A brief intro to growth rates

CS605
2024-2025
Maynooth University
T. Naughton

Asymptotic notation

- When analysing how much longer a computer program takes on inputs of various sizes, we are only interested in growth ***rates***. Additive and multiplicative constants are disregarded in *O* notation, so

  $n = O(n)$
  $2n + 3 = O(n)$

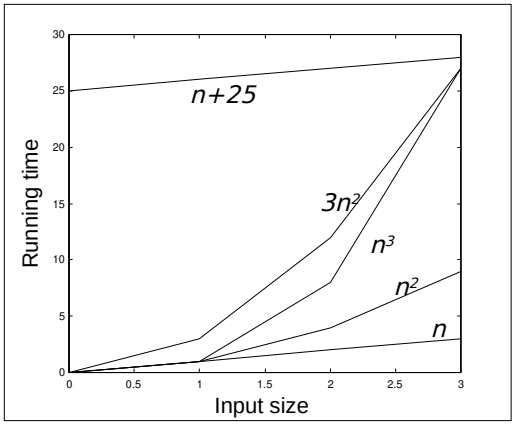
- and in general

  $c_1n^3 + c_2n^2 + c_3n + c_4 = O(n^3)$
  (where c_i is any constant)

Asymptotic notation

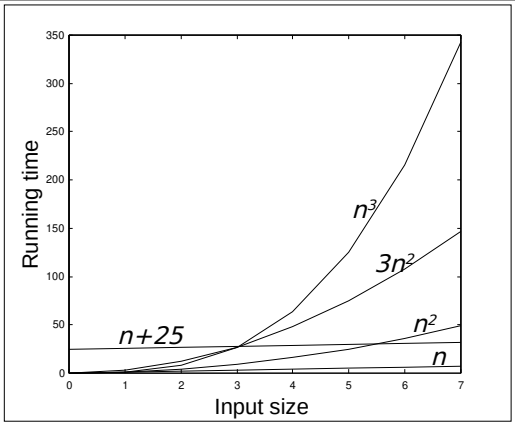
- For small values of *n* these constants might be significant...

For small n (n=0..3)



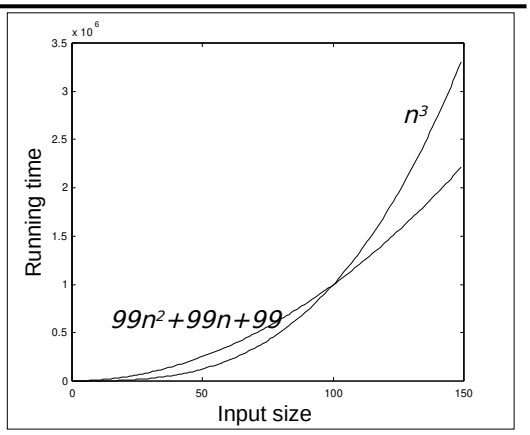
- However, as *n* becomes larger...

For larger n (n=0..7)



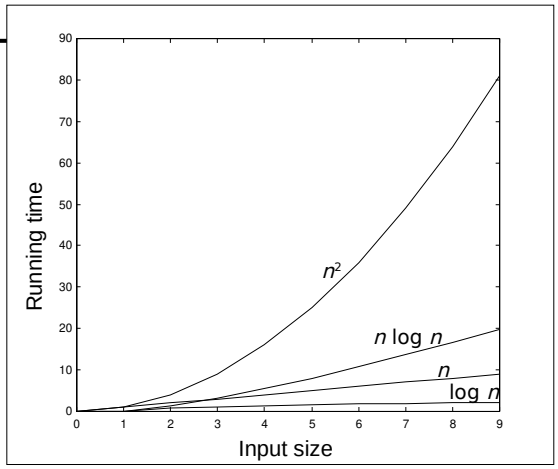
- As n becomes larger the power term becomes more significant than the additive constant and multiplicative constant terms.
- Looking at this in an extreme case...

For sufficiently large n (n=0..150)



Reasonable growth rates

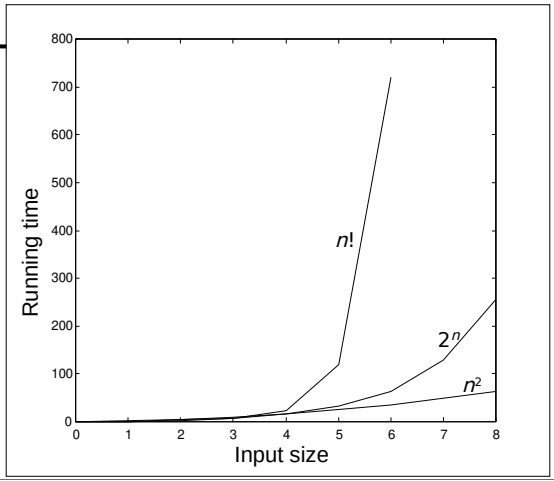
- 1 Constant (no growth)
- n^c , $0 < c < 1$ Sublinear
- n Linear
- n^c , $1 < c < 2$ Subquadratic
- n^2 Quadratic
- n^3 Cubic
- n^c , $c \geq 1$ Polynomial
- $\log n$ Logarithmic
- $\log^c n$, $c \geq 1$ Polylogarithmic



Other growth rates

- c^n , $c > 1$ Exponential
- $n!$ Factorial

- We also have Superpolynomial and Subexponential growth rates.



Standard growth rates

When n doubles

- log increments
- linear doubles
- quadratic quadruples
- exp squares

Ordering of growth rates

In increasing order

- 1
- log log n
- log n
- $\log^2 n$
- $n^{1/10}$
- $n^{1/2}$
- n
- $n \log n$
- n^2
- 2^n
- $n!$
- 2^{2^n}
- $2^{n!}$

Intractability

A problem is *intractable* if

- no polynomial solution has yet been found for it, AND
- its complexity has been proved to be equivalent to that of a problem that was previously found to be intractable
  - the complexity being measured with respect to some standard complexity measures and model of computation.

Intractability

- We say that the work involved in computing them is polynomially *unbounded* as the size of the input increases.
  - Algorithms with a resource cost function 'steeper' than any polynomial, such as an exponential function for large inputs, are referred to as polynomially unbounded.

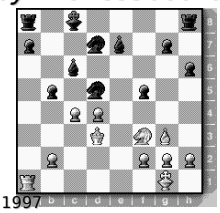
For example...

Self-Avoiding Walks of a Rook

The Rook Path Problem

- How many self-avoiding walks $W(m,n)$ can a rook take from one corner to the opposite corner of an m-by-n chessboard?

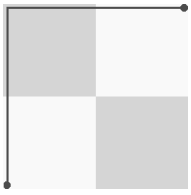
"The chessboard is the world..."
-- Huxley/Hardy

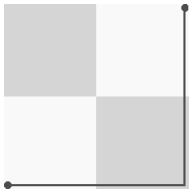


Deep Blue, move 19 c2-c4 , game 6, 11 May 1997

Solutions

For small values of m and n the answer is trivial. Clearly $W(2,2)=2$

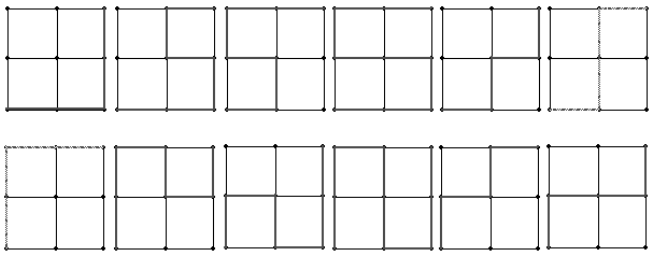




Similarly $W(3,2)=4$.

Solutions

With a little patience we can also verify $W(3,3)=12$.



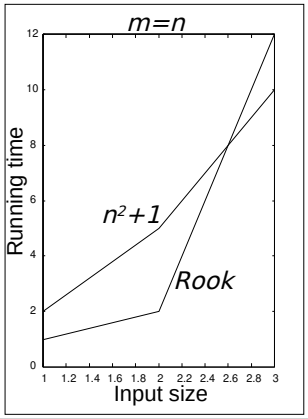
Complexity measure & model

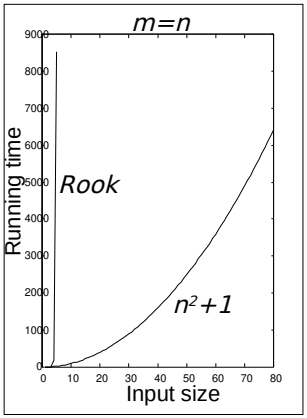
- For simplicity, we will only count the number of paths found (measure), ignoring the resource requirements of actually traversing each path (model).

Complexity measure & model

- Even though we only count the paths, the resource requirements of the algorithm still grow rapidly as the input increases.
- We can compare this growth with a quadratic growth rate of a slow sorting algorithm...

Resource cost-functions





Solutions (so far!)

$m=n$	$W(n,n)$
1	1
2	2
3	12
4	184
5	8512
6	1262816
7	575780564
8	789360053252
9	3266598486981642
10	41044208702632496804
11	156875803046475001321 4100
12	18241329151424804924147 0885236
13	64528039343270018963357185158482118
14	69450664761521361664274701548907358996488
15	22744971467681273963182645932798986338761 3323440

Bounds on the resource cost-function

- It has been found that $W(n,2)=2^{n-1}$ for all $n \geq 1$.

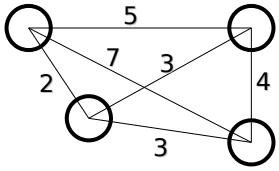
  Several expressions for $W(n,3)$ were found and it has been estimated that $W(n,n)<2^{n^2}$, however precise estimates of the bounds remain as difficult unsolved problems.

Aside...

- In general, any computation that takes 2^n steps on an input of length n would not be regarded as *practical* or *feasible*. No computer would ever finish such a computation in the lifetime of the universe with even an input n of 1000. Computational complexity theory tries to identify problems that are *feasibly computable*.
- If we had one processor capable of 10^6 steps/sec then in a year we would execute 3.2×10^{13} steps ... just enough for one 2^n problem with 44 inputs.
- If we had a million of such processors, each capable of 10^6 steps/sec, then in a year we would execute 3.2×10^{19} steps ... just enough for one 2^n problem with 64 inputs.

Let's look at the most widely known intractable problem...

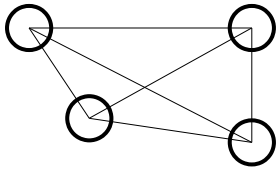
Travelling salesperson problem



$k = 20$

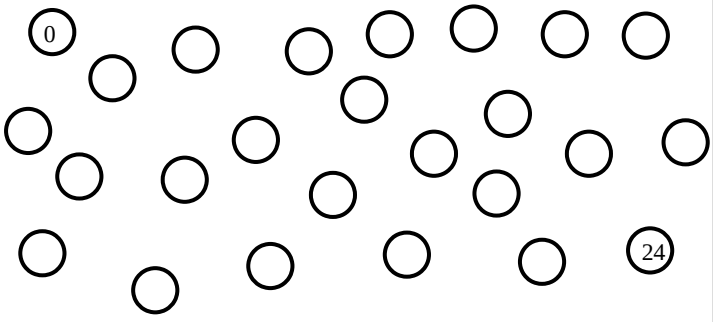
- Is there a tour through the graph (that visits each of the N nodes) of length $\leq k$?

Travelling salesperson problem



- The problem is easy to formulate, but no computer scientist or mathematician has come up with a tractable algorithm for our standard models of computation.
- What about other models of computation?

Travelling salesperson problem



- If you were offered a million euro...?

Travelling salesperson problem

- With 5 cities there are 12 possibilities, with 10 cities there are 181,440 possibilities.
- For 25 cities, a supercomputer evaluating a million possibilities per second would take 9.8×10^9 years to search them all.
  - That's about two-thirds the (scientifically agreed) age of the universe!

Similar optimisation problems

- Air traffic control
- Maximising the productivity of a widely skilled workforce
- Fault detection in logic circuits
- Timetabling
- Multiprocessor scheduling
- Dynamic memory allocation
- Optimal data compression

For all of these problems, computer scientists have developed very efficient heuristic algorithms that find approximate solutions that are often very close to the true solution. However, they cannot guarantee to find a solution within any particular percentage of the true solution (meaning, for some inputs they can give very poor results). This phenomenon is common in the field of computer algorithms; for example quicksort has excellent average case complexity but a worst case complexity as bad as bubble sort.