Algorithms and Complexity

Module 1

Data Structures and Algorithms

- An algorithm is a step-by-step procedure for performing some task in a finite amount of time
- An algorithm takes any of the possible input instances and transforms it to the desired output, e.g., an unsorted list of numbers to the sorted one
- A data structure is a systematic way of organizing and accessing data, e.g., array, stack, queue, heap
- Algorithms and data structures go hand in hand as some algorithms can operate only on specific data structures ("*Algorithms + Data Structures = Programs*" (Niklaus Wirth))

Data Structures and Algorithms

- Euclid's algorithm for finding the greatest common divisor (GCD) of two nonnegative numbers *p* and *q* (c. 300 BC):
  1. If *q* is 0, the answer is *p*
  2. If not, divide *p* by *q*, take the remainder *r*, set *p* = *q* and *q* = *r*, and go to 1
- Python code:

```python
def gdc(p, q):
    if q == 0:
        return p
    return gdc(q, p % q)
```

Algorithm Performance

- Algorithms can be studied in a language- and machine-independent way
- To compare the efficiency of algorithms without implementing them we use the RAM model of computation and the asymptotic analysis of worst-case complexity
- Random Access Machine (RAM) is a hypothetical computer for which:
  - Each simple operation (+, *, –, =, if, call) takes exactly one time step
  - Loops and subroutines are considered the composition of many single-step operations
  - Amount of memory is unlimited, and each memory access takes exactly one time step

Algorithm Performance

- The worst-case complexity of the algorithm is the function defined by the maximum number of steps taken in any instance of size *n*
- The best-case complexity is the function defined by the minimum number of steps taken in any instance of size *n*
- The average-case complexity is the function defined by the average number of steps over all instances of size *n*
- Example: searching for a number in an unsorted list of size *n*:
  - Worst case = *n* (the number is at the last position)
  - Best case = 1 (the number is at the first position)
  - Average case = *n*/2 (the number is in the middle)

Algorithm Performance

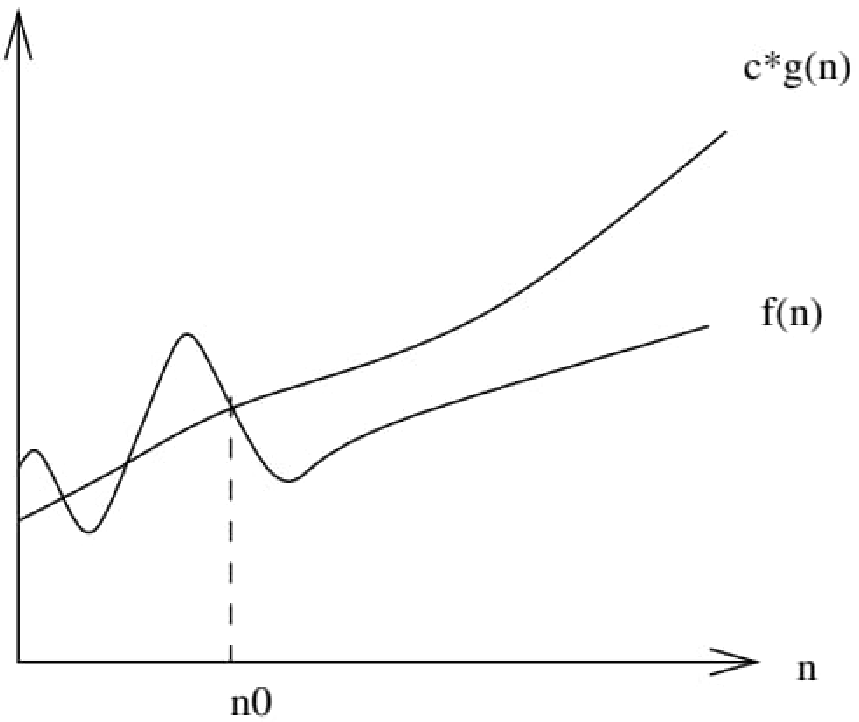
- The worst-case complexity is most useful in practice
- Average-case analysis is typically quite challenging as it requires knowledge of a probability distribution on the set of inputs
- Worst-case analysis is much easier as it requires only the ability to identify the worst-case input
- Also, if an algorithm performs well in the worst case, it will do well on every input

The Big Oh Notation

- Time complexities for any given algorithm are numerical functions over the size of possible problem instances
- However, the exact time complexity function for any algorithm can be very complicated, e.g., $T(n) = 12754n^2 + 4353n + 834log_2n + 13546$
- Counting the exact number of RAM instructions executed in the worst case requires the algorithm be specified to the detail of a complete computer program
- The Big Oh notation simplifies the analysis by ignoring levels of detail that do not impact the comparison of algorithms

The Big Oh Notation

- The Big Oh notation ignores the difference between multiplicative constants
- The functions $f(n) = 2n$ and $g(n) = n$ are identical in Big Oh analysis
- The formal definition associated with the Big Oh notation:
  – $f(n) = O(g(n))$ means $c \times g(n)$ is an upper bound on $f(n)$. Thus, there exists some constant c such that $f(n)$ is always $\leq c \cdot g(n)$, for large enough n (i.e., $n \geq n_0$ for some constant n_0)

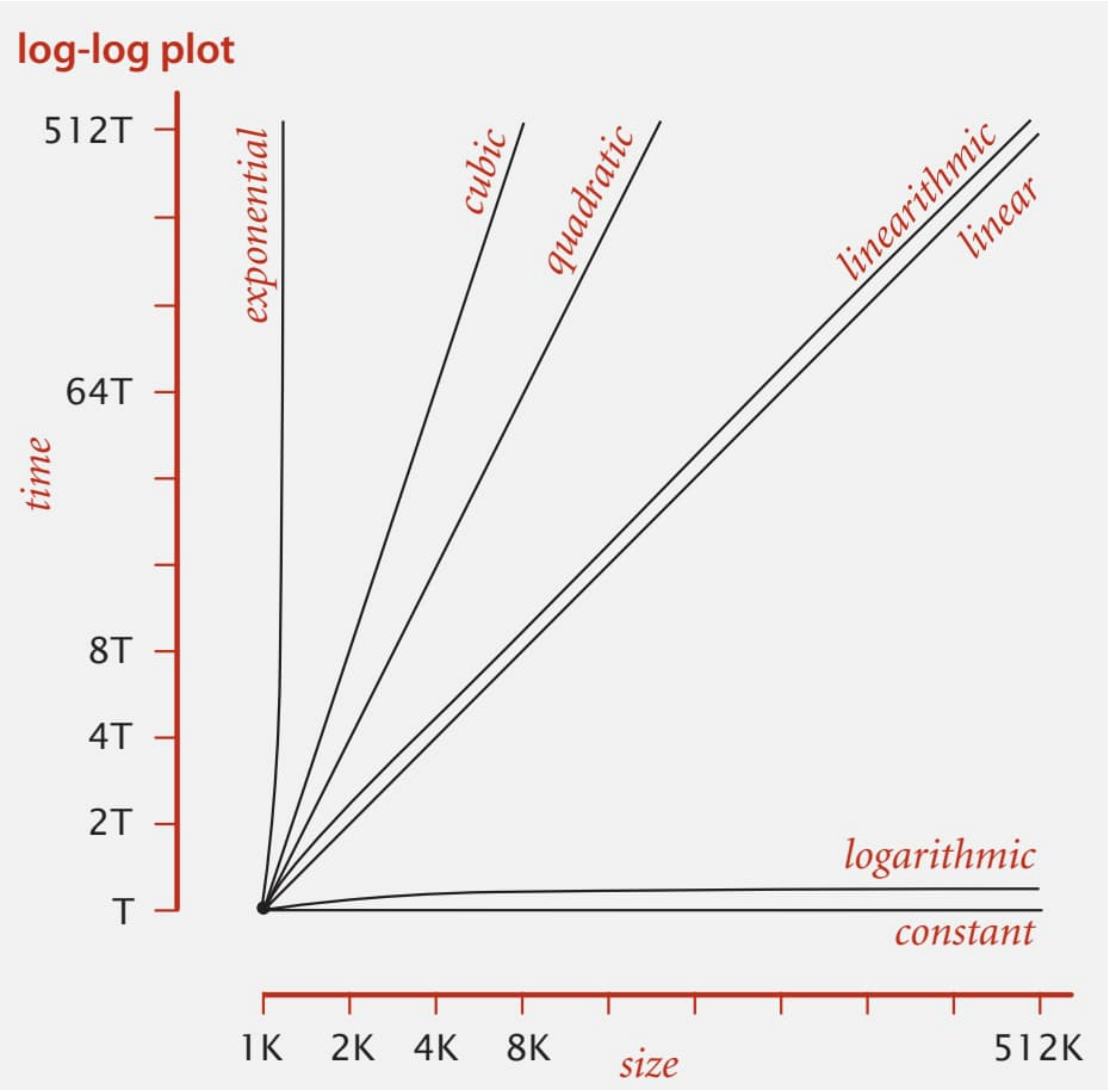


The Big Oh Notation

- The Big Oh notation allows us to say that a function $f(n)$ is "less than or equal to" another function $g(n)$ up to a constant factor and in the asymptotic sense as n grows toward infinity
- Examples:
  - $8n + 5 = O(n)$ since for $c = 9$ and $n_0 = 5$, $9n \geq 8n + 5$
  - $3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$
  - $3n^2 - 100n + 6 \neq O(n)$ since for any c, $c \times n < 3n^2$ when $n > c$
  - In general, if $f(n)$ is a polynomial of degree d, i.e., $f(n) = a_0 + a_1 n + \cdots + a_d n^d$, and $a_d > 0$, then $f(n)$ is $O(n^d)$
  - $3\log n + 2 = O(\log n)$ since c = 5 and $n_0 = 2$, $5\log n \geq 3\log n + 2$
  - $2^{n+2} = O(2^n)$, since for $c = 4$ and $n_0 = 1$, $4 \times 2^n \geq 2^{n+2}$

The Big Oh Notation

- Typical order-of-growth functions:
  - Constant - $O(1)$
  - Logarithmic - $O(\log n)$
  - Linear - $O(n)$
  - Linearithmic - $O(n \log n)$
  - Quadratic $O(n^2)$
  - Cubic $O(n^3)$
  - Exponential $O(2^n)$
  - Factorial $O(n!)$



The Big Oh Notation

- Growth rates of common functions measured in nanoseconds

N	*log N*	*N*	*N log N*	N^2	2^N	*N!*
10	0,003 µs	0,01 µs	0,033 µs	0,1 µs	1 µs	3,63 ms
20	0,004 µs	0,02 µs	0,086 µs	0,4 µs	1 ms	77,1 s
30	0,005 µs	0,03 µs	0,147 µs	0,9 µs	1 s	8,4 × 10^{15} years
40	0,005 µs	0,04 µs	0,213 µs	1,6 µs	18,3 min	
50	0,006 µs	0,05 µs	0,282 µs	2,5 µs	13 days	
100	0,007 µs	0,1 µs	0,644 µs	10 µs	4 × 10^{13} years	
1000	0,010 µs	1,00 µs	9,966 µs	1 ms		
10 000	0,013 µs	10 µs	130 µs	100 ms		
100 000	0,017 µs	0,10 ms	1,67 ms	10 s		
1 000 000	0,020 µs	1 ms	19,93 ms	16,7 min		
10 000 000	0,023 µs	0,01 sec	0,23 s	1,16 days		
100 000 000	0,027 µs	0,10 sec	2,66 s	115,7 days		
1 000 000 000	0,030 µs	1 sec	29,90 s	31,7 years		

The Big Oh Notation

- Example algorithms and their complexities:
  - Retrieving the k^{th} element of an array – $O(1)$
  - Binary search - $O(log\ n)$
  - Finding maximum/minimum value in an unordered array - $O(n)$
  - Merge sort - $O(n\ log\ n)$
  - Bubble sort - $O(n^2)$
  - Recursive Fibonacci algorithm - $O(2^n)$
  - Exhaustive search for the shortest route between n cities (the Travelling Salesman Problem) – $O(n!)$

Complexity Classes

- P (polynomial) – decision problems that can be solved using a polynomial amount of computation time. They are considered as efficiently solvable
- NP (nondeterministic polynomial) – decision problems for which the problem instances, where the answer is "yes", have proofs verifiable in polynomial time
- NPC (NP-complete) - the hardest of the problems to which solutions can be verified quickly. If we could find solutions of some NP-complete problem quickly, we could quickly find the solutions of every other NP problem since every problem in NP is reducible to any NPC problem in polynomial time
- NPH (NP-hard) – every problem for which exists a polynomial-time reduction to any NP problem. Consequently, finding a polynomial time algorithm to solve a single NP-hard problem would give polynomial time algorithms for all the problems in the complexity class NP

Complexity Classes

- Relation between sets P, NP, NPC, and NPH under the assumption P ≠ NP