RESOURCES FOR LEARNING DATA STRUCTURES AND ALGORITHMS

Data structures and algorithms are essential for any programmer. An engineer with a deep understanding of algorithms and data structures will be able to make informed design choices, and write programs that are more performant and easier to change.

Why Data Structures and Algorithms?

Before starting to learn anything you must know the purpose of it, so before you get started with any new data structure or algorithms try to understand why is this needed, what is the problem and how this new data structure will solve it.

Understand the Learning Sequence

Data structures are built using a previous data structure and algorithms use the understanding of some previous algorithms as facts, so it very important to learn things in a particular sequence. You cannot just start with trees without understanding how an array works.

Time and Space Complexities are Very Very Important

Whenever you are done with studying a new data structure it is very important to know the time complexities of every operation which that particular data structure performs. As every algorithm can be designed with a vast majority of data structures it is important to decide which will the best in some particular case.

Competitive Programming

Competitive Programming can help you understand the use of different data structures with lots of practical problems that we commonly do not encounter even in large-sized projects. Websites like codechef, codeforces, and spoj are helping thousand of programmers to get started with understanding data structures in depth.

Practice, Practice and Practice

Ultimately there is no easy way to success if you want to master data structures and algorithms you need lots and lots of practice. Do not just read through internet articles explaining data structure, understanding them is very important. Code the basic structure of all the data structures. Never skip things until you understand them perfectly as

they will surely haunt in the later stages of your carrier.

BEST BOOKS TO LEARN DSA :

Here is the list of some popular books that you can refer to learn DSA easily.

- Introduction to Algorithms by Thomas H. Cormen
- Data Structures with C by Seymour Lipschutz
- Data Structures and Algorithms Made Easy by Narasimha Karumanchi **(BEST FOR LEARNING ALGORITHMS)**
- Data Structures Through C in Depth by S.K.Srivastava/Deepali Srivastava.

Resources for each topic in DSA, and resources will be mostly the videos present on Youtube, and once you are done with the videos start practicing that topic, and do a good amount of questions on that topic, so that you get to know about that topic in-depth and you can completely understand it.

1. **Arrays** — The very first and basic topic of DSA.
   * *Basics of the array* — Array lectures by Neso Academy (*This is only for the people who want to learn from scratch*)

* *Sliding Window* — Playlist by Aditya Verma
* *Sorting* — Playlist by mycodeschool

2. **Greedy** — Videos by Abdul Bari Sir (from 3 to 3.5)
3. **Hashing** — Video by Abdul Bari Sir
4. **Stack** — Stack Playlist by Aditya Verma
5. **Queue** — To understand the basics of the queue go to Jenny's Lecture
6. **Recursion** — Recursion Playlist by Aditya Verma
7. **Linked List** — Playlist by Vivekanand Khyade Sir
8. **Binary Trees** — Playlist by Vivekanand Khyade Sir, Playlist by Kashisk Mehndiratta
9. **Binary Search Trees** — Video by mycodeschool
10. **Strings** —
    * *Rabin Karp Algo* —Video by Abdul Bari Sir, TECH DOSE
    * *KMP Algo* — Video by Abdul Bari Sir, TECH DOSE
11. **Backtracking** — Videos by Abdul Bari Sir
12. **Binary Search** — Playlist by Aditya Verma
13. **Dynamic Programming** — Playlist by Aditya Verma (Best DP lectures on Youtube)

14. **Heaps** — Playlist by Aditya Verma (If you want to learn basics then first refer Abdul Bari Sir videos)

15. **Graphs** — Playlist by TECH DOSE (If you want to learn basics then first refer Abdul Bari Sir videos)

16. **Trees** — Video by TECH DOSE ...

Online Platforms :

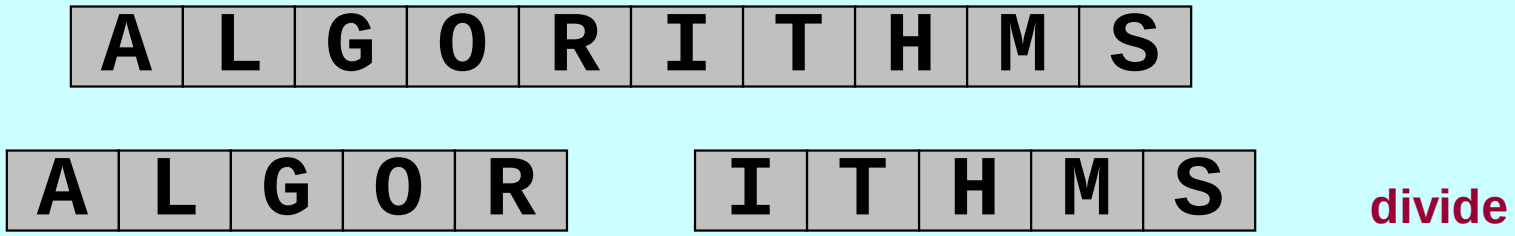
- CodeChef - CodeChef competitive programming site
- CodeSignal - (formerly CodeFights)Fun gaming approach to Coding contests and Interview practices.
- Codeforces - Great site for preparing for programming contests
- GeeksforGeeks - Must do coding questions for product based companies
- Hackerearth - Code Monk to start with programming - programming fundamentals
- Hackerrank - Interview preparation kit
- InterviewBit - Best platform to get prepared for Data Structures based interviews
- InterviewCake - An interactive interview prep site for DSA and some System Design with free 3 week access through Github student pack

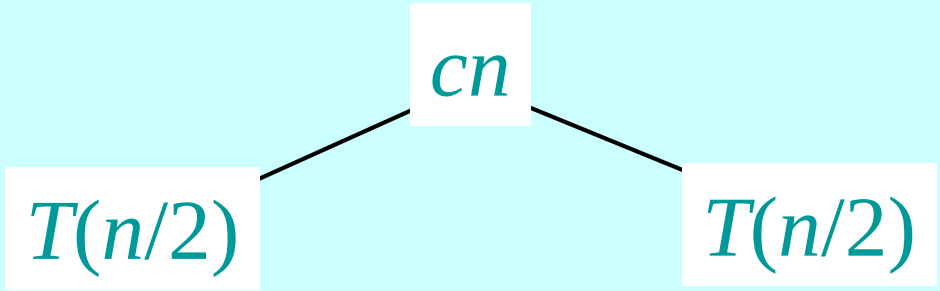
- AlgoDaily - Daily interview questions sent by mail, as well as a full course and online IDE as well as visualizations and tutorials to solve the problems
- LeetCode - Platform to prepare for technical interviews with real interview questions
- Sphere Online Judge - Great head start for learning Data Structures
- UVa Online Judge - The site to submit Competitive Programming 3 data structures problems
- Codewars - Interesting ranking system with beautiful UI for competitive programming and interview prep.
- CodinGame - Competitive programming with game like challenges
- CS50 on HarvardX - One of the best computer science courses available online ($ for certification)
- Codility - Develop your coding skills with lessons to take part in challenges
- Zen of Programming - A frequently updated blog great for beginners and simplified references.

Solving recurrences

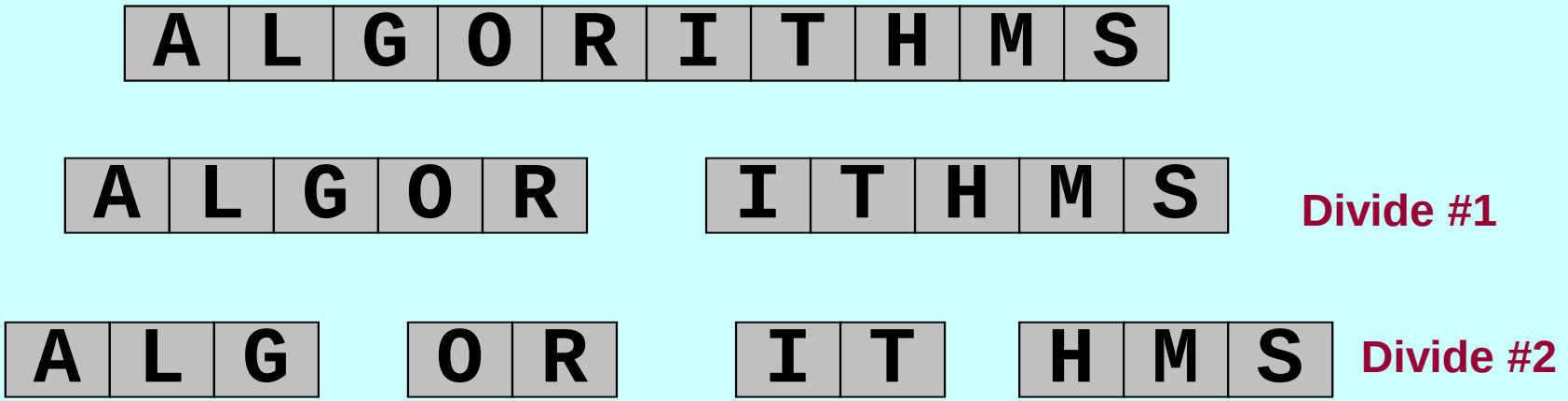
- The analysis of divide and conquer algorithms require us to solve a recurrence.
- Recurrences are a major tool for analysis of algorithms

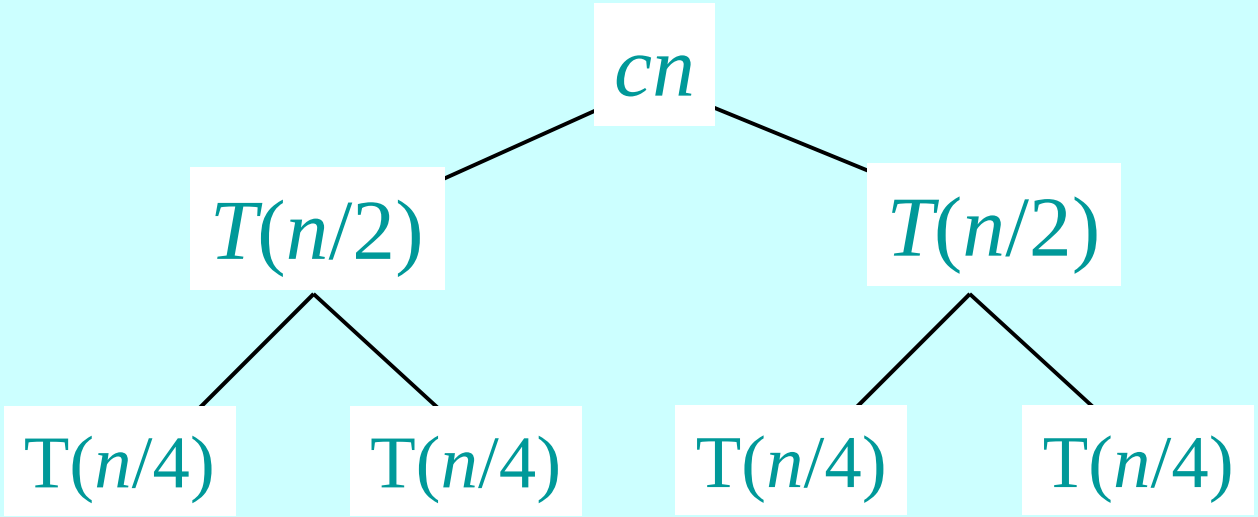
MergeSort





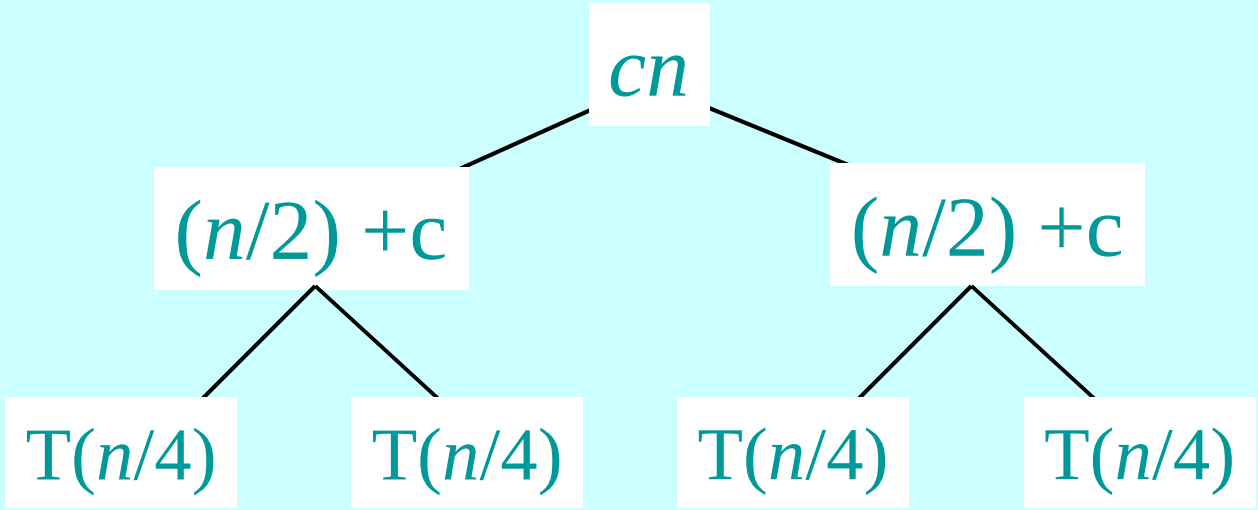
MergeSort


A L G O R I T H M S
A L G O R
I T H M S
Divide #1
A L G
O R
I T
H M S
Divide #2


cn
T(n/2)
T(n/2)
T(n/4)
T(n/4)
T(n/4)
T(n/4)

MergeSort

Solve $T(n) = T(n/2) + T(n/2) + cn$



Recurrence

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

Integer Multiplication

- Let X = |A|B| and Y = |C|D| where A,B,C and D are n/2 bit integers
- **Simple Method**: $XY = (2^{n/2}A+B)(2^{n/2}C+D)$
- **Running Time Recurrence**

$$T(n) < 4T(n/2) + 100n$$

How do we solve it?

Substitution method

The most general method:

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.

Example: $T(n) = 4T(n/2) + 100n$

- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \le ck^3$ for $k < n$.
- Prove $T(n) \le cn^3$ by induction.

Example of substitution

$$
\begin{aligned}
T(n) &= 4T(n/2) + 100n \\
&\le 4c(n/2)^3 + 100n \\
&= (c/2)n^3 + 100n \\
&= cn^3 - ((c/2)n^3 - 100n) \quad \leftarrow \textit{desired} - \textit{residual} \\
&\le cn^3 \quad \leftarrow \textit{desired}
\end{aligned}
$$

whenever $(c/2)n^3 - 100n \ge 0$, for example, if $c \ge 200$ and $n \ge 1$.

residual

Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion tree method is good for generating guesses for the substitution method.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- The recursion-tree method promotes intuition, however.

Example of recursion tree

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

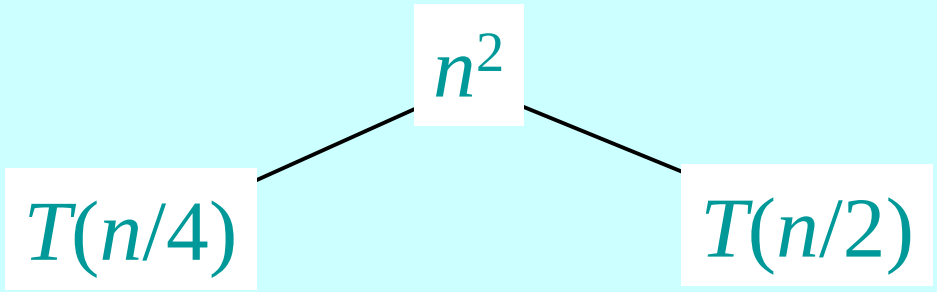
Example of recursion tree

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

$T(n)$

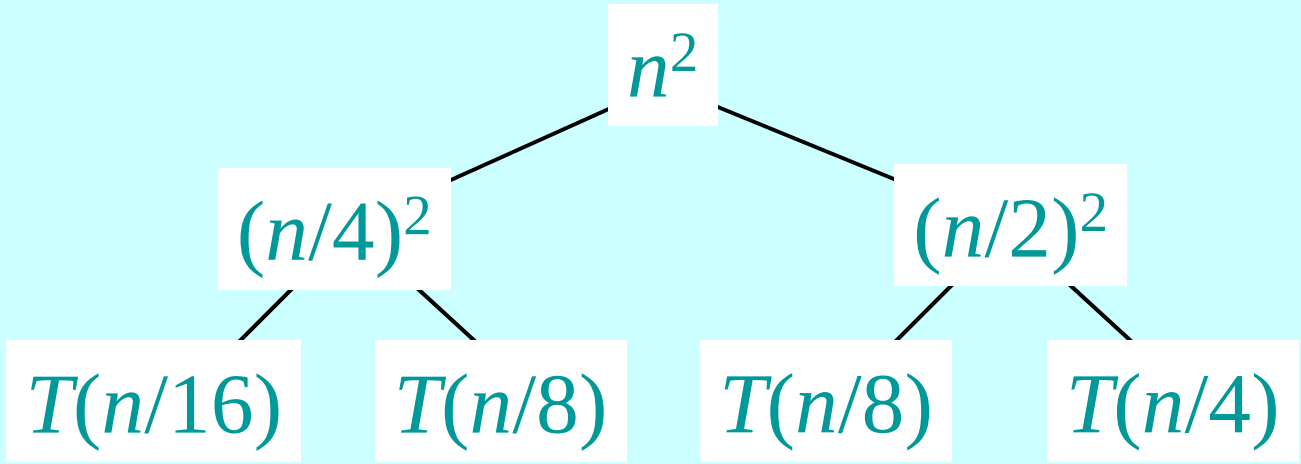
Example of recursion tree

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



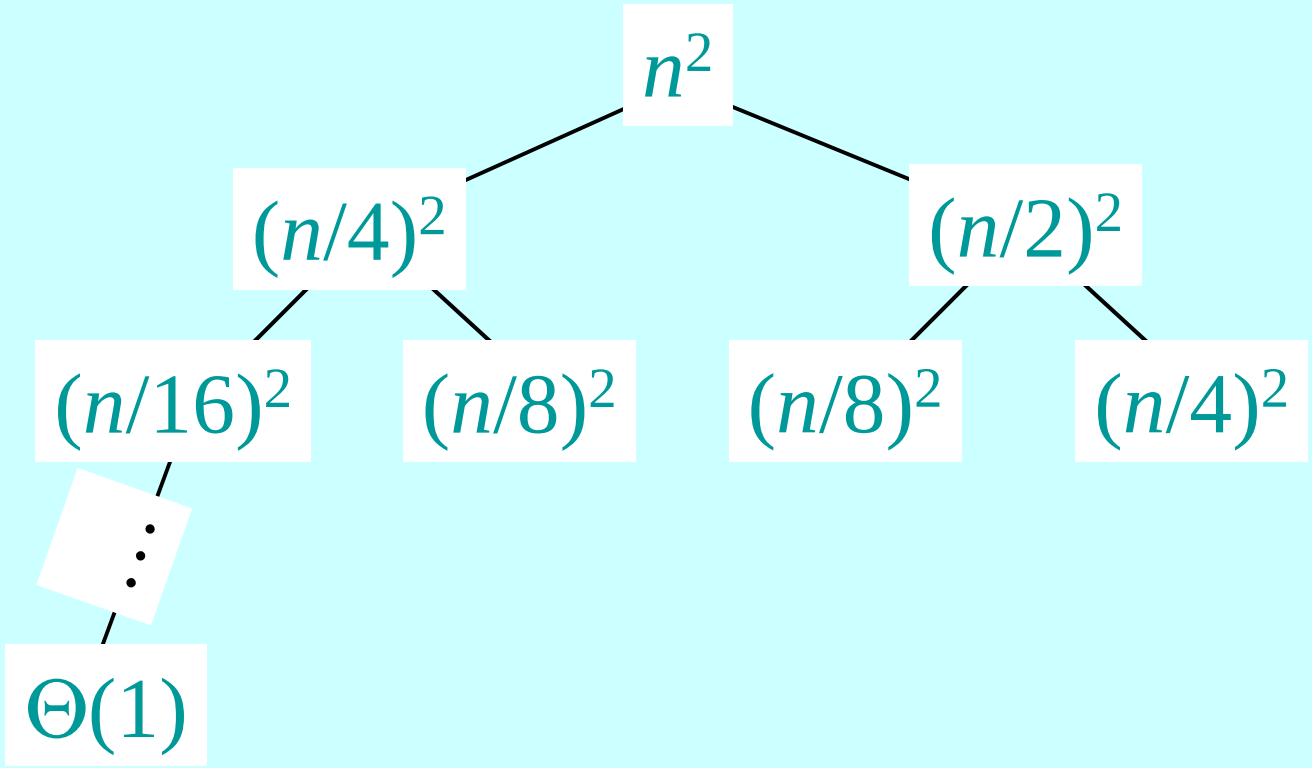
Example of recursion tree

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



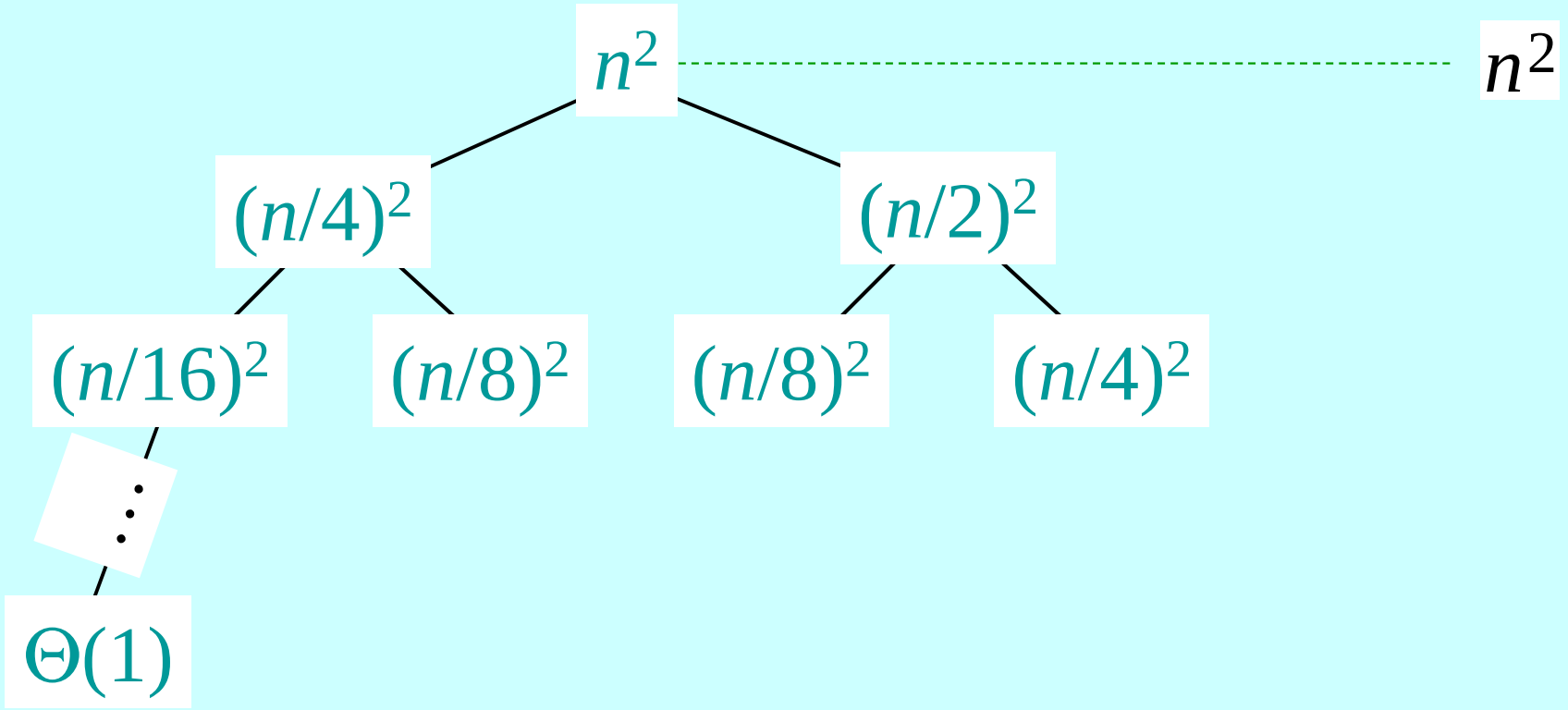
Example of recursion tree

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



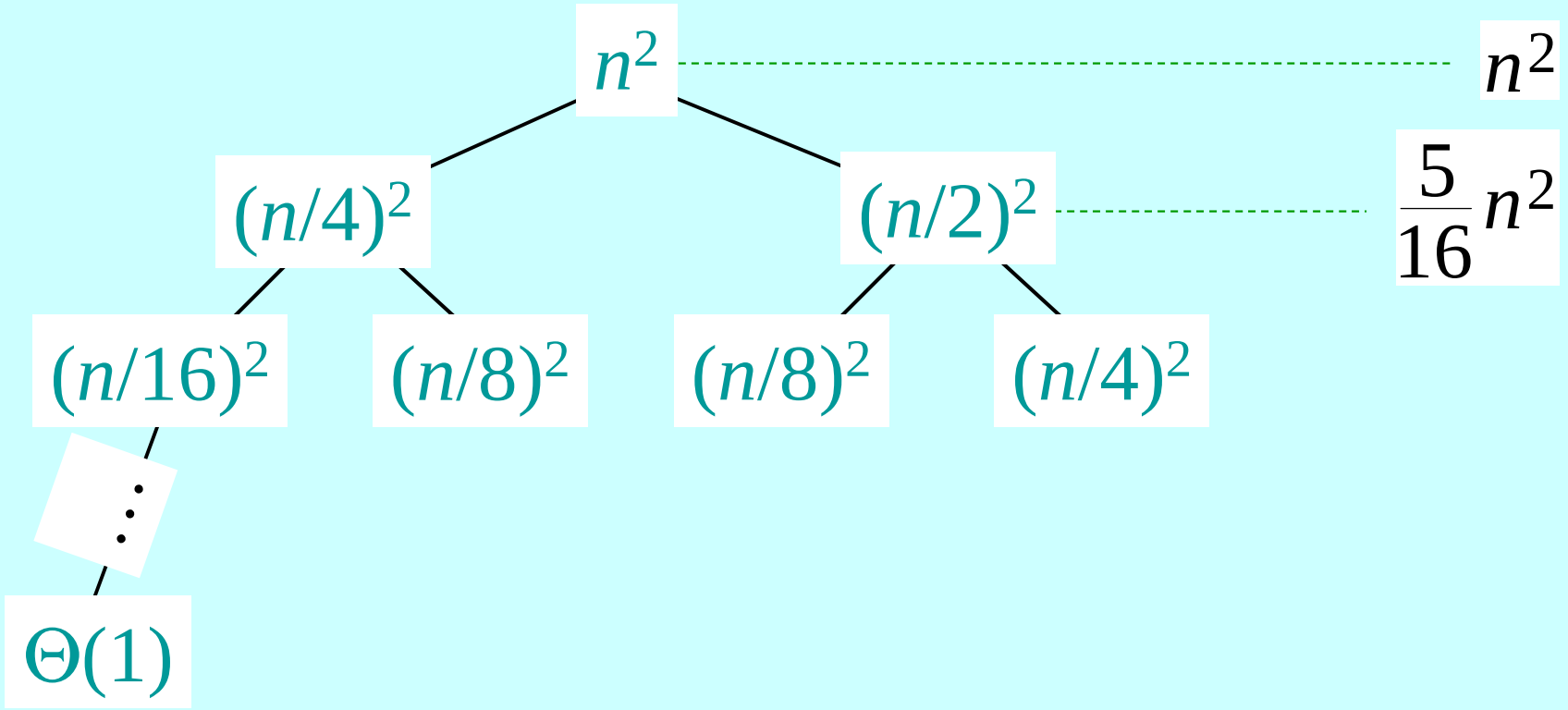
Example of recursion tree

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



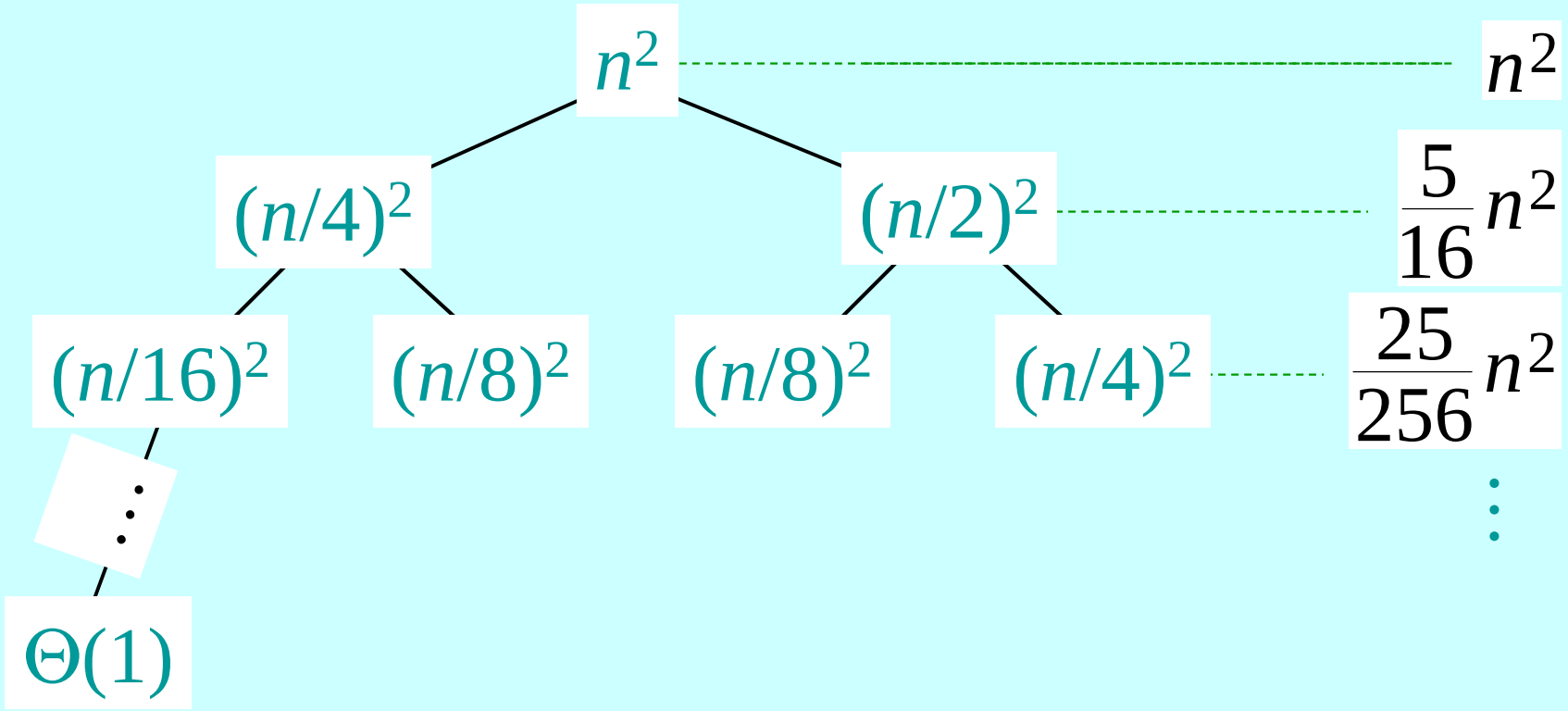
Example of recursion tree

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



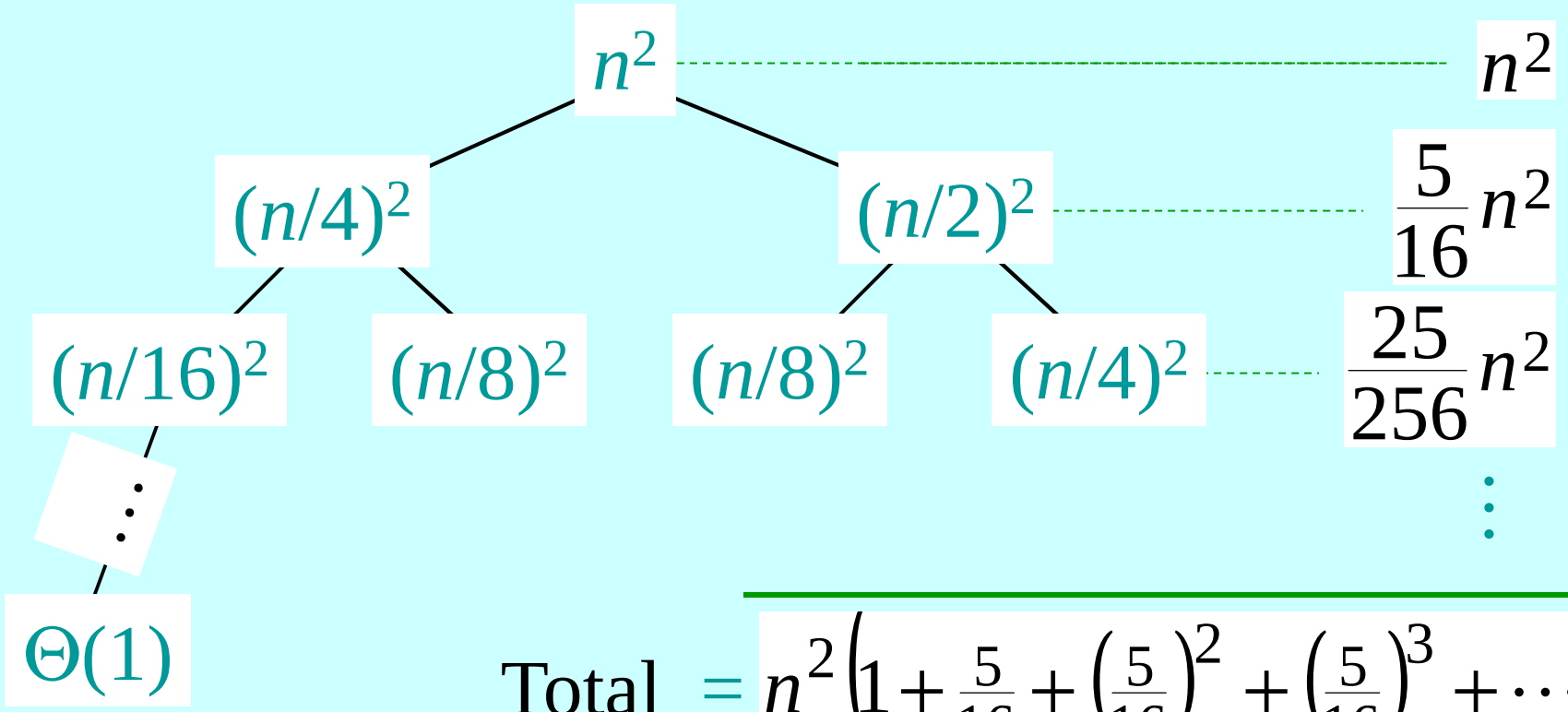
Example of recursion tree

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



Example of recursion tree

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



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$

$$= \Theta(n^2) \quad \textit{geometric series}$$

Appendix: geometric series

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \cdots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

The master method

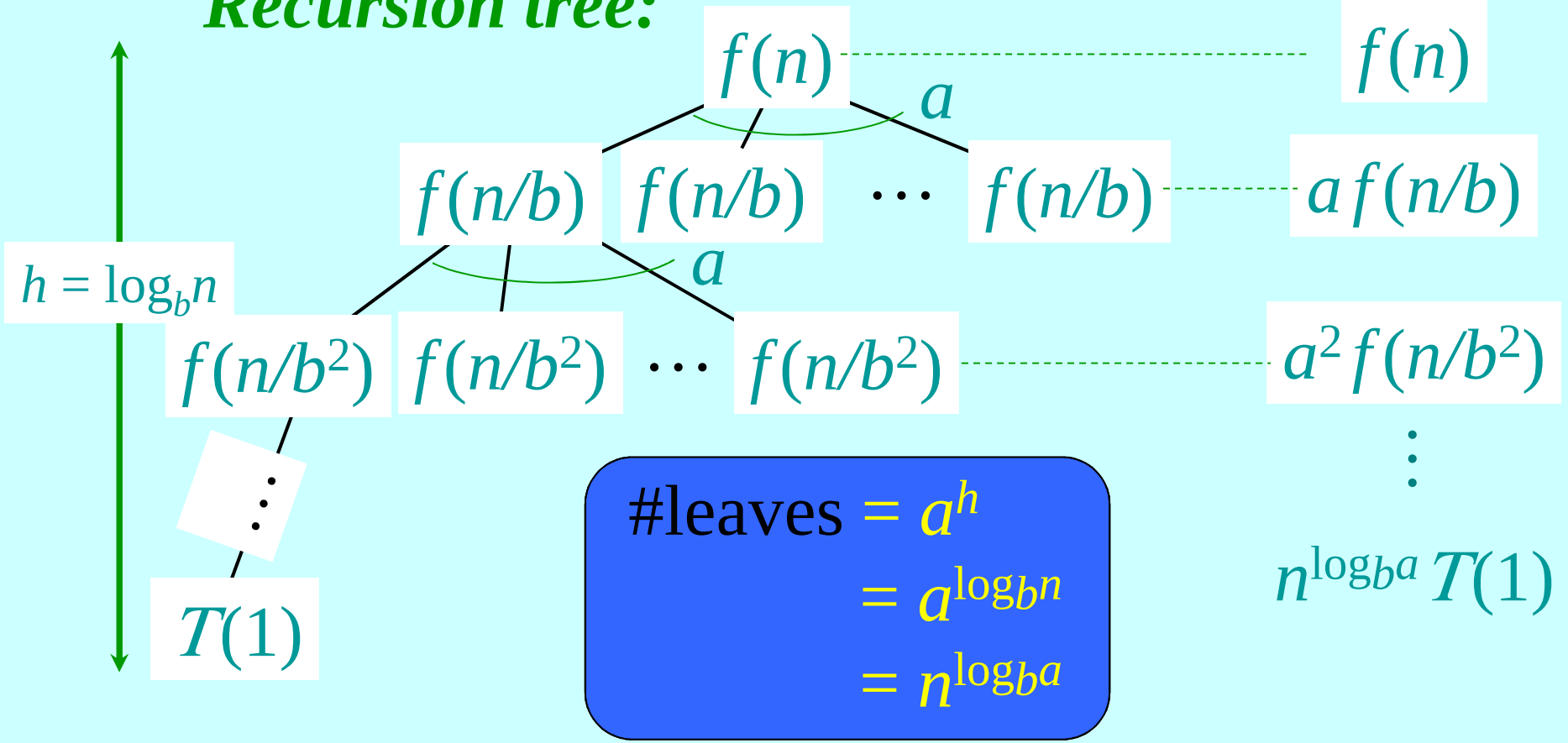
The master method applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n)\ ,$$

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

Idea of master theorem

Recursion tree:



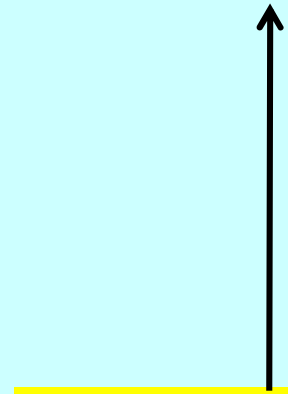
#leaves $= a^h$
$= a^{\log_b n}$
$= n^{\log_b a}$

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

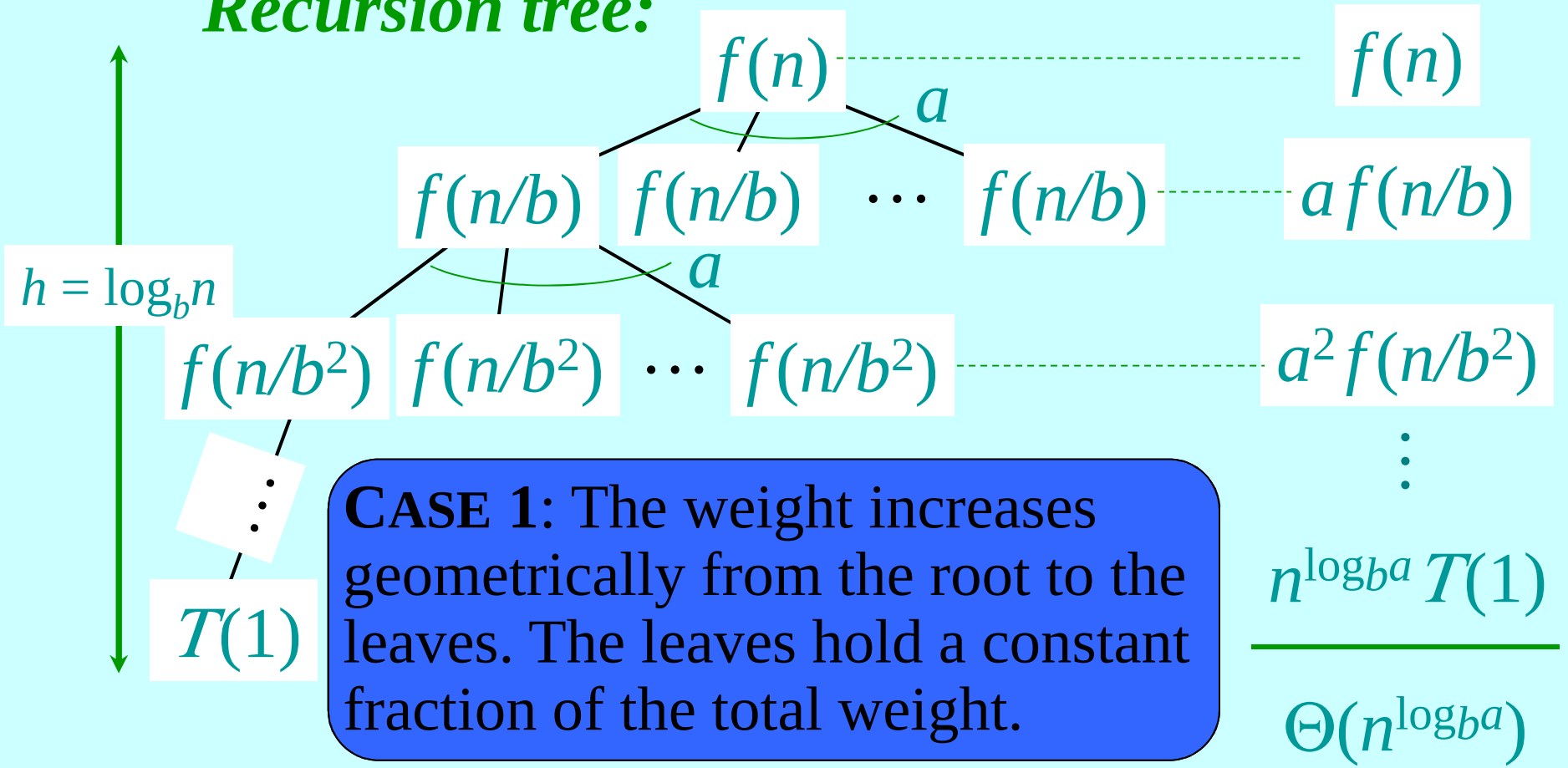
   Solution: $T(n) = \Theta(n^{\log_b a})$.



leaves in recursion tree

Idea of master theorem

Recursion tree:



CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Three common cases

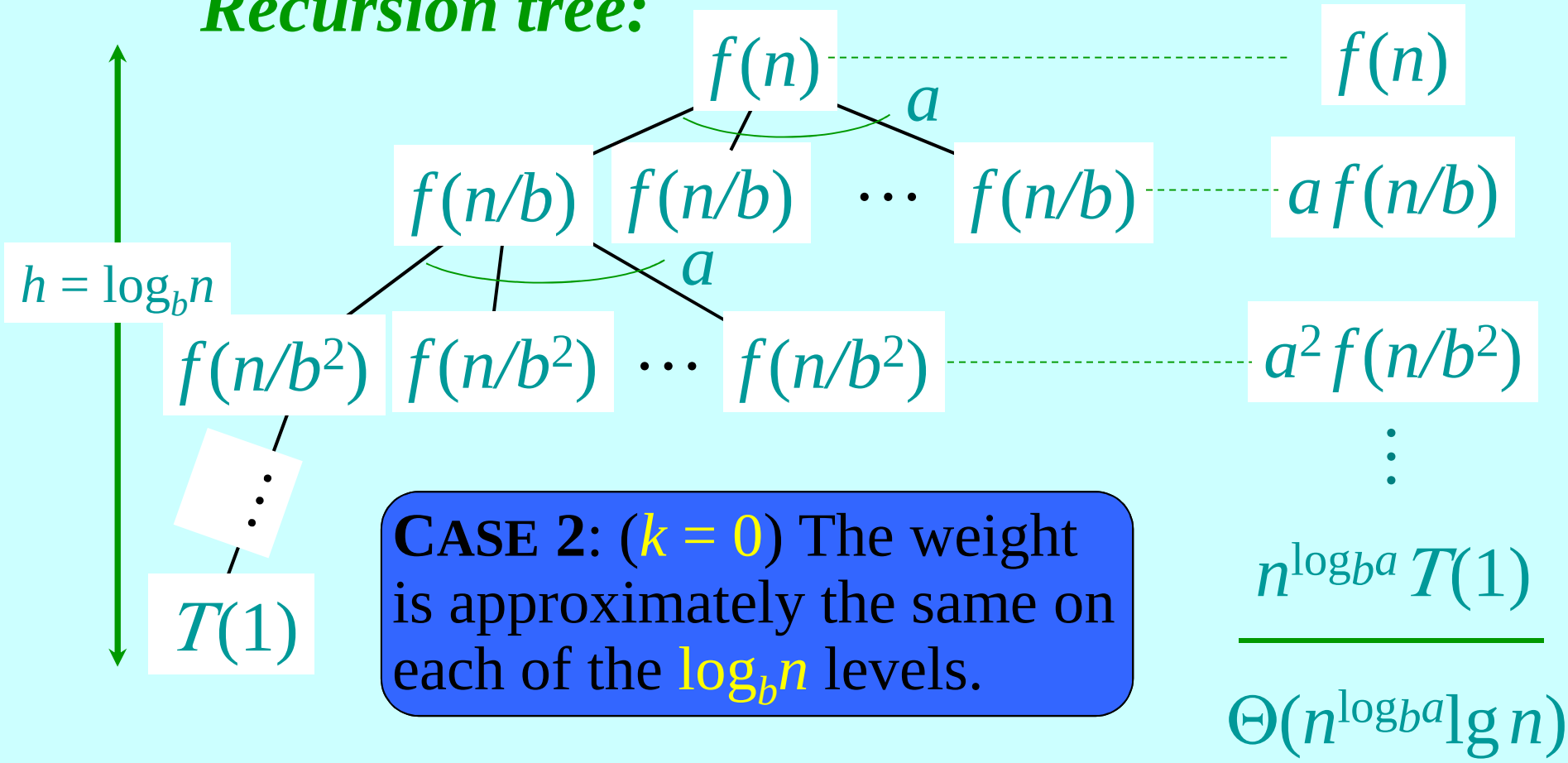
Compare $f(n)$ with $n^{\log_b a}$:

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Idea of master theorem

Recursion tree:



CASE 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

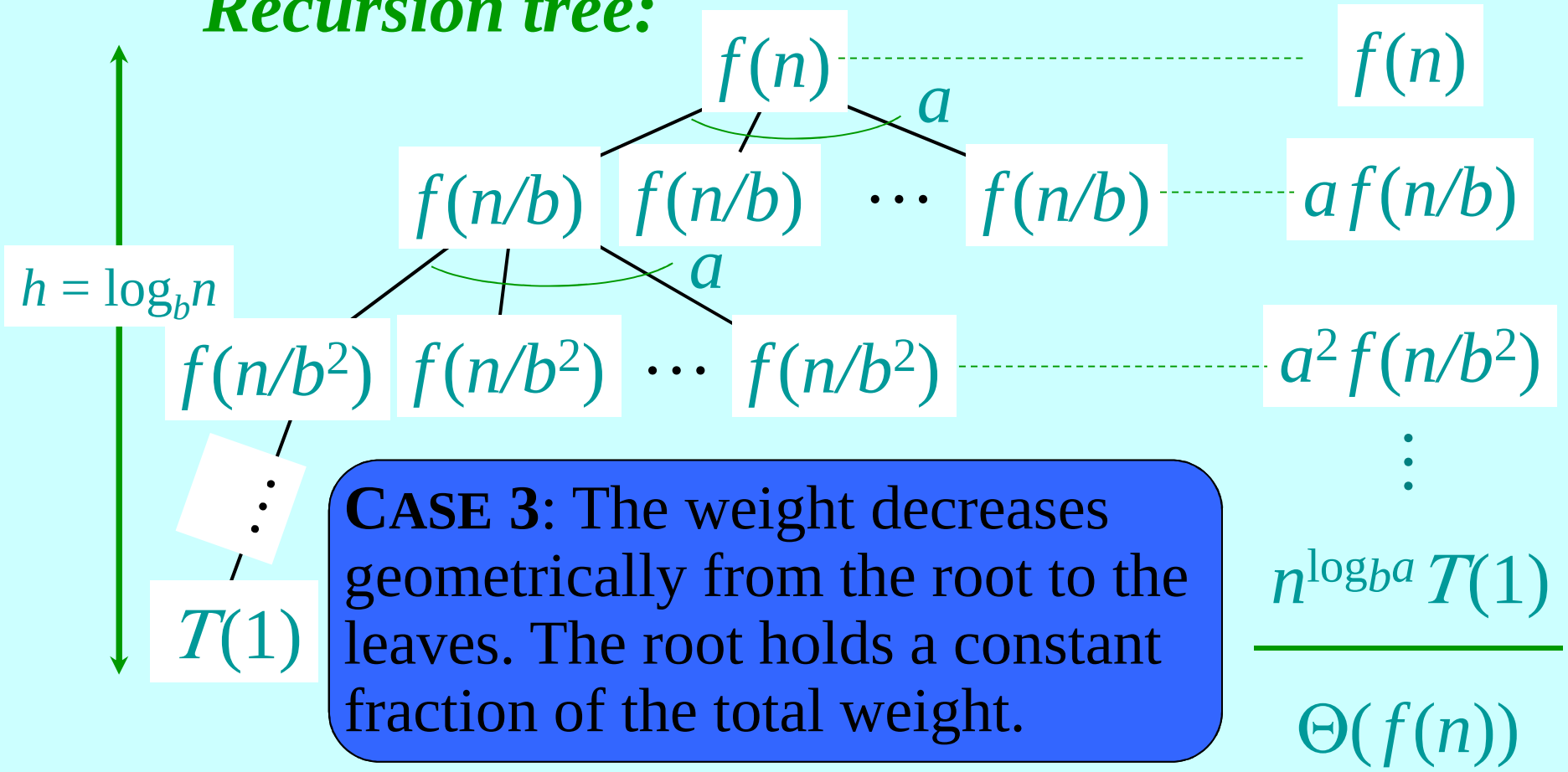
3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$.

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

Idea of master theorem

Recursion tree:



CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Examples

Ex. $T(n) = 4T(n/2) + n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n$.

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

$\therefore T(n) = \Theta(n^2)$.

Ex. $T(n) = 4T(n/2) + n^2$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^2$.

CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$\therefore T(n) = \Theta(n^2 \lg n)$.

Examples

Ex. $T(n) = 4T(n/2) + n^3$

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^3$.

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(cn/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3)$.

Ex. $T(n) = 4T(n/2) + n^2/\lg n$

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^2/\lg n$.

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\lg n)$.

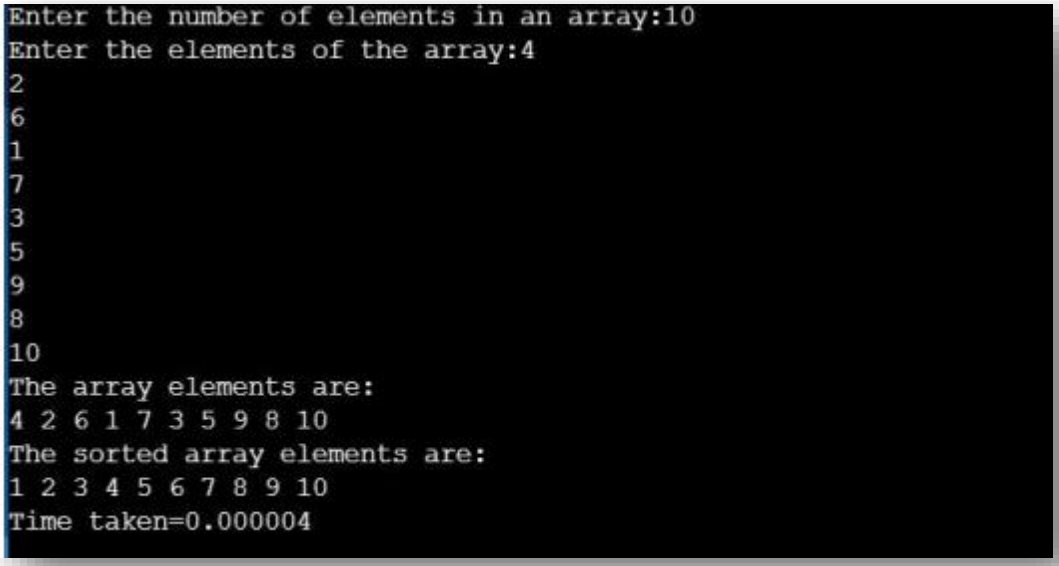
PRACTICAL - 3
Implementation and Time Analysis of Max-Heap Sort Algorithm

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;
void swap(int *a,int *b)
{
 int temp=*a;
 *a=*b;
 *b=temp;
}
void maxHeapify(int a[],int n,int i)
{
 int largest=i;
 int l=(2*i)+1;
 int r=(2*i)+2;
 if(l<n && a[l]>a[largest])
 {
 largest=l;
 }
 if(r<n && a[r]>a[largest])
 {
 largest=r;
 }
 if(largest!=i)//checking if the value of largest is updated or not
 {
 swap(&a[i],&a[largest]);
 maxHeapify(a,n,largest);
 }
}
void heapSort(int a[],int n)
{
 int i;
 for(i=n/2-1;i>=0;i--)
 {
 maxHeapify(a,n,i);
 for(i=n-1;i>0;i--)
 {
 swap(&a[0],&a[i]);
 maxHeapify(a,i,0);
 }
}
int main()
{
 cout<<"Enter the number of elements in an array:";
```

```
 int n,i;
 clock_t start,end;
 cin>>n;
 cout<<"Enter the elements of the array:";
 int a[n];
 for(i=0;i<n;i++)
 {
 cin>>a[i];
 }
 cout<<"The array elements are:"<<endl;
 for(i=0;i<n;i++)
 {
 cout<<a[i]<<" ";

 }
 cout<<endl;
 cout<<"The sorted array elements are:"<<endl;
 start=clock();
 heapSort(a,n);
 end=clock();
 for(i=0;i<n;i++)
 {
 cout<<a[i]<<" ";
 }
 cout<<endl;
 double tc=difftime(end,start)/CLOCKS_PER_SEC;
 printf("Time taken=%lf",tc);
 return 0;
}
```

Output:


Enter the number of elements in an array:10
Enter the elements of the array:4
2
6
1
7
3
5
9
8
10
The array elements are:
4 2 6 1 7 3 5 9 8 10
The sorted array elements are:
1 2 3 4 5 6 7 8 9 10
Time taken=0.000004