Time Complexity

Code Club, 19th Nov 2025

James Cornall

Efficiency

- What do we want out of our software?
  - Run quickly
  - Respond quickly
  - Use less…
    - Memory
    - Bandwidth
    - Power
- We need to optimise many things

So how do we optimise?

- Improve our code to run faster
  - Could be a small improvement
  - Could be an improvement of many magnitudes
- Improve how our data is represented

- But there are limits…
  - As the amount of data increases, will performance drop?
  - Is the **problem** we are trying to solve inherently hard?

Complexity

- A way to understand how complexity changes as the problem grows larger
- Space Complexity
  - e.g. HV Video
    - 1920x1080 pixels (4 bytes per pixel)
      - ~2MPixels, ~8Mbytes
    - 25 fps
      - 200MBytes/sec
  - Solutions:
    - Compression, reduce frame rate/resolution

Time Complexity

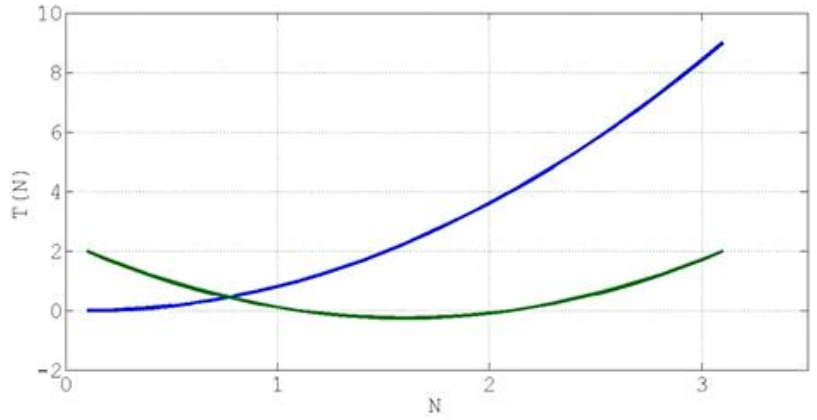
- The number of operations that an algorithm needs to execute, in terms of the size of the input or problem
- Defined as a function *T(n)*, where *n* is the number of elements
  - Focus on the algorithm, not the implementation
    - i.e. no fixed programming language, computer architecture

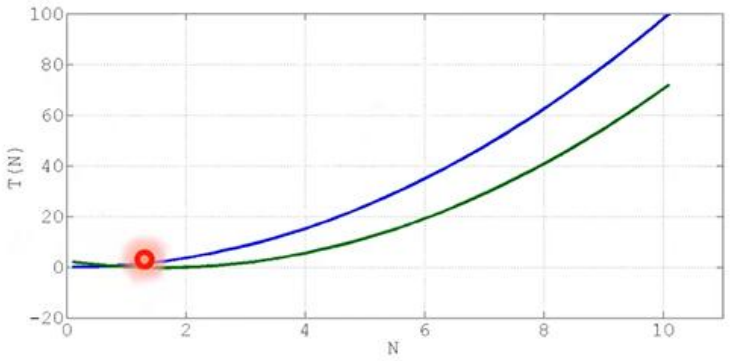
Time Complexity

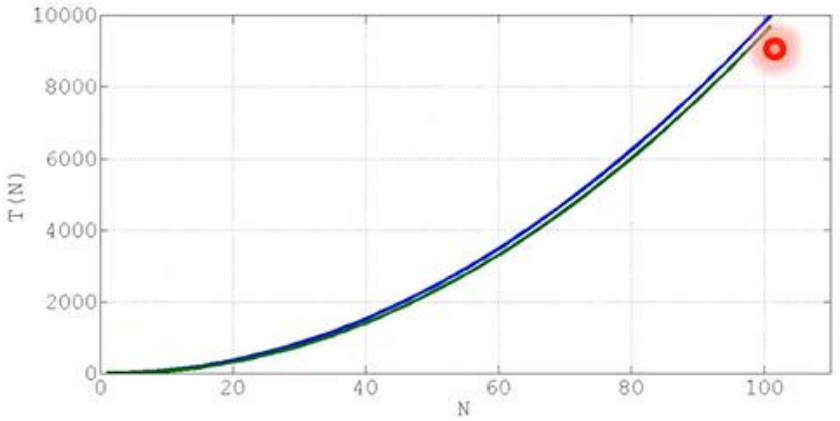
- The number of operations that an algorithm needs to execute, in terms of the size of the input or problem
- Defined as a function *T(n)*, where *n* is the number of elements
  - Focus on the algorithm, not the implementation
    - i.e. no fixed programming language, computer architecture
- Focus on the **worst-case**

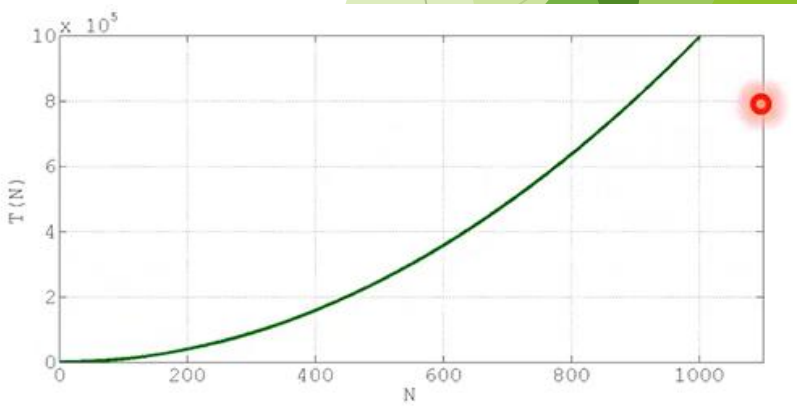
Example

- Consider two functions:
  - *f(n) = n^2*
  - *g(n) = n^2 - 3n + 2*





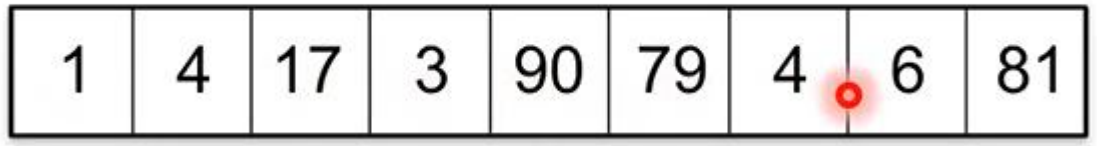




More examples

- Linear Search
- *T(n) = n*

- Look up a value v in an array x of integers



- T(n) = n

```
for i=0...n-1:
    if x[i] == v:
        return i
return -1
```

More examples

- Matrix-Vector Multiplication
- $x = A\ b$
  - n x n matrix A, vector of b size n
- Inputs: Matrix A, vector b
- Result stored in vector x (initially all 0)
- $T(n) = 2n^2$

```
for i=0...n-1:
    for j=0...n-1:
        x[i] += x[i] + A[i][j] * b[j]
```

Big-O Notation

- A way to express the behaviour of time complexity – the complexity class
- Not exact
- Ignores constants/lower orders
- Asymptotic analysis

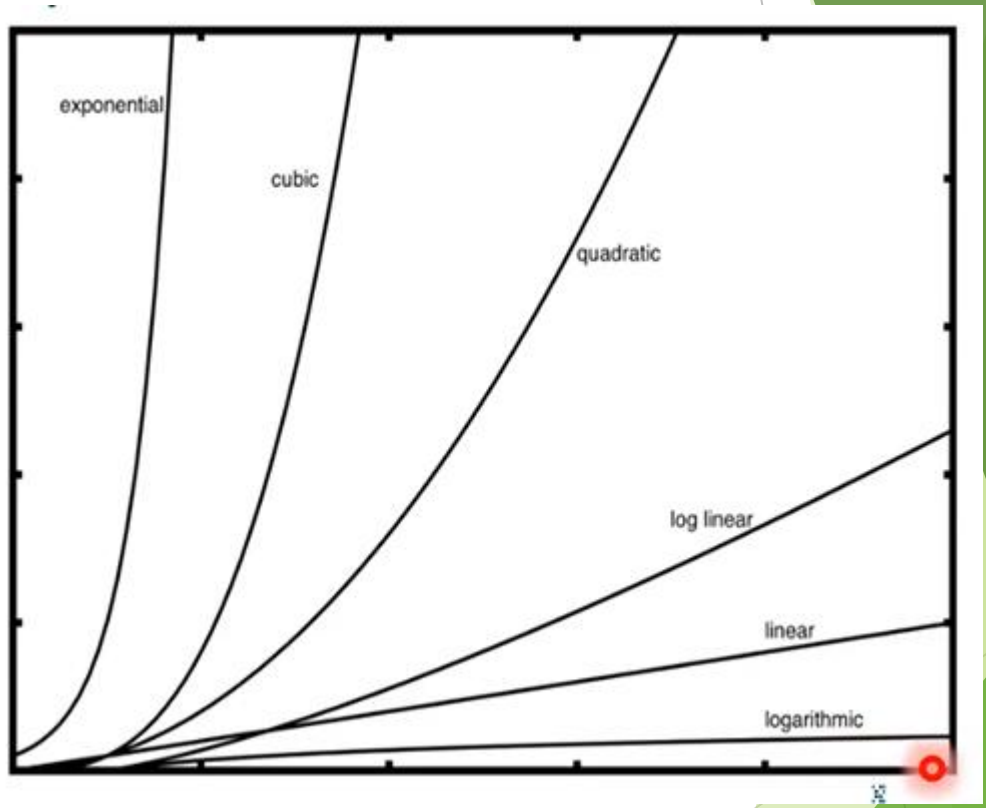
- e.g. O(n^2)

Even more examples

- T(n) = n … O(n)
- T(n) = n + 2 … O(n)
- T(n) = 2n^2 … O(n^2)
- T(n) = 10n^3 + 1 … O(n^3)
- T(n) = 5(n + 2) … O(n)
- T(n) = 1000 … O(1)
- T(n) = n^2 + n + 1 … O(n^2)

Complexity Classes

- O(1): Constant
- O(log2 n): Logarithmic
- O(n): Linear
- O(n log2 n): Log Linear
- O(n^2): Quadratic
- O(n^3): Cubic
- O(2^n): Exponential



Complexity Classes

- Polynomials:
  - O(n), O(n^2), O(n^3)…
  - Manageable
- Exponential:
  - O(2^n), O(C^n)…
  - Manageable with small amounts of data
  - Impossible with large amounts of data

Complexity in Numbers…

f(n)	n = 4	n = 16	n = 256	n = 1024	n = 1048576
1	1	1	1	1.00×10^{0}	1.00×10^{0}
$\log_2 \log_2 n$	1	2	3	3.32×10^{0}	4.32×10^{0}
$\log_2 n$	2	4	8	1.00×10^{1}	2.00×10^{1}
n	4	16	256	1.02×10^{3}	1.05×10^{6}
$n \log_2 n$	8	64	2.05×10^{3}	1.02×10^{4}	2.10×10^{7}
n^2	16	256	6.55×10^{4}	1.05×10^{6}	1.10×10^{12}
n^3	64	4.10×10^{3}	1.68×10^{7}	1.07×10^{9}	1.15×10^{18}
2^n	16	65536	1.16×10^{77}	1.80×10^{308}	6.74×10^{315652}

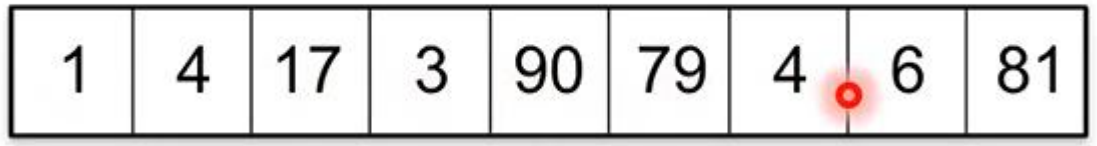
- Assuming 1 million operations per second

f(n)	n = 4	n = 16	n = 256	n = 1024	n = 1048576
1	1 usec	1 usec	1 usec	1 usec	1 usec
$\log_2 \log_2 n$	1 usec	2 usec	3 usec	3.32 usec	4.32 usec
$\log_2 n$	2 usec	4 usec	8 usec	10 usec	20 usec
n	4 usec	16 usec	256 usec	1.02 msec	1.05 sec
$n \log_2 n$	8 usec	64 usec	2.05 msec	10.2 msec	21 sec
n^2	16 usec	256 usec	65.5 msec	1.05 sec	12.72 days
n^3	64 usec	4.1 msec	16.8 sec	17.83 min	36559 yrs
2^n	16 usec	65.5 msec	3.7×10^{63} yrs	5.7×10^{294} yrs	2.14×10^{315639} yrs

Revisiting Linear Search

- Linear Search
- Complexity = O(n)

- Look up a value v in an array x of integers

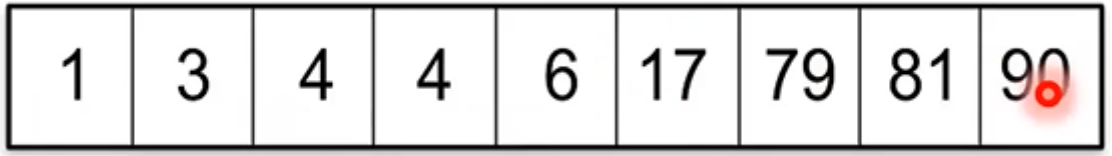


- T(n) = n

```
for i=0...n-1:
    if x[i] == v:
        return i
return -1
```

Binary Search

- Binary Search
- Complexity = O(log2 n)




```
for i=0...n-1:
    binary_search(x,v_i)
```

Computing Complexity Classes

- Sequential Algorithm Phases
- Complexity = O(n) + O(n^2)
  - Take the maximum complexity
  - Complexity = O(n^2)

```
for i=0...n-1:
   x[i] = 0
for i=0...n-1:
   for j=0...n-1:
      x[i] = x[i] + A[i][j] * b[j]
```

- Function/Method Calls
- Complexity = O(n) x O(log n)
  - Multiply the complexities
  - Complexity = O(n log n)

```
for i=0...n-1:
   binary_search(x,v_i)
```

Harder Problems

- Travelling Salesman Problem (TSP)
  - Given n cities and the distances between them, what is the shortest possible route that visits each city exactly once and then returns to the first city?
- Boolean Satisfiability Problem (SAT)
  - Given a Boolean formula F(x1, x2, x3, ..., xn), can F evaluate to 1 (true)?
  - If yes, return values of xi's (satisfying the assignment) that make F true

Thank you!

Any questions?