Wrocław University of Science and Technology

The Efficiency of Algorithms

Information Technologies

Wojciech Myszka

Department of Mechanics, Materials and Biomedical Engineering

January 2023

Introduction

Let’s think about building a bridge I

When asked to construct a bridge over a river, it is easy to construct an “incorrect” one:

1. The bridge might not be wide enough for the required lanes,
2. it might not be strong enough to carry rush-hour traffic, or
3. it might not reach the other side at all!

However, even if it is “correct,” in the sense that it fully satisfies the operational requirements, not every candidate design for the bridge will be acceptable:

- It is possible that the design calls for too much
  - manpower, or
  - too many materials or
  - components.
- It might also require far too much time to bring to completion.

Let's think about building a bridge II

In other words, although it will result in a good bridge, a design might be **too expensive**

Software

Creating a software

1. The same problems as above
2. *Incorrect* algorithms are bad
3. Even a *correct* algorithm might leave much to be desired.

Example

Fibonacci series

$$f(0) = 1; \qquad f(1) = 1; \qquad f(n) = f(n-2) + f(n-1)$$

Recursive

n	time
10	0,003s
20	0,003s
30	0,016s
40	1,265s
45	14,016s

Non-Recursive

n	time
10	0,003s
20	0,003s
30	0,003s
40	0,003s
45	0,003s

Efficiency criteria

1. **Complexity measures** of memory space (or simply space)

   and

2. **time.**

Space

The first of these is measured by several things, including the number of variables, and the number and sizes of the data structures used in executing the algorithm.

Time

The other is measured by the number of elementary actions carried out by the processor in such an execution.

Computer speed

Moore's Law I

Moore's law is the observation (made in 1965 by Gordon Moore from Intel) that the number of transistors in a dense integrated circuit (IC) doubles about every two years. Moore's law is an observation and projection of a historical trend. Rather than a law of physics linked to gains from experience in production.

Moore's Law II

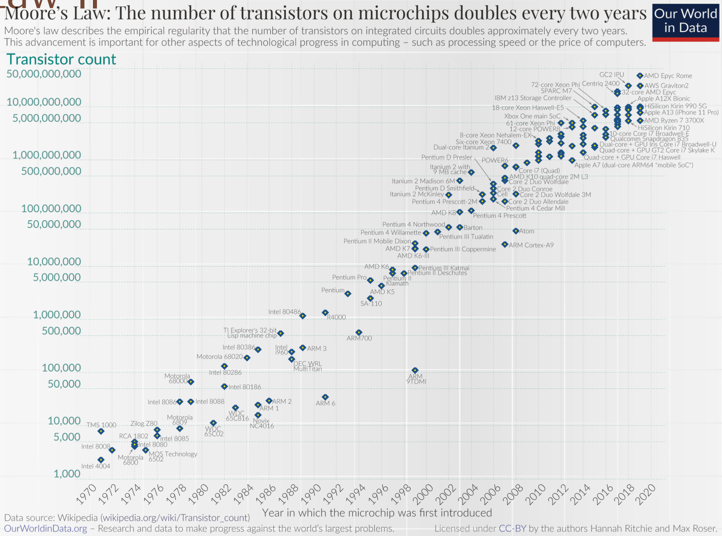


Figure 1: By Max Roser, Hannah Ritchie, CC BY 4.0,

Moore’s Law III

Wrocław University of Science and Technology

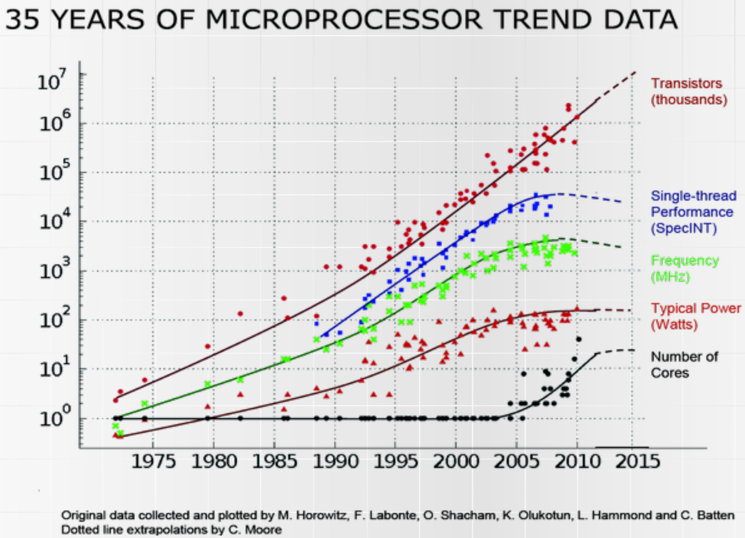
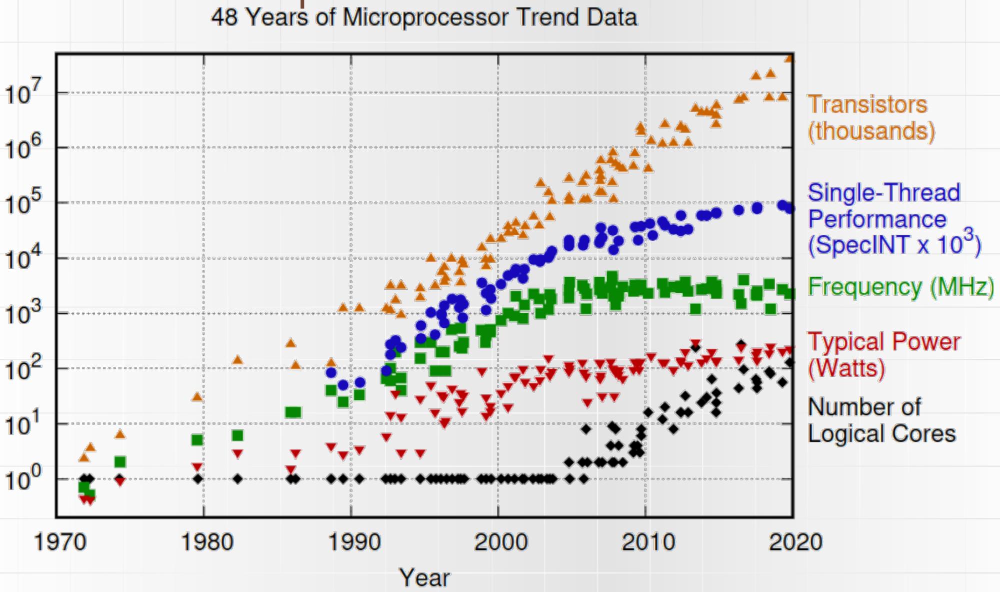


Figure 2: 40 Years of Microprocessor Trend Data

48 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten

Battery capacity vs processor performance

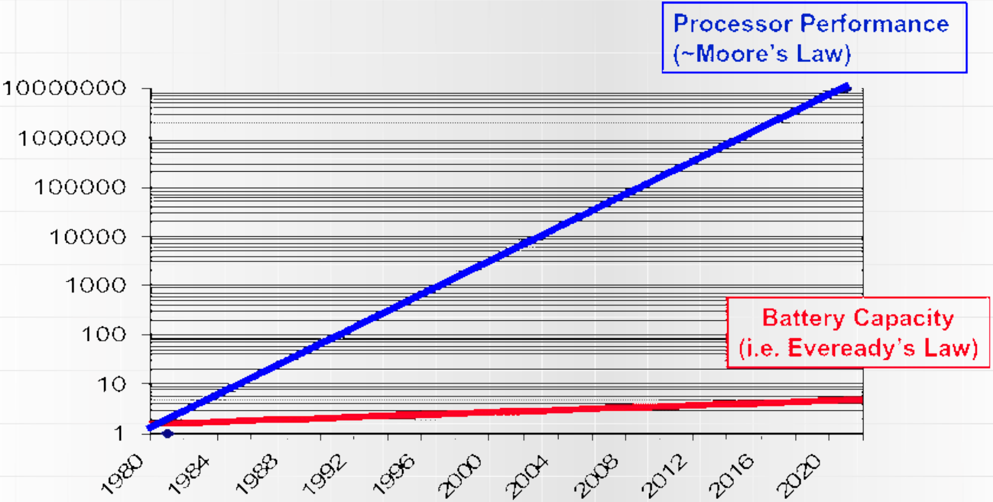
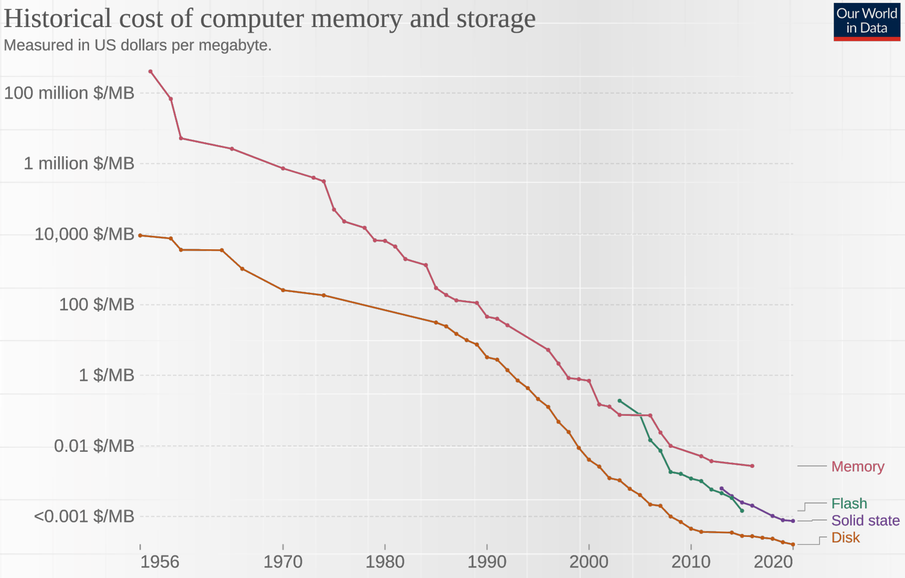


Figure 3: Energy Harvesting for Structural Health Monitoring Sensor Networks

Wrocław University of Science and Technology

Historical cost of computer memory and storage



Problems

1. We are interested in finding the **shortest** route for a traveler who wishes to visit each of, say, 200 cities. As of now, there is no computer that can find the route in fewer than millions of years of computing time!
2. No computer is capable of factoring (that is, finding the prime numbers that divide) large integers, say, 300 digits long, in fewer than millions of years.

Improvements

Ways for improving computation I

Transferring instructions from the inside to the outside of loops

Assume that a teacher wants to normalize the list of grades, by giving the student who scored best in the exam 100 points and upgrading the rest accordingly. The algorithm is simple:

(1) compute the maximum score in MAX;
(2) multiply each score by 100 and divide it by MAX.

```
for I from 1 to N do:
   L(I) ← L(I) × 100/MAX
```

this can be improved this way

```
FACTOR ← 100/MAX;
for I from 1 to N do:
   L(I) ← L(I) × FACTOR
```

Wrocław University of Science and Technology

Ways for improving computation II

Searching for an element X in an unordered list

(say for a telephone number in a jumbled telephone book)
The standard algorithm calls for a simple loop, within which two tests are carried out:

(1) "have we found X?" and
(2) "have we reached the end of the list?"

A positive answer to any one of these questions causes the algorithm to terminate—successfully in the first case and unsuccessfully in the second.
How to improve this algorithm?

Complexity

Binary search I

- For concreteness, let us assume that the telephone book contains a million names, that is, N is 1,000,000, and let us call them $X_1, X_2, \ldots, X_{1,000,000}$. We are searching for Y.
- A naive algorithm that searches for Y's telephone number is the one previously described for an unsorted list: work through the list L one name at a time, checking Y against the current name at each step, and checking for the end of the list at the same time.
- The first comparison carried out by the **new algorithm** is not between Y and the first or last name in L, but between Y and the middle name (or, if the list is of even length, then the last name in the first half of the list), namely $X_{500,000}$.

Binary search II

- Assuming that the compared names turn out to be unequal, meaning that we are not done yet, there are two possibilities:
  (1) Y precedes $X_{500{,}000}$ in alphabetic order, and
  (2) $X_{500{,}000}$ precedes Y.
- Since the list is sorted alphabetically, if
  (1) is the case we know that if Y appears in the list at all it has to be in the **first half**, and if
  (2) is the case it must appear in the **second half**.
- Hence, we can restrict our successive search to the appropriate half of the list.
- The next comparison will be between Y and the middle element of that half:

Binary search III

- $X_{250,000}$ in case (1) and
- $X_{750,000}$ in case (2).

This process continues, reducing the length of the list, or in more general terms, the **size of the problem**, by half at each step.

This procedure is called **binary search**, and it is really an application of the **divide-and-conquer** paradigm discussed ealier

Binary search IV

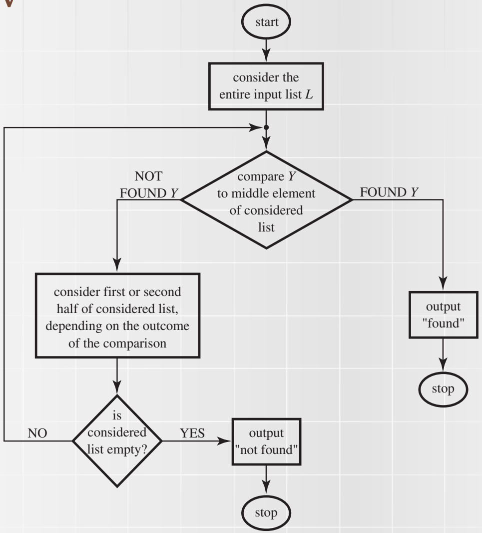


Figure 4: Block diagram

Binary search V

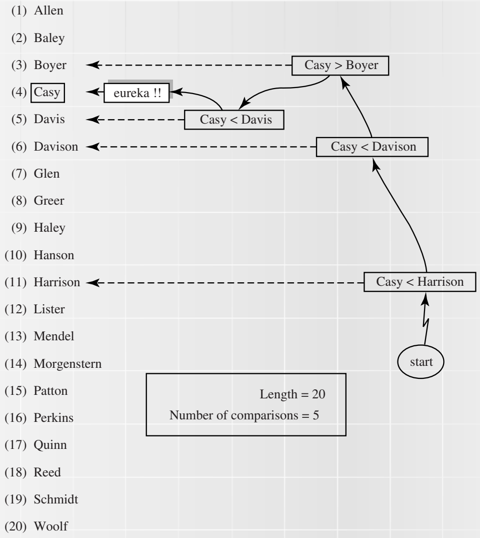


Figure 5: Example

Binary search VI

Wrocław University of Science and Technology

Comparing the speed I

1. Very difficult problem
2. Execution time of the normalizing students scores (in both cases) is proportional to the number of students (in the second case shorter, but also proportional to number of students)
3. Time of the linear search algorithm depends on:
   3.1 the list length, and
   3.2 the data.
4. In the **worst case** (searched value not in the list) is proportional to the list length.
5. Improved version (shorter), but in the worst case is proportional to the list length.
6. Execution time of the binary search algorithm (in the worst case) is proportional to the base two logarithm of the list length.

Comparing the speed II

the big-O notation

instead of telling that execution time is proportional to N we will use term

$$O(N)$$

This means that when the size of the problem increases, the (worst case) time increases proportionally. So the binary search algorithm execution time is in range of $O(\log_2 N)$.
This means that when the list length doubles execution time increases by 1 (time of one elementary search operation).

How much $O(\log_2 N)$ is better than $O(N)$?

N	$\log_2 N$
10	4
100	7
1000	10
a million	20
a billion	30
a billion billions	60

Time Analysis of Nested Loops

Sometimes we have something like this:

1. do the following $N-1$ times:

   ...

   1.1 do the following $N-1$ times:

   ...

- The total time performance of this algorithm is on the order of $(N-1)\times(N-1)$, which is N^2-2N+1.
- The N^2 is called the dominant term of the expression, meaning that the other parts, namely, the $-2N$ and the $+1$, get "swallowed" by the N^2 when the big-O notation is used. Consequently, this algorithm is an

  $$O(N^2),$$

  or quadratic-time, algorithm.

Time Analysis of Recursion I

Let us now consider the min&max problem for finding the extremal elements in a list L. The naive algorithm runs through the list iteratively, updating two variables that hold the current extremal elements. It is clearly linear. Here is the recursive routine, which was claimed to be better:

subroutine `find-min&max-of` L:

1. if L consists of one element, then set MIN and MAX to it; if it consists of two elements, then set MIN to the smaller of them and MAX to the larger;
2. otherwise do the following:
   2.1 split L into two halves, L_{left} and L_{right};
   2.2 call `find-min&max-of` L_{left}, placing returned values in MIN_{left} and MAX_{left};

Time Analysis of Recursion II

2.3 call `find-min&max-of` L_{right}, placing returned values in MIN_{right} and MAX_{right}

2.4 set MIN to smaller of MIN_{left} and MIN_{right};

2.5 set MAX to larger of MAX_{left} and MAX_{right};

3. return with MIN and MAX.

- The iterative algorithm operates by carrying out two comparisons for each element in the list, one with the current maximum and one with the current minimum. Hence it yields a total comparison count of $2N$.
- Let $C(N)$ denote the (worst-case) number of comparisons required by the recursive `min&max` routine on lists of length N.
  1. If N is 2, precisely one comparison is carried out—the one implied by line 1 of the routine; if N is 3, three comparisons are carried out, as you can verify.

Time Analysis of Recursion III

2. If N is greater than 3, the comparisons carried out consist precisely of two sets of comparisons for lists of length $N/2$, since there are two recursive calls, and two additional comparisons—those appearing on lines (2.4) and (2.5). (If N is odd, the lists are of length $(N+1)/2$ and $(N-1)/2$.)

- We can write this:
  - $C(2) = 1$
  - $C(N) = 2 \times C(N/2) + 2$
- And solve as:

$$C(N) = 3N/2 - 2$$

(in case where N is power of two)

Still $O(N)$!

The Towers of Hanoi Revisited

The time to solve this puzzle is

$$O(2^N)$$

where N is number of rings.

The Monkey Puzzle Problem I

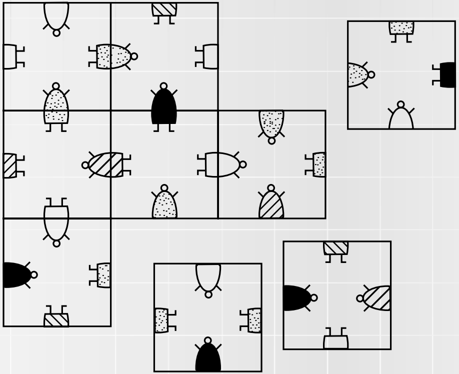


Figure 6: The Monkey Puzzle

- This puzzle involves nine square cards whose sides are imprinted with the upper and lower halves of colored monkeys.

The Monkey Puzzle Problem II

- The objective is to arrange the cards in the form of a 3 by 3 square such that halves match and colors are identical wherever edges meet.
- We shall assume that the **cards are oriented,** meaning that the edges have fixed directions, “up,” “down,” “right,” and “left,” so that they are not to be rotated.
- A naive solution to the problem is not too hard to come by.
- We need only observe that each input involves only finitely many cards, and that there are only finitely many locations to fill with them.
- Hence, there are only finitely many different ways of arranging the input cards into an M by M square.

The Monkey Puzzle Problem III

- On the first step, the first card can be placed on $N = M \times M$ locations, second car on $N - 1$ locations, and so on. In general we have
$$N \times (N-1) \times (N-2) \times \cdots \times 2 \times 1 = N!$$
arrangements.

The time needed to solve this puzzle is

$$O(N!)$$

.

Reasonable vs. Unreasonable Time I

	Function \ N	20	60	100	300	1000
Polynomial	$5N$	100	300	500	1500	5000
	$N \times \log_2 N$	86	354	665	2469	9966
	N^2	400	3600	10,000	90,000	1 million (7 digits)
	N^3	8000	216,000	1 million (7 digits)	27 million (8 digits)	1 billion (10 digits)
Exponential	2^N	1,048,576	a 19-digit number	a 31-digit number	a 91-digit number	a 302-digit number
	$N!$	a 19-digit number	an 82-digit number	a 161-digit number	a 623-digit number	unimaginably large
	N^N	a 27-digit number	a 107-digit number	a 201-digit number	a 744-digit number	unimaginably large

Figure 7: Some values of some functions.

Reasonable vs. Unreasonable Time II

For comparison: the number of protons in the known universe has 79 digits; the number of nanoseconds since the Big Bang has 27 digits.

Reasonable vs. Unreasonable Time III

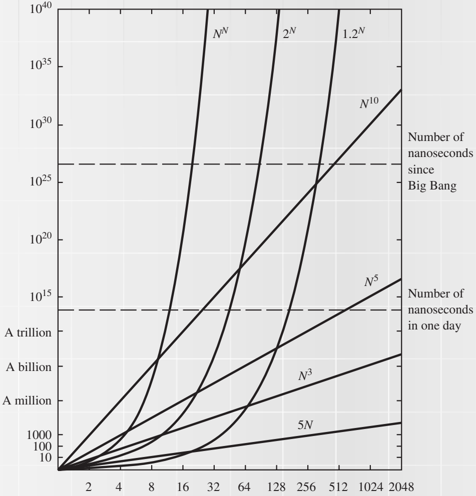


Figure 8: Growth rates of some functions

Reasonable vs. Unreasonable Time IV

	Function \ N	20	40	60	100	300
Polynomial	N^2	1/2500 millisecond	1/625 millisecond	1/278 millisecond	1/100 millisecond	1/11 millisecond
	N^5	1/300 second	1/10 second	78/100 second	10 seconds	40.5 minutes
Exponential	2^N	1/1000 second	18.3 minutes	36.5 years	400 billion centuries	a 72-digit number of centuries
	N^N	3.3 billion years	a 46-digit number of centuries	an 89-digit number of centuries	a 182-digit number of centuries	a 725-digit number of centuries

Figure 9: Time consumption of hypothetical solutions to the monkey puzzle problem (assuming one instruction per nanosecond)

Reasonable vs. Unreasonable Time V

- These facts lead to a fundamental classification of functions into "good" and "bad" ones.
- The distinction to be made is between polynomial and super-polynomial functions.
- For our purposes a polynomial function of N is one which is bounded from above by N^K for some fixed K (meaning, essentially, that it is no greater in value than N^K for all values of N from some point on).
- All others are super-polynomial.

Is it all real?

1. Computers are becoming faster by the week. Over the past 10 years or so computer speed has increased roughly by a factor of 50. **Perhaps obtaining a practical solution to the problem is just a question of awaiting an additional improvement in computer speed**.
2. **Doesn't the fact that we have not found a better algorithm for this problem indicate our incompetence at devising efficient algorithms**? Shouldn't computer scientists be working at trying to improve the situation rather than spending their time writing books about it?
3. Haven't people tried to look for an exponential-time lower bound on the problem, so that **we might have a proof that no reasonable algorithm exists?**
4. **Maybe the whole issue is not worth the effort, as the monkey puzzle problem is just one specific problem**. It might be a colorful one, but it certainly doesn't look like a very important one.

Objection number 1

Function	Maximal number of cards solvable in one hour:		
	with today's computer	with computer 100 times faster	with computer 1000 times faster
N	A	$100 \times A$	$1000 \times A$
N^2	B	$10 \times B$	$31.6 \times B$
2^N	C	$C + 6.64$	$C + 9.97$

Figure 10: Algorithmic improvements resulting from improvements in computer speed

Objection number 4

- It so happens that the monkey puzzle problem is not alone.
- There are other problems in the same boat. Moreover, the boat is large, impressive, and many-sided.
- The monkey puzzle problem is just one of close to 1000 diverse algorithmic problems, all of which exhibit precisely the same phenomena.
- They all admit unreasonable, exponential-time solutions, but none of them is known to admit reasonable ones.

Solutions?

1. Parallel computing?
2. Quantum Computing?