Week 4

Solving Recurrences

General remarks

- We present a basic tool for analysing algorithms by **Solving Recurrences**.

Reading from CLRS for week 4

- Chapter 4

Analysing divide-and-conquer algorithms

Recall the divide-and-conquer paradigm:

Divide the problem into a number of subproblems, each of them a *smaller instance* of the same problem.

Conquer the subproblems by solving them *recursively*.
Base case: If the subproblems are small enough, just solve them by "brute force".

Combine the subproblem-solutions to give a solution to the original problem.

Analysing divide-and-conquer algorithms

Recall the divide-and-conquer paradigm:

Divide the problem into a number of subproblems, each of them a *smaller instance* of the same problem.

Conquer the subproblems by solving them *recursively*.
Base case: If the subproblems are small enough, just solve them by "brute force".

Combine the subproblem-solutions to give a solution to the original problem.

We use recurrences to characterise the running time of a divide-and-conquer algorithm. Solving the recurrence gives us the asymptotic running time.

Analysing divide-and-conquer algorithms

Recall the divide-and-conquer paradigm:

Divide the problem into a number of subproblems, each of them a *smaller instance* of the same problem.

Conquer the subproblems by solving them *recursively*. *Base case:* If the subproblems are small enough, just solve them by "brute force".

Combine the subproblem-solutions to give a solution to the original problem.

We use recurrences to characterise the running time of a divide-and-conquer algorithm. Solving the recurrence gives us the asymptotic running time.

A recurrence is a function defined in terms of

- one or more base cases, and
- itself, with smaller arguments.

Examples for recurrences

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$

  Solution: $T(n) = n$.

Examples for recurrences

- $$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$$

  Solution: $T(n) = n$.

- $$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 \cdot T(n-1) & \text{if } n > 0 \end{cases}$$

  Solution: $T(n) = 2^n$.

Examples for recurrences

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1)+1 & \text{if } n > 1 \end{cases}$

  Solution: $T(n) = n$.

- $T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 \cdot T(n-1) & \text{if } n > 0 \end{cases}$

  Solution: $T(n) = 2^n$.

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2)+n & \text{if } n > 1 \end{cases}$

  Solution: $T(n) \approx n \lg n + n$.

Examples for recurrences

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$

  Solution: $T(n) = n$.

- $T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 \cdot T(n-1) & \text{if } n > 0 \end{cases}$

  Solution: $T(n) = 2^n$.

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$

  Solution: $T(n) \approx n \lg n + n$.

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/3) + T(2n/3) + n & \text{if } n > 1 \end{cases}$

  Solution: $T(n) = \Theta(n \lg n)$.

Main technical issues with recurrences

Floors and ceilings: The recurrence describing worst-case running time of Merge-Sort is really

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Main technical issues with recurrences

Floors and ceilings: The recurrence describing worst-case running time of Merge-Sort is really

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Exact vs. asymptotic functions Sometimes we are interested in the exact analysis of an algorithm (as for the Min-Max-Problem), at other times we are concerned with the asymptotic analysis (as for the Sorting Problem).

Main technical issues with recurrences

Floors and ceilings: The recurrence describing worst-case running time of Merge-Sort is really

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Exact vs. asymptotic functions Sometimes we are interested in the exact analysis of an algorithm (as for the Min-Max-Problem), at other times we are concerned with the asymptotic analysis (as for the Sorting Problem).

Boundary conditions Running time on small inputs is bounded by a constant: $T(n) = \Theta(1)$ for small n. We usually do not mention this constant, as it typically doesn't change the order of growth of $T(n)$. Such constants only play a role if we are interested in exact solutions.

Main technical issues with recurrences

Floors and ceilings: The recurrence describing worst-case running time of Merge-Sort is really

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Exact vs. asymptotic functions Sometimes we are interested in the exact analysis of an algorithm (as for the Min-Max-Problem), at other times we are concerned with the asymptotic analysis (as for the Sorting Problem).

Boundary conditions Running time on small inputs is bounded by a constant: $T(n) = \Theta(1)$ for small n. We usually do not mention this constant, as it typically doesn't change the order of growth of $T(n)$. Such constants only play a role if we are interested in exact solutions.

When we state and solve recurrences, we often omit floors, ceilings, and boundary conditions, as they usually do not matter.

An important fact

For all natural numbers $k \geq 0$ holds:

$$\sum_{i=0}^{k} 2^i = 1 + 2 + 4 + \cdots + 2^k = 2^{k+1} - 1 = \Theta(2^k).$$

An example for $k = 15$, in hexadecimal notation (16 bits, that is, four hexadecimal places; using the prefix "0x"), with the -1 moved to the other side:

$$(1 + \ldots 2^{15}) + 1 = 0\mathrm{xFFFF} + 1 = 0\mathrm{x}10000 = 2^{16} = 65536.$$

The asymptotic analysis holds in general:

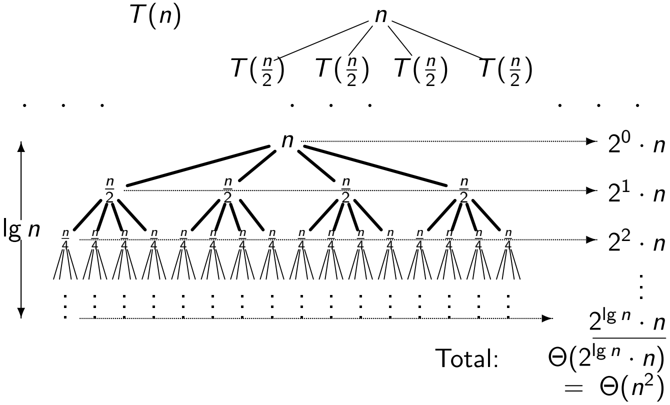
Exponential sums are asymptotically dominated by the last summand.

(This makes them actually useful!)

Recursion trees (quadratic growth)

Unfolding of the recurrence

$$T(n) = n + 4T(n/2):$$



Recursion trees (quasi-linear and linear growth)

What about the MergeSort recurrence

$$T(n) = n + 2T(n/2) \ ?$$

- Again the height of the tree is $\lg n$.
- However now the "workload" of each level is equal to n.
- So all workloads are the same.

So here we get

$$T(n) = \Theta(n \cdot \lg n).$$

And what about the recurrence (as in Min-Max)

$$T(n) = 1 + 2T(n/2) \ ?$$

- Again the height of the tree is $\lg n$.
- The "workload" of the levels are $1, 2, 4, 8, \dots, 2^{\lg n}$.
- Back to the original method, we use the exponential sum.

So here we get

$$T(n) = \Theta(n).$$

Master Theorem (simplified version)

Let $a \geq 1$ and $b > 1$ and $c \geq 0$ be constants.

Let $T(n)$ be defined by the recurrence

$$T(n) = aT(n/b) + \Theta(n^c),$$

where n/b represents either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Then $T(n)$ is bounded asymptotically as follows:

1. If $c < \log_b a$ then $T(n) = \Theta(n^{\log_b a})$.
2. If $c = \log_b a$ then $T(n) = \Theta(n^c \lg n)$.
3. If $c > \log_b a$ then $T(n) = \Theta(n^c)$.

(**General version:** CLRS, Thm 4.1, p94.)

In other words

We start by an equation for $T(n)$ of the form

$$T(n) = b^x \cdot T(\frac{n}{b}) + \Theta(n^c),$$

where the x you have to find: $x = \log_b a$ (so $b^x = b^{\log_b a} = a$).

Then $T(n)$ is bounded asymptotically as follows:

1. If $x > c$ then $T(n) = \Theta(n^x)$.
2. If $x = c$ then $T(n) = \Theta(n^c \lg n)$.
3. If $x < c$ then $T(n) = \Theta(n^c)$.

The meaning of the three parameters in a divide-and-conquer scheme:

- $a = b^x$: the number of subproblems to be solved
- b: how often the subproblems (all of the same size) fit into the full problem
- c: power in the runtime of the local workload, that is, the division- and the combination-computation.

Using the Master Theorem

- The runtime for MIN-MAX satisfies:
  $$T(n) = 2T(n/2) + \Theta(1).$$
  The Master Theorem (case 1) applies:
  $$a = b = 2 \quad \textit{and} \quad c = 0 < 1 = \log_b a,$$
  giving $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

- The runtime for MERGE-SORT satisfies:
  $$T(n) = 2T(n/2) + \Theta(n).$$
  The Master Theorem (case 2) applies:
  $$a = b = 2 \quad \textit{and} \quad c = 1 = \log_b a,$$
  giving $T(n) = \Theta(n^c \lg n) = \Theta(n \lg n)$.

What's happening

For the recurrences

$$\begin{aligned} T_1(n) &= 4T(n/2) + n \\ T_2(n) &= 4T(n/2) + n^2 \\ T_3(n) &= 4T(n/2) + n^3 \end{aligned}$$

the Master Theorem with case $i = 1, 2, 3$ applies:

$$a = 4 \text{ and } b = 2 \text{ (so } \log_b a = 2 \text{), and } c = i \ ,$$

giving

$$T_1(n) = \Theta(n^2) \ , \ T_2(n) = \Theta(n^2 \lg n) \ , \text{ and } T_3(n) = \Theta(n^3) \ .$$

Case 1: applies if the workload-cost (n^c) is negligible compared to the number and size of the subproblems.

Case 2: applies if the workload-cost (n^c) is as costly as the subproblems.

Case 3: applies if the workload-cost (n^c) is the dominating factor.

If we have Case 1 or 3, then in general this might indicate, that the divide-and-conquer approach can be replaced by a simpler approach (as we have seen for the min-max algorithm).

Easy decision between the three cases

Consider (again)

$$T(n) = a \cdot T(\frac{n}{b}) + \Theta(n^c).$$

The main question to start with is always:

Which of the three cases applies?

Apparently you needed to compute $x = \log_b a$ for that. But it is actually easier:

1. If $b^c < a$ then Case 1 applies.
2. If $b^c = a$ then Case 2 applies.
3. If $b^c > a$ then Case 3 applies.

(Try to understand why this holds — it's easy.)

Examples

- $T(n) = 5T(n/2) + \Theta(n^2)$
  In Master Theorem: $a = 5,\ b = 2,\ c = 2$.
  As $\log_b a = \log_2 5 > \log_2 4 = 2 = c$, case 1 applies:
  $T(n) = \Theta(n^{\lg 5})$.
- $T(n) = 27T(n/3) + \Theta(n^3)$
  In Master Theorem: $a = 27,\ b = 3,\ c = 3$.
  As $\log_b a = \log_3 27 = 3 = c$, case 2 applies:
  $T(n) = \Theta(n^3 \lg n)$.
- $T(n) = 5T(n/2) + \Theta(n^3)$
  In Master Theorem: $a = 5,\ b = 2,\ c = 3$.
  As $\log_b a = \log_2 5 < \log_2 8 = 3 = c$, case 3 applies:
  $T(n) = \Theta(n^3)$.

More examples

1. $T(n) = 3T(n/3) + 1$:

More examples

1. $T(n) = 3T(n/3) + 1: \; T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1:$

More examples

1. $T(n) = 3T(n/3) + 1: \; T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \; T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1:$

More examples

1. $T(n) = 3T(n/3) + 1 : \; T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1 : \; T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1 : \; T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1 :$

More examples

1. $T(n) = 3T(n/3) + 1$: $T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1$: $T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1$: $T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1$: $T(n) = \Theta(n^{\log_3 4})$
5. $T(n) = 3T(n/3) + n$:

More examples

1. $T(n) = 3T(n/3) + 1: \ T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \ T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 4})$
5. $T(n) = 3T(n/3) + n: \ T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n:$

More examples

1. $T(n) = 3T(n/3) + 1: \ T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \ T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 4})$

5. $T(n) = 3T(n/3) + n: \ T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n: \ T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2}:$

More examples

1. $T(n) = 3T(n/3) + 1: \; T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \; T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1: \; T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1: \; T(n) = \Theta(n^{\log_3 4})$
5. $T(n) = 3T(n/3) + n: \; T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n: \; T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2}: \; T(n) = \Theta(n^{\log_3 2} \log n)$
8. $T(n) = 4T(n/3) + n^{\log_3 4}:$

More examples

1. $T(n) = 3T(n/3) + 1: \ T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \ T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 4})$

5. $T(n) = 3T(n/3) + n: \ T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n: \ T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2}: \ T(n) = \Theta(n^{\log_3 2} \log n)$
8. $T(n) = 4T(n/3) + n^{\log_3 4}: \ T(n) = \Theta(n^{\log_3 4} \log n)$

9. $T(n) = 3T(n/3) + n^{1.5}:$

More examples

1. $T(n) = 3T(n/3) + 1: \ T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \ T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1: \ T(n) = \Theta(n^{\log_3 4})$

5. $T(n) = 3T(n/3) + n: \ T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n: \ T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2}: \ T(n) = \Theta(n^{\log_3 2} \log n)$
8. $T(n) = 4T(n/3) + n^{\log_3 4}: \ T(n) = \Theta(n^{\log_3 4} \log n)$

9. $T(n) = 3T(n/3) + n^{1.5}: \ T(n) = \Theta(n^{1.5})$
10. $T(n) = aT(n/a) + n^{1.5}:$

More examples

1. $T(n) = 3T(n/3) + 1: \; T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1: \; T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1: \; T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1: \; T(n) = \Theta(n^{\log_3 4})$
5. $T(n) = 3T(n/3) + n: \; T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n: \; T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2}: \; T(n) = \Theta(n^{\log_3 2} \log n)$
8. $T(n) = 4T(n/3) + n^{\log_3 4}: \; T(n) = \Theta(n^{\log_3 4} \log n)$
9. $T(n) = 3T(n/3) + n^{1.5}: \; T(n) = \Theta(n^{1.5})$
10. $T(n) = aT(n/a) + n^{1.5}: \; T(n) = \Theta(n^{1.5})$
11. $T(n) = 2T(n/3) + n:$

More examples

1. $T(n) = 3T(n/3) + 1 : \ T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1 : \ T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1 : \ T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1 : \ T(n) = \Theta(n^{\log_3 4})$
5. $T(n) = 3T(n/3) + n : \ T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n : \ T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2} : \ T(n) = \Theta(n^{\log_3 2} \log n)$
8. $T(n) = 4T(n/3) + n^{\log_3 4} : \ T(n) = \Theta(n^{\log_3 4} \log n)$
9. $T(n) = 3T(n/3) + n^{1.5} : \ T(n) = \Theta(n^{1.5})$
10. $T(n) = aT(n/a) + n^{1.5} : \ T(n) = \Theta(n^{1.5})$
11. $T(n) = 2T(n/3) + n : \ T(n) = \Theta(n)$
12. $T(n) = 4T(n/3) + n^2 :$

More examples

1. $T(n) = 3T(n/3) + 1 : \; T(n) = \Theta(n)$
2. $T(n) = aT(n/a) + 1 : \; T(n) = \Theta(n)$
3. $T(n) = 2T(n/3) + 1 : \; T(n) = \Theta(n^{\log_3 2})$
4. $T(n) = 4T(n/3) + 1 : \; T(n) = \Theta(n^{\log_3 4})$
5. $T(n) = 3T(n/3) + n : \; T(n) = \Theta(n \log n)$
6. $T(n) = aT(n/a) + n : \; T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/3) + n^{\log_3 2} : \; T(n) = \Theta(n^{\log_3 2} \log n)$
8. $T(n) = 4T(n/3) + n^{\log_3 4} : \; T(n) = \Theta(n^{\log_3 4} \log n)$
9. $T(n) = 3T(n/3) + n^{1.5} : \; T(n) = \Theta(n^{1.5})$
10. $T(n) = aT(n/a) + n^{1.5} : \; T(n) = \Theta(n^{1.5})$
11. $T(n) = 2T(n/3) + n : \; T(n) = \Theta(n)$
12. $T(n) = 4T(n/3) + n^2 : \; T(n) = \Theta(n^2)$

Final examples

$$T(n) = 10\,T(n/3) + \boxed{n^2}$$

Final examples

$$T(n) = 10\,T(n/3) + \boxed{n^2}$$

$$3^2 < 10$$

Final examples

$$T(n) = 10\,T(n/3) + \boxed{n^2}$$

$$3^2 < 10$$

Thus Case I: $T(n) = \Theta(n^{\log_3 10})$ (note $\log_3(10) > 2$).

Final examples

$$T(n) = 10\,T(n/3) + \boxed{n^2}$$

$$3^2 < 10$$

Thus Case I: $T(n) = \Theta(n^{\log_3 10})$ (note $\log_3(10) > 2$).

The other cases:

- $T(n) = 9\,T(n/3) + n^2$: Case II, thus $T(n) = \Theta(n^2 \cdot \lg n)$.
- $T(n) = 8\,T(n/3) + n^2$: Case III, thus $T(n) = n^2$.

Growth rates as response rates

The fundamental setting:

- input size $n > 0$
- some abstract "runtime" $f(n) > 0$.

Understanding the "growth rate" of $f(n)$ means:

understanding how a change of n effects $f(n)$ —
increasing n in a certain way, how much is $f(n)$ increased?

Thus we get the "dictionary":

slow growth a big change of n causes only a small change of $f(n)$

large growth a small change of n causes a large change of $f(n)$

intermediate growth the change of $f(n)$ is kind of proportional to the change of n.

Ways of change

The two most basic forms of quantitative change are:

additive: the quantity (n or $f(n)$) is changed by adding a constant:

$$n \rightsquigarrow n + 1$$

$$f(n) \rightsquigarrow f(n) + c$$

multiplicative: the quantity (n or $f(n)$) is changed by multiplying a constant:

$$n \rightsquigarrow 2n$$

$$f(n) \rightsquigarrow c \cdot f(n)$$

For us "additive" means "small", and "multiplicative" means "big".

Linear growth

Considering all growth rates, **linear growth** is the middle (while for pure algorithms it is the bottom):

- a small change of n yields a small change of $f(n)$:

$$n \rightsquigarrow n+1 \Longrightarrow f(n) \rightsquigarrow c + f(n)$$

- a big change of n yields the *same* big change of $f(n)$:

$$n \rightsquigarrow 2 \cdot n \Longrightarrow f(n) \rightsquigarrow 2 \cdot f(n).$$

For the proof let $f(n) = \alpha \cdot n$:

$$\begin{aligned} f(n+1) &= \alpha \cdot (n+1) = \alpha \cdot n + \alpha = f(n) + \alpha \\ f(2n) &= \alpha \cdot (2n) = 2 \cdot (\alpha \cdot n) = 2f(n). \end{aligned}$$

Polynomial growth

Polynomial growth means (roughly) a function $f(n) = n^{\alpha}$ for some (constant) $\alpha > 0$:

- sublinear $\alpha < 1$
- superlinear $\alpha > 1$.

Characteristic here is that a big change of n yields a proportional change of $f(n)$:

$$n \rightsquigarrow 2n \Longrightarrow f(n) \rightsquigarrow 2^{\alpha} \cdot f(n).$$

The proof is simple:

$$f(2n) = (2n)^{\alpha} = 2^{\alpha} \cdot n^{\alpha} = 2^{\alpha} \cdot f(n).$$

(Note that 2^{α} is constant.)

Logarithmic growth

Logarithmic growth means (roughly) a function $f(n) = \log_b(n)$ for some (constant) $b > 1$:

Characteristic here is that a big change of n yields a small change of $f(n)$:

$$n \rightsquigarrow 2n \Longrightarrow f(n) \rightsquigarrow \log_b(2) + f(n).$$

The proof is again simple:

$$f(2n) = \log_b(2n) = \log_b(2) + \log_b(n) = \log_b(2) + f(n).$$

(Note that $\log_b(2)$ is constant.)

Exponential growth

Exponential growth means (roughly) a function $f(n) = b^n$ for some (constant) $b > 1$:

Characteristic here is that a small change of n yields a big change of $f(n)$:

$$n \rightsquigarrow n + 1 \Longrightarrow f(n) \rightsquigarrow b \cdot f(n).$$

The proof is, as always, simple:

$$f(n + 1) = b^{n+1} = b^n \cdot b^1 = b \cdot f(n).$$

(Note that b is constant.)