



UNIVERSITY
OF TRIESTE

Algorithmic Design

University of Trieste – a.a. 2026/2027

Prof. Antonio Emanuele Cinà

`antonio.cina@units.it`

Department of Mathematics, Informatics and Geosciences

University of Trieste

a.a. 2026/2027

Chapter 2:

Asymptotic computational complexity

Making the analysis choices explicit

Chapter 1 compared methods for the same Fibonacci problem:

- Direct recursion makes exponentially many calls.
- Iteration performs linearly many additions.
- Repeated squaring uses logarithmically many matrix multiplications under unit cost.

Each comparison chooses a charged operation, a size parameter, and an arithmetic model.

Notation used in the afternotes

$T(n)$ denotes algorithmic running time. $S(n)$ denotes auxiliary space.
In both cases, n is the declared input-size parameter.

Elapsed time and abstract operation counts

Observed execution time depends on the processor, programming language, compiler, operating system, and implementation details.

A faster measurement need not identify a better algorithm: changing the machine may change the comparison.

Cost criteria

Algorithm analysis replaces elapsed time with an abstract count of selected operations.

Before stating a cost, specify the resource being measured and how each operation is charged.

The uniform-cost model

Definition (Uniform-cost model)

In the *uniform-cost model*, elementary operations on machine-sized values are charged a constant cost. Comparisons, assignments, arithmetic operations, and array accesses are typical examples.

The model is a useful first approximation when operands fit into a fixed machine word. It keeps the analysis focused on the structure of the algorithm.

The logarithmic-cost model

Definition (Logarithmic-cost model)

In the *logarithmic-cost model*, the cost of an operation depends on the number of bits needed to represent its operands.

Adding two b -bit integers examines a number of bits proportional to b under the usual addition algorithm.

The cost of this operation depends on operand size.

Numerical value and representation length

A positive integer m requires

$$b = \lfloor \log_2 m \rfloor + 1$$

bits in binary. Thus

$$2^{b-1} \leq m < 2^b.$$

An algorithm performing m iterations is linear in the value m , but its worst-case running time is exponential in the binary length b .

Chapter 1 uses the Fibonacci index n directly as the analysis parameter. Calling an algorithm linear is incomplete without stating that parameter.

Fibonacci costs depend on the arithmetic model

Under uniform cost, multiplying fixed-size 2×2 matrices takes constant time.

Under bit cost, the Fibonacci entries grow with n , so arithmetic does not have constant bit cost.

For $n \geq 3$, optimized iteration performs $n - 2$ additions. Each has a constant cost in the uniform model, but growing operands make later additions more expensive under bit cost.

A consistent analysis

Neither model is universally correct. State the charged resource and apply the model consistently.

Choosing the input-size parameters

Definition (Input size)

The *input size* is the length of the representation of an input, or a declared collection of representation parameters, relative to which the resource usage of an algorithm is measured.

Input	Usual parameter
Array	Number of elements n
String	Number of characters n
Graph	Both $ V $ and $ E $

The number of vertices alone does not determine the amount of graph data.

Best-case and worst-case costs

Let $\mathcal{I}_n$ be the set of valid inputs of size n , and let $T(I)$ be the cost on input I .

Definition (Best-case cost)

The best-case cost is

$$T_{\text{best}}(n) = \min_{I \in \mathcal{I}_n} T(I).$$

It is the smallest cost incurred by an input of size n .

Definition (Worst-case cost)

The worst-case cost is

$$T_{\text{worst}}(n) = \max_{I \in \mathcal{I}_n} T(I).$$

It is the largest cost incurred by an input of size n .

Different instances of the same size can cause different executions.

Average-case cost requires a distribution

Definition (Average-case cost)

Let $p_n(l)$ be the probability that the input of size n is l . The average-case cost is

$$T_{\text{avg}}(n) = \sum_{l \in \mathcal{I}_n} p_n(l) T(l).$$

An average-case statement is incomplete unless p_n is specified.

Best, worst, and average costs answer different questions. Unless another case or probability model is stated, this course uses the **worst case** as its default.

Sequential search

Compare the key with each array entry from left to right. Return after finding a match or exhausting the array.

```
Sequential-search(A, key)
  for i <- 0 to length(A)-1
    if A[i] = key
      return i
  return Not-found
```

The returned index is zero-based.

Counting comparisons in sequential search

For an array of length $n \geq 1$:

Best case

The first entry contains the key. The algorithm makes one comparison.

Worst case

The first match is in the final position, or the key does not occur. The algorithm makes n comparisons.

For the empty array, the loop executes zero times and returns Not - found.

A uniform successful search

Suppose the search is successful and each of the n positions is equally likely to be the matching position.

A match at position $i - 1$ requires i comparisons, for $i = 1, \dots, n$.

$$\mathbb{E}[\text{comparisons}] = \frac{1}{n} \sum_{i=1}^n i = \frac{n+1}{2}.$$

Probability-model clarification

Here the matching position means the first match returned by the algorithm. Its position is uniform, and $n \geq 1$.

Eventual growth of cost functions

Definition (Asymptotic complexity)

The *asymptotic complexity* describes the eventual growth of $T(n)$ for an algorithm by comparing it with a reference function up to positive constant factors.

An exact cost such as $7n^2 + 3n + 10$ includes implementation-dependent constants and lower-order terms. Asymptotic analysis retains the growth that dominates for sufficiently large inputs.

Let $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$, with $g(n) > 0$ for all sufficiently large n .

- f is the function under study.
- g is a reference growth rate.

Asymptotic upper bounds

Definition (Asymptotic upper bound)

$$O(g(n)) = \left\{ z(n) : \begin{array}{l} \exists c > 0 \wedge n_0 \geq 0 : \\ z(n) \leq c g(n), \quad \forall n \geq n_0 \end{array} \right\}.$$

$f(n) = O(g(n))$ means that f grows no faster than g , up to a constant factor, once n is sufficiently large.

The function g is an asymptotic upper bound for f .

Asymptotic lower bounds

Definition (Asymptotic lower bound)

$$\Omega(g(n)) = \left\{ z(n) : \begin{array}{l} \exists c > 0 \wedge n_0 \geq 0 : \\ c g(n) \leq z(n), \quad \forall n \geq n_0 \end{array} \right\}.$$

$f(n) = \Omega(g(n))$ means that f grows at least as fast as g , up to a constant factor, once n is sufficiently large.

Asymptotically tight bounds

Definition (Asymptotically tight bound)

$$\Theta(g(n)) = \left\{ z(n) : \begin{array}{l} \exists c_1, c_2 > 0 \wedge n_0 \geq 0 : \\ c_1 g(n) \leq z(n) \leq c_2 g(n), \quad \forall n \geq n_0 \end{array} \right\}.$$

$f(n) = \Theta(g(n))$ means that f and g have the same asymptotic order of growth. Beyond a threshold, f lies between two positive constant multiples of g .

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

Constants, thresholds, and sets of functions

The constants c, c_1, c_2 must not depend on n . The threshold n_0 permits different behavior on a finite initial part of the domain.

$O(g)$, $\Omega(g)$, and $\Theta(g)$ are **sets of functions**.

$$f(n) = O(g(n)) \text{ abbreviates } f \in O(g).$$

This conventional notation is not an ordinary symmetric equality:

$$n = O(n^2), \quad n^2 \neq O(n).$$

Asymptotic notation describes eventual growth, not necessarily the first few values.

Finding the crossover point

Consider the modeled costs

$$T_{\text{linear}}(n) = 100n, \quad T_{\text{quadratic}}(n) = n^2.$$

At a crossover,

$$100n = n^2 \iff n^2 - 100n = 0 \iff n(n - 100) = 0.$$

The solutions are $n = 0$ and $n = 100$. The origin is the trivial intersection; the relevant positive crossover is $n = 100$.

$$T_{\text{linear}}(100) = T_{\text{quadratic}}(100) = 10,000.$$

Which modeled cost is smaller?

Compare the difference:

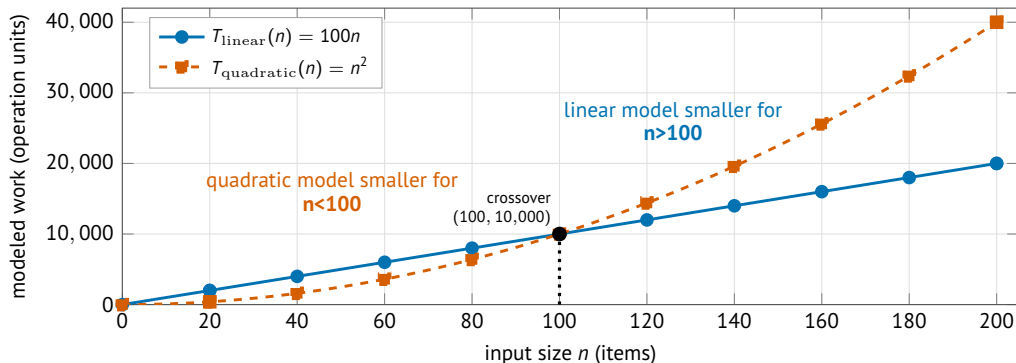
$$T_{\text{quadratic}}(n) - T_{\text{linear}}(n) = n^2 - 100n = n(n - 100).$$

For positive input sizes, the sign depends only on $n - 100$:

$$\begin{cases} n^2 < 100n, & 0 < n < 100, \\ n^2 = 100n, & n = 100, \\ n^2 > 100n, & n > 100. \end{cases}$$

The quadratic model is smaller before the crossover. The linear model is smaller after it.

The crossover in the two cost models



At $n = 20$: $100n = 2,000$, $n^2 = 400$. At $n = 200$: $100n = 20,000$, $n^2 = 40,000$.

The linear model $100n$ becomes smaller than n^2 only after $n = 100$, despite its better asymptotic growth.

The role of an eventual domain

The next example uses

$$f(n) = \frac{1}{2}n^2 - 3n.$$

It is negative for $0 < n < 6$, although the initial setup assumed $f : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$.

Source clarification

For this example and the later properties, apply the defining inequalities on a sufficiently large tail, where the functions are nonnegative.

This convention matches the later proposition's explicit assumption of eventual nonnegativity. Its inequalities also cover functions with zero values. The ratio-limit theorem will separately require eventual positivity.

Constructing a quadratic upper bound

Consider $f(n) = \frac{1}{2}n^2 - 3n$. The goal is to find $c_1, c_2 > 0$ and n_0 such that

$$c_1 n^2 \leq f(n) \leq c_2 n^2 \quad \forall n \geq n_0.$$

The dominant-term intuition suggests $g(n) = n^2$, but intuition alone is not a proof.

Step 1. Upper bound

Since $-3n \leq 0$ for every $n \geq 0$,

$$f(n) = \frac{1}{2}n^2 - 3n \leq \frac{1}{2}n^2.$$

Thus $c_2 = \frac{1}{2}$ works for all $n \geq 0$.

The dash-dotted upper curve in the figure remains above f , including near the origin.

Deriving a lower-bound threshold

For $n > 0$, rearrange the desired inequality:

$$\begin{aligned} c_1 n^2 \leq \frac{1}{2} n^2 - 3n &\iff n^2 \left(\frac{1}{2} - c_1 \right) \geq 3n \\ &\iff n \left(\frac{1}{2} - c_1 \right) \geq 3. \end{aligned}$$

The last step divides by $n > 0$. To isolate a lower threshold for n , the coefficient must be strictly positive:

$$0 < c_1 < \frac{1}{2}, \quad n_0 = \frac{3}{\frac{1}{2} - c_1}.$$

If $c_1 \geq \frac{1}{2}$, no threshold works: the coefficient is zero or negative.

A smaller admissible c_1 gives an earlier threshold. Values closer to $\frac{1}{2}$ push the threshold further out.

Choosing explicit witnesses

Choose $c_1 = \frac{1}{4}$ for a clean integer threshold. For $n > 0$,

$$\begin{aligned}\frac{1}{4}n^2 \leq \frac{1}{2}n^2 - 3n &\iff \frac{1}{4}n^2 \geq 3n \\ &\iff \frac{1}{4}n \geq 3 \\ &\iff n \geq 12.\end{aligned}$$

The lower bound fails at small positive n . The chosen c_1 determines a sufficient threshold; it is not an arbitrary choice independent of the inequality.

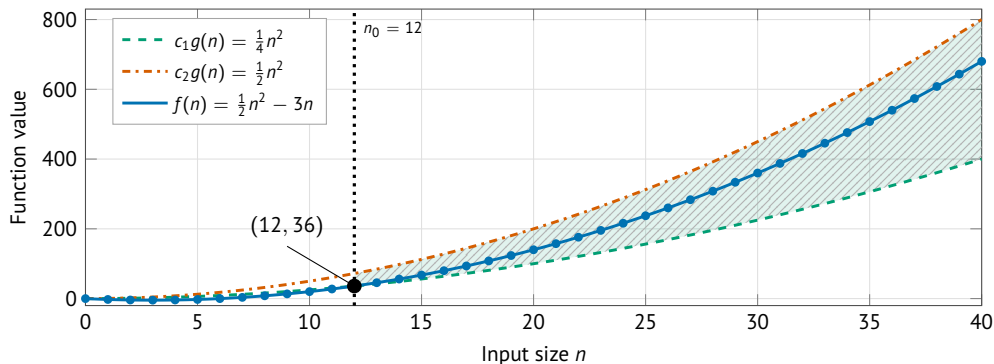
Step 3. Conclusion

$c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$, and $n_0 = 12$ give

$$\frac{1}{4}n^2 \leq f(n) \leq \frac{1}{2}n^2 \quad \forall n \geq 12.$$

Hence $f(n) = \Theta(n^2)$.

The witnesses beyond the threshold



$(c_1, c_2, n_0) = (1/4, 1/2, 12)$: $c_1g(n) \leq f(n) \leq c_2g(n)$ for $n \geq 12$.

The dashed lower curve lies above f for small positive inputs. From $n_0 = 12$ onward, the two witnesses trap f in the shaded, hatched band.

Reflexivity and constant scaling

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, properties 1 and 2

$$f(n) = O(f(n)), \quad f(n) = \Omega(f(n)), \quad f(n) = \Theta(f(n)), \\ af(n) = \Theta(f(n)).$$

Proof. Where $f(n) \geq 0$,

$$f(n) \leq f(n) \leq f(n).$$

Choosing both constants as 1 proves the tight bound and its upper and lower parts. For scaling, $af(n) \leq af(n) \leq af(n)$ uses both constants equal to a .

A positive constant factor changes numerical values but not asymptotic growth. □

Chaining asymptotic upper bounds

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, property 3 (upper bounds)

$f(n) = O(g(n))$ and $g(n) = O(h(n))$ imply $f(n) = O(h(n))$.

Proof. Positive constants c_1, c_2 and thresholds n_1, n_2 give

$$f(n) \leq c_1 g(n) \quad (n \geq n_1), \quad g(n) \leq c_2 h(n) \quad (n \geq n_2).$$

For $n \geq \max\{n_1, n_2\}$, both hold:

$$f(n) \leq c_1 g(n) \leq c_1 c_2 h(n).$$

The positive constant $c_1 c_2$ and threshold $\max\{n_1, n_2\}$ witness $f(n) = O(h(n))$. □

Transitivity for lower and tight bounds

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, property 3 (continued)

$$\begin{aligned} f(n) = \Omega(g(n)), g(n) = \Omega(h(n)) &\implies f(n) = \Omega(h(n)), \\ f(n) = \Theta(g(n)), g(n) = \Theta(h(n)) &\implies f(n) = \Theta(h(n)). \end{aligned}$$

Proof. For Ω , reverse the inequalities in the upper-bound proof:

$$f(n) \geq c_1 g(n) \geq c_1 c_2 h(n) \quad \text{for } n \geq \max\{n_1, n_2\}.$$

The constants are positive.

A Θ bound has both an upper and a lower bound. Apply the two transitivity results to obtain transitivity for Θ . □

Upper and lower bound duality

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, property 4

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n)).$$

Proof. If $f(n) = O(g(n))$, some $c > 0$ gives, for all sufficiently large n ,

$$f(n) \leq cg(n) \implies \frac{1}{c}f(n) \leq g(n).$$

The latter inequality is a lower bound for g in terms of f .

The converse follows by the same reasoning in the opposite direction. □

Combining upper and lower bounds

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, property 5

$f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Forward direction. The witnesses for a tight bound give

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad (n \geq n_0).$$

The left inequality is the lower bound, and the right is the upper bound.

Reverse direction. Suppose the lower bound uses $c_1 > 0, n_1$ and the upper bound uses $c_2 > 0, n_2$. For $n \geq \max\{n_1, n_2\}$, both hold simultaneously:

$$c_1 g(n) \leq f(n) \leq c_2 g(n).$$

These are exactly the witnesses for Θ .



Symmetry of a tight bound

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, property 6

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n)).$$

Proof. Rewrite the tight-bound inequalities, using $c_1, c_2 > 0$:

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \implies \frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n).$$

The reciprocal constants are positive, so $g(n) = \Theta(f(n))$.

The same argument with f and g exchanged proves the converse. □

The dominant term in a sum

Let f, g, h be eventually nonnegative functions, and let $a > 0$ be a constant.

Proposition, property 7

$$f(n) + g(n) = \Theta(\max\{f(n), g(n)\}).$$

Proof. Set $m(n) = \max\{f(n), g(n)\}$.

- Eventual nonnegativity gives $m(n) \leq f(n) + g(n)$.
- Both terms are at most $m(n)$, so $f(n) + g(n) \leq 2m(n)$.

Thus, eventually,

$$m(n) \leq f(n) + g(n) \leq 2m(n).$$

The constants 1 and 2 prove $f(n) + g(n) = \Theta(m(n))$. □

Simplifying a sum by its growth

If one term grows asymptotically faster than all others, it determines the growth of the entire sum. For example,

$$n^2 + 7n + 4 = \Theta(n^2),$$

because n^2 eventually dominates $7n$ and the constant term.

More generally, if $f(n) = O(g(n))$ and both functions are eventually nonnegative,

$$f(n) + g(n) = \Theta(g(n)).$$

The smaller term can affect concrete running times without changing the asymptotic growth class.

Comparison by a ratio limit

The ratio $f(n)/g(n)$ measures the size of f relative to g . Constants and thresholds remain the formal basis; limits often give a faster comparison.

Theorem (Comparison by a ratio limit)

Let f and g be eventually positive functions.

$$\begin{aligned}\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 &\implies f(n) = O(g(n)), \\ \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = L \in (0, \infty) &\implies f(n) = \Theta(g(n)), \\ \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = +\infty &\implies f(n) = \Omega(g(n)).\end{aligned}$$

A zero limit makes f negligible relative to g ; a positive finite limit gives equal order; an infinite limit makes f dominate g .

Proof for zero and infinite ratio limits

Assume f and g are eventually positive.

Limit zero

By the definition of the limit, some n_0 gives

$$\frac{f(n)}{g(n)} \leq 1 \quad \forall n \geq n_0.$$

Multiplying by $g(n) > 0$ gives $f(n) \leq g(n)$, so $f(n) = O(g(n))$ with $c = 1$.

Limit $+\infty$

The ratio eventually exceeds every fixed positive value, in particular 1:

$$\frac{f(n)}{g(n)} \geq 1 \quad \forall n \geq n_0.$$

Thus $f(n) \geq g(n)$, proving $f(n) = \Omega(g(n))$.

Proof for a positive finite ratio limit

Assume $\lim_{n \rightarrow \infty} f(n)/g(n) = L$, where $0 < L < \infty$. Choose $\varepsilon = L/2$.

For every sufficiently large n ,

$$L - \frac{L}{2} \leq \frac{f(n)}{g(n)} \leq L + \frac{L}{2},$$

so

$$\frac{L}{2} \leq \frac{f(n)}{g(n)} \leq \frac{3L}{2}.$$

Multiplication by the eventually positive function $g(n)$ gives

$$\frac{L}{2}g(n) \leq f(n) \leq \frac{3L}{2}g(n).$$

The positive constants $L/2$ and $3L/2$ witness $f(n) = \Theta(g(n))$. □

Comparing linearithmic and quadratic growth

Let

$$f(n) = n \log n, \quad g(n) = n^2.$$

Then

$$\frac{f(n)}{g(n)} = \frac{n \log n}{n^2} = \frac{\log n}{n}.$$

Since $\lim_{n \rightarrow \infty} (\log n)/n = 0$, the theorem gives

$$n \log n = O(n^2).$$

The limit says more than a non-strict upper bound: $n \log n$ becomes an increasingly small fraction of n^2 .

The highest degree determines polynomial growth

Let

$$P(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0, \quad a_k > 0.$$

Divide by n^k :

$$\frac{P(n)}{n^k} = a_k + \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \cdots + \frac{a_0}{n^k}.$$

Every fraction after a_k tends to zero, so

$$\lim_{n \rightarrow \infty} \frac{P(n)}{n^k} = a_k.$$

The limit is positive and finite. Hence $P(n) = \Theta(n^k)$.

Changing a fixed logarithm base

For any two fixed bases $a, b > 1$,

$$\log_a n = \frac{\log_b n}{\log_b a}.$$

The multiplier $1/\log_b a$ is a positive constant, independent of n .

Consequently,

$$\log_a n = \Theta(\log_b n).$$

This is why the base is normally omitted in asymptotic expressions.

Ranking standard growth rates

For eventually positive functions, write

$$f \prec g \quad \text{when} \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$$

This means that f grows strictly more slowly than g .

For fixed constants $0 < \varepsilon < k$ and $c > 1$,

$$1 \prec \log n \prec n^\varepsilon \prec n^k \prec c^n \prec n! \prec n^n.$$

The constants ε, k, c are independent of n . Allowing them to vary with the input can invalidate the comparisons.

Chapter 2 exercises 1–2

Choose the size and case, state the cost model, derive a count, and justify its asymptotic form with valid witnesses.

1. Design an experiment comparing the recursive and iterative Fibonacci algorithms. Separate the analytical prediction, deterministic operation counts, and wall-clock measurements.
2. Find explicit constants $c_1, c_2 > 0$ and a threshold n_0 such that

$$c_1 n^2 \leq 3n^2 + 7n + 4 \leq c_2 n^2$$

for every $n \geq n_0$.

Chapter 2 exercises 3–4

3. Analyse sequential search in the best and worst cases. Then derive its average cost when a successful position is uniform and failure has probability $1/4$.
4. Suppose that there exist constants $c_1, c_2 > 0$ and thresholds n_1, n_2 such that

$$f(n) \leq c_1 g(n) \quad \forall n \geq n_1,$$

and

$$g(n) \leq c_2 h(n) \quad \forall n \geq n_2.$$

Use these assumptions to prove that $f(n) = O(h(n))$. Then write the corresponding proof of transitivity for Ω without consulting the proposition.

Chapter 2 exercises 5–6

5. Order the following functions by asymptotic growth: $n \log n$, $n^{3/2}$, 2^n , n^3 , $\log n$, and $n!$. State every condition used.
6. Order the following functions from slowest to fastest asymptotic growth:

$$\begin{array}{ccccccc}
 n^2, & n \log n, & n^3 + \log n, & \sqrt{n}, \\
 n^2 + 2n \log n, & \log \log n, & 17 \log n, & 10n^{3/2}, \\
 n^5 - n^4 + 2n, & 5n^2 \log \log n, & 3n^2 + n^3 \log n, & n + 6 \log n, & 5.
 \end{array}$$

Place functions in the same group when they belong to the same Θ class. Justify each comparison and state every condition used.

Sources for further study

- Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms*, 3rd ed., Sections 2.2 and 3.1–3.2.
- Kleinberg and Tardos, *Algorithm Design*, Sections 2.1–2.4.
- Horowitz, Sahni, and Rajasekaran, *Computer Algorithms*, Section 1.3.

Next chapter in the afternotes: Searching, recursion, and recurrences.

Thank you!



UNIVERSITY
OF TRIESTE