive approach takes thousands of years
  - Smarter solution takes a few minutes
- Two main resources of interest
  - Running time — how long the algorithm takes
  - Space — memory requirement
- Time depends on processing power
  - Impossible to change for given hardware
  - Enhancing hardware has only a limited impact at a practical level
- Storage is limited by available memory
  - Easier to configure, augment
- Typically, we focus on time rather than space

Input size

- Running time depends on input size
    - Larger arrays will take longer to sort

Input size

- Running time depends on input size
  - Larger arrays will take longer to sort
- Measure time efficiency as function of input size
  - Input size n
  - Running time $t(n)$

Input size

- Running time depends on input size
  - Larger arrays will take longer to sort
- Measure time efficiency as function of input size
  - Input size n
  - Running time $t(n)$
- Different inputs of size n may take different amounts of time
  - We will return to this point later

Input size

- Running time depends on input size
  - Larger arrays will take longer to sort
- Measure time efficiency as function of input size
  - Input size n
  - Running time $t(n)$
- Different inputs of size n may take different amounts of time
  - We will return to this point later

Example 1 SIM cards vs Aadhaar cards

- $n \approx 10^9$ — number of cards

Input size

- Running time depends on input size
  - Larger arrays will take longer to sort
- Measure time efficiency as function of input size
  - Input size n
  - Running time $t(n)$
- Different inputs of size n may take different amounts of time
  - We will return to this point later

Example 1 SIM cards vs Aadhaar cards

- $n \approx 10^9$ — number of cards
- Naive algorithm: $t(n) \approx n^2$

Input size

- Running time depends on input size
  - Larger arrays will take longer to sort
- Measure time efficiency as function of input size
  - Input size n
  - Running time $t(n)$
- Different inputs of size n may take different amounts of time
  - We will return to this point later

Example 1 SIM cards vs Aadhaar cards

- $n \approx 10^9$ — number of cards
- Naive algorithm: $t(n) \approx n^2$
- Clever algorithm: $t(n) \approx n \log_2 n$
  - $\log_2 n$ — number of times you need to divide n by 2 to reach 1
  - $\log_2(n) = k \Rightarrow n = 2^k$

Input size . . .

Example 2 Video game

- Several objects on screen

Input size . . .

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects

Input size ...

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs

Input size . . .

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes sime $n \log_2 n$

Input size ...

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes sime $n \log_2 n$

- High resolution gaming consle may have $4000x2000$ pixels
  - 8×10^6 points — 8 million

Input size . . .

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes sime $n \log_2 n$

- High resolution gaming consle may have $4000x2000$ pixels
  - 8×10^6 points — 8 million
- Suppose we have $100,000 = 1 \times 10^5$ objects

Input size . . .

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes sime $n \log_2 n$

- High resolution gaming consle may have $4000x2000$ pixels
  - 8×10^6 points — 8 million
- Suppose we have $100,000 = 1 \times 10^5$ objects
- Naive algorithm takes 10^{10} steps
  - 1000 seconds, or 16.7 minutes in Python
  - Unacceptable response time!

Input size . . .

Example 2 Video game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes sime $n \log_2 n$

- High resolution gaming consle may have $4000x2000$ pixels
  - 8×10^6 points — 8 million
- Suppose we have $100,000 = 1 \times 10^5$ objects
- Naive algorithm takes 10^{10} steps
  - 1000 seconds, or 16.7 minutes in Python
  - Unacceptable response time!
- $\log_2 100,000$ is under 20, so $n \log_2 n$ takes a fraction of a second

Orders of magnitude

- When comparing $t(n)$, focus on orders of magnitude
  - Ignore constant factors

Orders of magnitude

- When comparing $t(n)$, focus on orders of magnitude
  - Ignore constant factors
- $f(n) = n^3$ eventually grows faster than $g(n) = 5000n^2$
  - For small values of n, $f(n) < g(n)$
  - After $n = 5000$, $f(n)$ overtakes $g(n)$

Orders of magnitude

- When comparing $t(n)$, focus on orders of magnitude
  - Ignore constant factors
- $f(n) = n^3$ eventually grows faster than $g(n) = 5000n^2$
  - For small values of n, $f(n) < g(n)$
  - After $n = 5000$, $f(n)$ overtakes $g(n)$
- Asymptotic complexity
  - What happens in the limit, as n becomes large

Orders of magnitude

- When comparing $t(n)$, focus on orders of magnitude
  - Ignore constant factors
- $f(n) = n^3$ eventually grows faster than $g(n) = 5000n^2$
  - For small values of n, $f(n) < g(n)$
  - After $n = 5000$, $f(n)$ overtakes $g(n)$
- Asymptotic complexity
  - What happens in the limit, as n becomes large
- Typical growth functions
  - Is $t(n)$ proportional to $\log n, \ldots, n^2, n^3, \ldots, 2^n$?
    - Note: $\log n$ means $\log_2 n$ by default
  - Logarithmic, polynomial, exponential, ...

Orders of magnitude

Input size	Values of $t(n)$					
	$\log n$	n	$n \log n$	n^2	n^3	2^n	$n!$
10	3.3	10	33	100	1000	1000	10^6
100	6.6	100	66	10^4	10^6	10^{30}	10^{157}
1000	10	1000	10^4	10^6	10^9		
10^4	13	10^4	10^5	10^8	10^{12}		
10^5	17	10^5	10^6	10^{10}			
10^6	20	10^6	10^7	10^{12}			
10^7	23	10^7	10^8				
10^8	27	10^8	10^9				
10^9	30	10^9	10^{10}				
10^{10}	33	10^{10}	10^{11}				

Measuring running time

- Analysis should be independent of the underlying hardware
  - Don't use actual time
  - Measure in terms of basic operations

Measuring running time

- Analysis should be independent of the underlying hardware
  - Don't use actual time
  - Measure in terms of basic operations
- Typical basic operations
  - Compare two values
  - Assign a value to a variable

Measuring running time

- Analysis should be independent of the underlying hardware
  - Don't use actual time
  - Measure in terms of basic operations
- Typical basic operations
  - Compare two values
  - Assign a value to a variable
- Exchange a pair of values?

```
(x,y) = (y,x)
```

```
t = x
x = y
y = t
```

  - If we ignore constants, focus on orders of magnitude, both are within a factor of 3
  - Need not be very precise about defining basic operations

What is the input size

- Typically a natural parameter
  - Size of a list/array that we want to search or sort
  - Number of objects we want to rearrange
  - Number of vertices and number edges in a graph
    - We shall see why these are separate parameters

What is the input size

- Typically a natural parameter
  - Size of a list/array that we want to search or sort
  - Number of objects we want to rearrange
  - Number of vertices and number edges in a graph
    - We shall see why these are separate parameters
- What about numeric problems? Is n a prime?

What is the input size

- Typically a natural parameter
  - Size of a list/array that we want to search or sort
  - Number of objects we want to rearrange
  - Number of vertices and number edges in a graph
    - We shall see why these are separate parameters
- What about numeric problems? Is n a prime?
  - Magnitude of n is not the correct measure

What is the input size

- Typically a natural parameter
  - Size of a list/array that we want to search or sort
  - Number of objects we want to rearrange
  - Number of vertices and number edges in a graph
    - We shall see why these are separate parameters
- What about numeric problems? Is n a prime?
  - Magnitude of n is not the correct measure
  - Arithmetic operations are performed digit by digit
    - Addition with carry, subtraction with borrow, multiplication, long division ...

What is the input size

- Typically a natural parameter
  - Size of a list/array that we want to search or sort
  - Number of objects we want to rearrange
  - Number of vertices and number edges in a graph
    - We shall see why these are separate parameters
- What about numeric problems? Is n a prime?
  - Magnitude of n is not the correct measure
  - Arithmetic operations are performed digit by digit
    - Addition with carry, subtraction with borrow, multiplication, long division . . .
  - Number of digits is a natural measure of input size
    - Same as $\log_b n$, when we write n in base b

Which inputs should we consider?

- Performance varies across input instances
  - By luck, the value we are searching for is the first element we examine in an array

Which inputs should we consider?

- Performance varies across input instances
  - By luck, the value we are searching for is the first element we examine in an array
- Ideally, want the "average" behaviour
  - Difficult to compute
  - Average over what? Are all inputs equally likely?
  - Need a probability distribution over inputs

Which inputs should we consider?

- Performance varies across input instances
  - By luck, the value we are searching for is the first element we examine in an array
- Ideally, want the “average” behaviour
  - Difficult to compute
  - Average over what? Are all inputs equally likely?
  - Need a probability distribution over inputs
- Instead, worst case input
  - Input that forces algorithm to take longest possible time
    - Search for a value that is not present in an unsorted list
    - Must scan all elements

Which inputs should we consider?

- Performance varies across input instances
  - By luck, the value we are searching for is the first element we examine in an array
- Ideally, want the "average" behaviour
  - Difficult to compute
  - Average over what? Are all inputs equally likely?
  - Need a probability distribution over inputs
- Instead, worst case input
  - Input that forces algorithm to take longest possible time
    - Search for a value that is not present in an unsorted list
    - Must scan all elements
  - Pessimistic — worst case may be rare

Which inputs should we consider?

- Performance varies across input instances
  - By luck, the value we are searching for is the first element we examine in an array
- Ideally, want the “average” behaviour
  - Difficult to compute
  - Average over what? Are all inputs equally likely?
  - Need a probability distribution over inputs
- Instead, worst case input
  - Input that forces algorithm to take longest possible time
    - Search for a value that is not present in an unsorted list
    - Must scan all elements
  - Pessimistic — worst case may be rare
  - Upper bound for worst case guarantees good performance

Summary

- Two important parameters when measuring algorithm performance
  - Running time, memory requirement (space)
  - We mainly focus on time

Summary

- Two important parameters when measuring algorithm performance
  - Running time, memory requirement (space)
  - We mainly focus on time
- Running time $t(n)$ is a function of input size n
  - Interested in orders of magnitude
  - Asymptotic complexity, as n becomes large

Summary

- Two important parameters when measuring algorithm performance
  - Running time, memory requirement (space)
  - We mainly focus on time
- Running time $t(n)$ is a function of input size n
  - Interested in orders of magnitude
  - Asymptotic complexity, as n becomes large
- From running time, we can estimate feasible input sizes

Summary

- Two important parameters when measuring algorithm performance
  - Running time, memory requirement (space)
  - We mainly focus on time
- Running time $t(n)$ is a function of input size n
  - Interested in orders of magnitude
  - Asymptotic complexity, as n becomes large
- From running time, we can estimate feasible input sizes
- We focus on worst case inputs
  - Pessimistic, but easier to calculate than average case
  - Upper bound on worst case gives us an overall guarantee on performance