
COMP110: Principles of Computing

# 5: Computational Complexity

# Learning outcomes

- **Explain** the notion of computability
- **Use** "big $O$" notation to express computational complexity
- **Apply** appropriate algorithms to achieve efficiency

FALMOUTH UNIVERSITY

# More on complexity

# Common complexity classes

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |
| Fractional power | $O(n^k), k < 1$ |

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |
| Fractional power | $O(n^k),\ k < 1$ |
| Linear | $O(n)$ |

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |
| Fractional power | $O(n^k)$, $k < 1$ |
| Linear | $O(n)$ |
| Quadratic | $O(n^2)$ |

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |
| Fractional power | $O(n^k), k < 1$ |
| Linear | $O(n)$ |
| Quadratic | $O(n^2)$ |
| Polynomial | $O(n^k), k > 1$ |

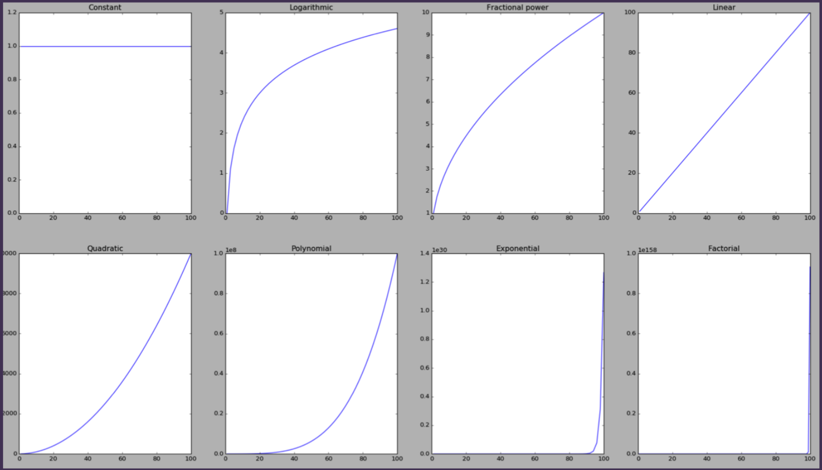
# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |
| Fractional power | $O(n^k), k < 1$ |
| Linear | $O(n)$ |
| Quadratic | $O(n^2)$ |
| Polynomial | $O(n^k), k > 1$ |
| Exponential | $O(e^n)$ |

# Common complexity classes

| | |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(\log n)$ |
| Fractional power | $O(n^k)$, $k < 1$ |
| Linear | $O(n)$ |
| Quadratic | $O(n^2)$ |
| Polynomial | $O(n^k)$, $k > 1$ |
| Exponential | $O(e^n)$ |
| Factorial | $O(n!)$ |

# Common complexity classes



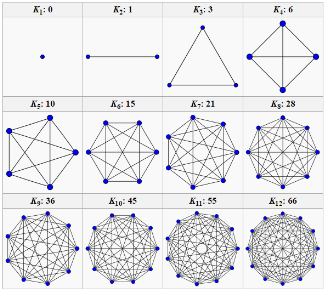
# Complexity in games

- Example: collision detection between $n$ objects

# Complexity in games

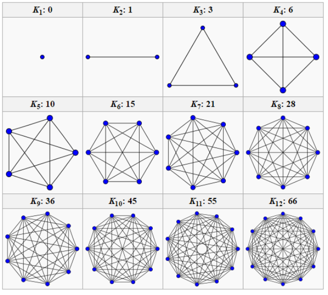
- Example: collision detection between *n* objects
- The naïve way: check **each pair** of objects to see whether they have collided

# Complexity in games



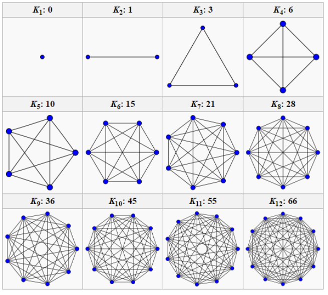

- Example: collision detection between $n$ objects
- The naïve way: check **each pair** of objects to see whether they have collided
- This is **quadratic** or $O(n^2)$

# Complexity in games



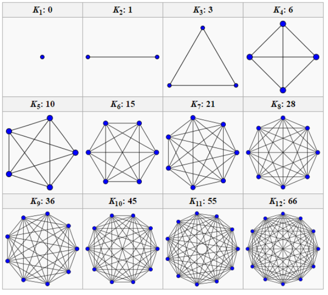

- Example: collision detection between *n* objects
- The naïve way: check **each pair** of objects to see whether they have collided
- This is **quadratic** or $O(n^2)$
- Doubling the number of objects would **quadruple** the time required!

# Complexity in games




- Example: collision detection between $n$ objects
- The naïve way: check **each pair** of objects to see whether they have collided
- This is **quadratic** or $O(n^2)$
- Doubling the number of objects would **quadruple** the time required!
- Cleverer methods exist that are more scalable

# Complexity in games




- Example: collision detection between *n* objects
- The naïve way: check **each pair** of objects to see whether they have collided
- This is **quadratic** or $O(n^2)$
- Doubling the number of objects would **quadruple** the time required!
- Cleverer methods exist that are more scalable
  - Further reading: spatial hashing, quadtrees, octrees, Verlet lists

# Aside: a famous unanswered question in computing

# Aside: a famous unanswered question in computing

- A problem is “in $P$” if it can be solved with an algorithm running in $O(n^k)$ time

# Aside: a famous unanswered question in computing

- A problem is "in $P$" if it can be solved with an algorithm running in $O(n^k)$ time
- A problem is in $NP$ if a potential solution can be checked in $O(n^k)$ time

# Aside: a famous unanswered question in computing

- A problem is "in *P*" if it can be solved with an algorithm running in $O(n^k)$ time
- A problem is in *NP* if a potential solution can be checked in $O(n^k)$ time
  - Equivalently, it can be solved with an algorithm running in $O(n^k)$ time on an infinitely parallel machine

# Aside: a famous unanswered question in computing

- A problem is "in *P*" if it can be solved with an algorithm running in $O(n^k)$ time
- A problem is in *NP* if a potential solution can be checked in $O(n^k)$ time
  - Equivalently, it can be solved with an algorithm running in $O(n^k)$ time on an infinitely parallel machine
- Are there any problems in *NP* but not in *P*?

# P versus NP

# P versus NP

- If you can find a mathematical proof that $P = NP$ or $P \neq NP$, there's a \$1 million prize...

# P versus NP

- If you can find a mathematical proof that $P = NP$ or $P \neq NP$, there's a \$1 million prize...
- It is believed that $P \neq NP$, so large instances of $NP$-hard problems are not solvable in a feasible amount of time

# P versus NP

- If you can find a mathematical proof that $P = NP$ or $P \neq NP$, there's a \$1 million prize...
- It is believed that $P \neq NP$, so large instances of $NP$-hard problems are not solvable in a feasible amount of time
    - Many types of cryptography are based on this assumption

# P versus NP

- If you can find a mathematical proof that $P = NP$ or $P \neq NP$, there's a \$1 million prize...
- It is believed that $P \neq NP$, so large instances of $NP$-hard problems are not solvable in a feasible amount of time
  - Many types of cryptography are based on this assumption
  - Quantum computers are "infinitely parallel" in a sense so *can* solve some large $NP$-hard problems

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important
  - Linear search is quicker than binary search if you only ever have 3 elements

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important
  - Linear search is quicker than binary search if you only ever have 3 elements
  - Naïve collision detection is fine if your game only ever has 4 objects on screen

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important
  - Linear search is quicker than binary search if you only ever have 3 elements
  - Naïve collision detection is fine if your game only ever has 4 objects on screen
  - Sometimes complexity in terms of other resources (e.g. space, bandwidth) are more important than time

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important
  - Linear search is quicker than binary search if you only ever have 3 elements
  - Naïve collision detection is fine if your game only ever has 4 objects on screen
  - Sometimes complexity in terms of other resources (e.g. space, bandwidth) are more important than time
- Software development is all about choosing **the right tool for the job**

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important
  - Linear search is quicker than binary search if you only ever have 3 elements
  - Naïve collision detection is fine if your game only ever has 4 objects on screen
  - Sometimes complexity in terms of other resources (e.g. space, bandwidth) are more important than time
- Software development is all about choosing **the right tool for the job**
  - If you need scalability, choose a scalable algorithm

# Caveats

- Time complexity only tells us how an algorithm **scales** with the size of the input
  - If we know the input will always be **small**, time complexity is not so important
  - Linear search is quicker than binary search if you only ever have 3 elements
  - Naïve collision detection is fine if your game only ever has 4 objects on screen
  - Sometimes complexity in terms of other resources (e.g. space, bandwidth) are more important than time
- Software development is all about choosing **the right tool for the job**
  - If you need scalability, choose a scalable algorithm
  - Otherwise, choose simplicity

# Summary

- Time complexity tells us how the running time of an algorithm **scales** with the size of the data it is given

# Summary

- Time complexity tells us how the running time of an algorithm **scales** with the size of the data it is given
- Choice of data structures and algorithms can have a large impact on the efficiency of your software

# Summary

- Time complexity tells us how the running time of an algorithm **scales** with the size of the data it is given
- Choice of data structures and algorithms can have a large impact on the efficiency of your software
- ... but only if scalability is actually a factor