|  |
| --- |
| Swayam |
| **Introduction to Algorithms and Analysis** |
| Prof. Sourav Mukhopadhyay | IIT Kharagpur |

|  |
| --- |
| VARUN KUMAR  02-11-2025 |

Table of Contents

[Week – 01: Sorting Problem, Time Complexity, and Asymptotic Analysis 3](#_Toc201000410)

[Lecture 01: Insertion Sort 3](#_Toc201000411)

[Lecture 02: Analysis of Insertion Sort 4](#_Toc201000412)

[Lecture 03: Asymptotic Notation 5](#_Toc201000413)

[Lecture 04: Recurrence for Merge Sort 6](#_Toc201000414)

[Lecture 05: Substitution Method 8](#_Toc201000415)

[Weekley Test 9](#_Toc201000416)

[2023 9](#_Toc201000417)

[Week – 02: Solving Recurrence, Divide and Conquer 10](#_Toc201000418)

[Lecture 06: The Master Method 10](#_Toc201000419)

[Week – 03: Quick Sort and Heap Sort, Decision Tree 15](#_Toc201000420)

[Week – 04: Linear Time Sorting, Order Statistic 16](#_Toc201000421)

[Week – 05: Hash Function, Binary Search Tree (BST) Sort 17](#_Toc201000422)

[Week – 06: Randomly Build BST, Red Black Tree, Augmentation of Data Structure 18](#_Toc201000423)

[Week – 07: Van Emde Boas, Amortized Analysis, Computational Geometry 19](#_Toc201000424)

[Week – 08: Dynamic Programming, Graph, Prim’s Algorithm 20](#_Toc201000425)

[Week – 09: BFS & DFS, Shortest Path Problem, Dijktra, Bellman-Ford 21](#_Toc201000426)

[Week – 10: All Pair Shortest Path, Floyd-Warshall, Jhonson Algorithm 22](#_Toc201000427)

[Week – 11: More Amortized Analysis, Disjoint Set Data Structure 23](#_Toc201000428)

[Week – 12: Network Flow, Computational Complexity 24](#_Toc201000429)

[Appendix – 01: Test of 2025 25](#_Toc201000430)

[Week – 01 25](#_Toc201000431)

[Appendix – 02: Important Links 26](#_Toc201000432)

[Appendix – 03: Chat GPT and Deep Seek 27](#_Toc201000433)

[Insertion Sort 27](#_Toc201000434)

[Whimsical Diagrams 31](#_Toc201000435)

[Sorting Technique 31](#_Toc201000436)

[Divide and Conquer Algorithm 32](#_Toc201000437)

[Greedy Algorithm 33](#_Toc201000438)

[Dynamic Programming 34](#_Toc201000439)

[Master’s Theorem 35](#_Toc201000440)

[Appendix – 04: Python Setup Guide 36](#_Toc201000441)

[Appendix – 05: Step-by-Step Guide of Various Algorithm with Python Code 39](#_Toc201000442)

[01 – Implementation of Dijkstra’s Algorithm 39](#_Toc201000443)

[02 - Implementation of Bellman-Ford Algorithm 40](#_Toc201000444)

[03 - Implementation of Kahn’s Algorithm 41](#_Toc201000445)

[04 - Implementation of Dinic’s Algorithm 42](#_Toc201000446)

[05 - Implementation of Ford-Fulkerson Algorithm 43](#_Toc201000447)

[06 - Implementation of Prim’s Algorithm 44](#_Toc201000448)

[07 - Implementation of Kruskal’s Algorithm 45](#_Toc201000449)

[08 - Implementation of Basic Operation Associated with B+ Tree 46](#_Toc201000450)

[09 - Implementation of K – Dimensional Tree 47](#_Toc201000451)

[10 - Implementation of Rabin-Krap Algorithm 48](#_Toc201000452)

[11 - Implementation of KMP Algorithm 49](#_Toc201000453)

[12 - Implementation of Union by Rank Algorithm 50](#_Toc201000454)

[13 - Implementation of Various Sorting Algorithm 51](#_Toc201000455)

[14- Implementation of Quick Sort Algorithm 52](#_Toc201000456)

[15- Implementation of Merge Sort Algorithm 53](#_Toc201000457)

[16 - Implementation of Heap Sort Algorithm 54](#_Toc201000458)

[Appendix – 06: Working with Graph using NetworkX 55](#_Toc201000459)

[Appendix – 07: Essential Problems from CLRS 60](#_Toc201000460)

# Week – 01: Sorting Problem, Time Complexity, and Asymptotic Analysis

## Lecture 01: Insertion Sort

**Topics to be Covered: -**

* Problem of Sorting, Pseudo Code,
* Insertion Sort, Loop Invariant, Runtime, Parameterise the runtime by the size of the input

Ink Drawings
Input:- A sequence of < a1, a2, ...., an> of numbers
Output:- A permutation of < a1', a2', ...., an'> such that

a1' ≤ a2' ≤ .... ≤ an
The Problem of Sorting


Ink Drawings
Example
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Pseudo Code:- Insertion Sort
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


**Do a Dry Run of the code**.

Ink Drawings
Example of Insertion Sort
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Runtime of Insertion Sort
Runtime of Insertion Sort

The running time depends on input i.e., already sorted sequence is easier to sort
Parameterize the running time by the size of the input, since short sequence are easier to sort than long one.
Generally, we seek upper bounds on running time.


## Lecture 02: Analysis of Insertion Sort

**Topics to be Covered: -**

* Types of Analysis: Worst Case, Best Case and Average Case, Machine Independency
* Asymptotic Notation, Big-Theta Notation ()

Ink Drawings
Types of Analysis
Types of Analysis

Worst Case (Usually) :- 
T(n) = Maximum time of algorithm on any input of size 'n'.

Average Case (Sometimes) :- 
T(n) = Expected time of algorithm on any input of size 'n'.

Best Case :- 
Cheat with a slow algorithm that works fast on 'some' input.


Ink Drawings
Machine-Independent Time
Machine-Independent Time
What is Insertion Sort's worst-case time?
It depends on the speed of the computer
Relative Speed (on the same machine)
Absolute Speed (on different machine)
Big Idea
Big Idea
Ignore machine-dependent constants.
Look at 'growth' of T(n) as n →∞

ASYMPTOTIC ANALYSIS


Ink Drawings
𝛉 Notation
𝛉 Notation

Maths:-
𝜽﷐𝒈﷐𝒏﷯﷯ = {𝑓﷐𝑛﷯ : ∃ 𝑝𝑜𝑠𝑖𝑡𝑖𝑣𝑒 𝑐𝑜𝑛𝑠𝑡𝑎𝑛𝑡 c1, c2 and n0 such that 0 ≤ c1 . g(n) ≤ f(n) ≤ c2 . g(n), ∀ n ≥ n0}

Engineering:- 
Drop low order terms; ignore leading constants

Example:- 
3n - 90n +5n - 1024 = θ﷐﷐𝑛﷮3﷯﷯
Ink Drawings


## Lecture 03: Asymptotic Notation

**Topics to be Covered: -**

* Asymptotic Notation: - Big-Oh, Big-Theta, and Big-Omega
* Time Complexity of Insertion Sort: - Worst Case, Best Case, and Average Case
* Merge Sort

Ink Drawings
𝚶 Notation 

Ο﷐𝑔﷐𝑛﷯﷯ = { f(n) : ∃ positive constant c and n0 such that f(n) ≤ c * g(n) ∀ n ≥ n0}
𝚶 Notation 

Ο﷐𝑔﷐𝑛﷯﷯ = { f(n) : ∃ positive constant c and n0 such that f(n) ≤ c * g(n) ∀ n ≥ n0}
 𝛀 Notation 

Ω﷐𝑔﷐𝑛﷯﷯ = { f(n) : ∃ positive constant c and n0 such that f(n) ≥ c * g(n) ∀ n ≥ n0}
 𝛀 Notation 

Ω﷐𝑔﷐𝑛﷯﷯ = { f(n) : ∃ positive constant c and n0 such that f(n) ≥ c * g(n) ∀ n ≥ n0}


Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Asymptotic Notation
Asymptotic Notation
When n gets large enough a 𝜃﷐﷐𝑛﷮2﷯﷯ algorithm always beats a Ο﷐﷐𝑛﷮3﷯﷯ algorithm.
We shouldn't ignore asymptotically slower algorithm.
Real world design situations often calls for a careful balancing of engineering objectives.
It is a useful tool to help structure our thinking.
Insertion Sort Analysis
Insertion Sort Analysis

Worst Case: Input Inversely sorted.

T(n) = ﷐𝑗=2﷮𝑛﷮𝜃﷐𝑗﷯﷯= 𝜃﷐﷐𝑛﷮2﷯﷯ [Arithmetic Series]

Average Case: All permutation equally likely.

T(n) = ﷐𝑗=2﷮𝑛﷮𝜃﷐﷐𝑗﷮2﷯﷯﷯= 𝜃﷐﷐𝑛﷮2﷯﷯

It is moderately fast for small 'n'.
It is not at all fast for large 'n'.
Ink Drawings
Ink Drawings


## Lecture 04: Recurrence for Merge Sort

Topics to be Covered: -

* Merge Sort, Run time of Merge Sort
* Recurrence and Recursive Tree

Ink Drawings
Merge Sort
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Analysis of Merge Sort
Analysis of Merge Sort
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Recurrence for Merge Sort
Recurrence for Merge Sort
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
We shall usually omit the base case when  for sufficiently small 'n' and when it has no effect on the solution to the recurrence


Ink Drawings
Recursion Tree
Recursion Tree
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Best case of Merge Sort
Best case of Merge Sort
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


## Lecture 05: Substitution Method

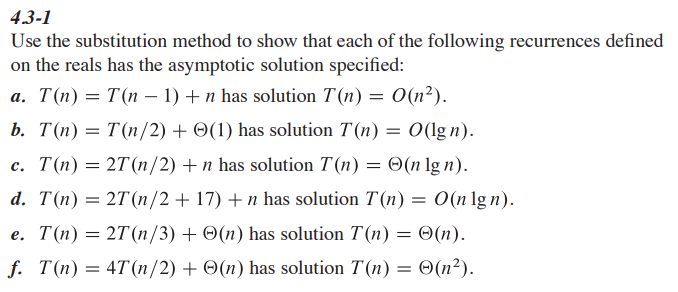
Topics to be Covered: -

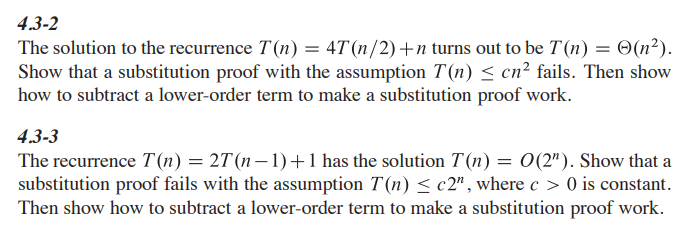
* Solving the Recurrence: Substitution Method
* Method of Induction

It is the most general method:

* Guess the form of solution
* Verify by Induction
* Solve for constants

Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings



## Weekley Test

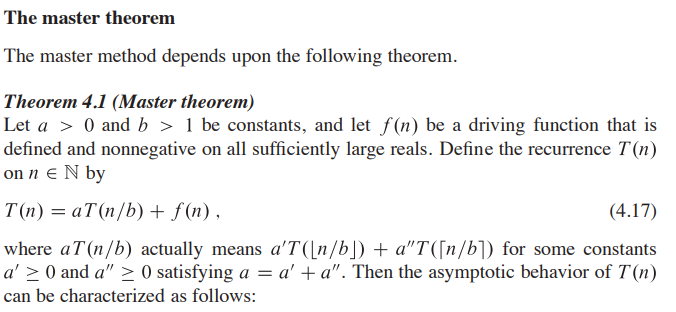
### 2023

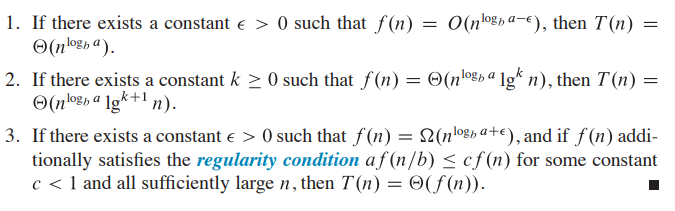
# Week – 02: Solving Recurrence, Divide and Conquer

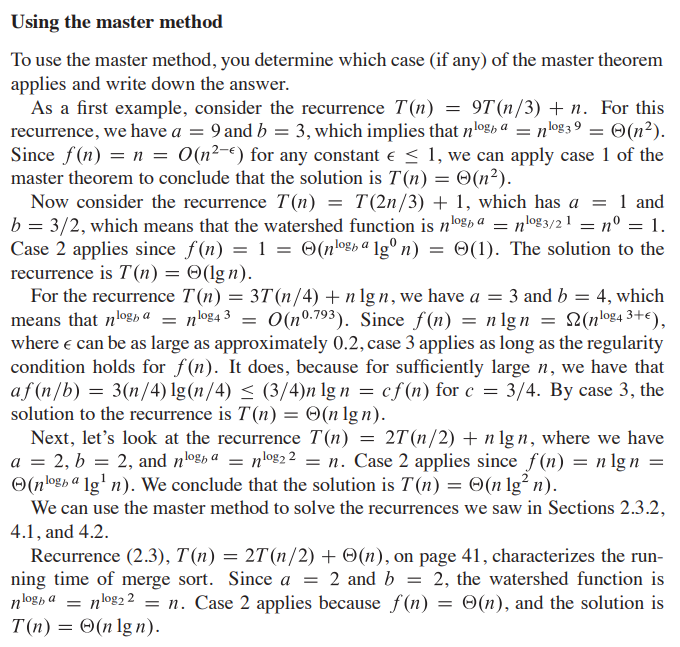
## Lecture 06: The Master Method

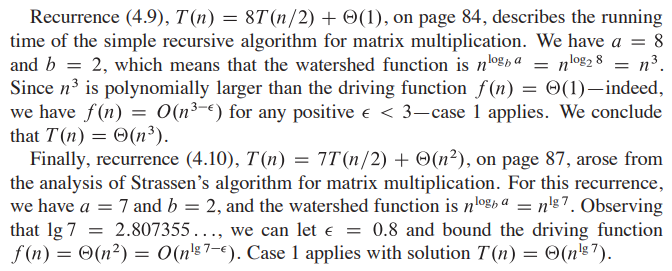
Topics to be Covered: -

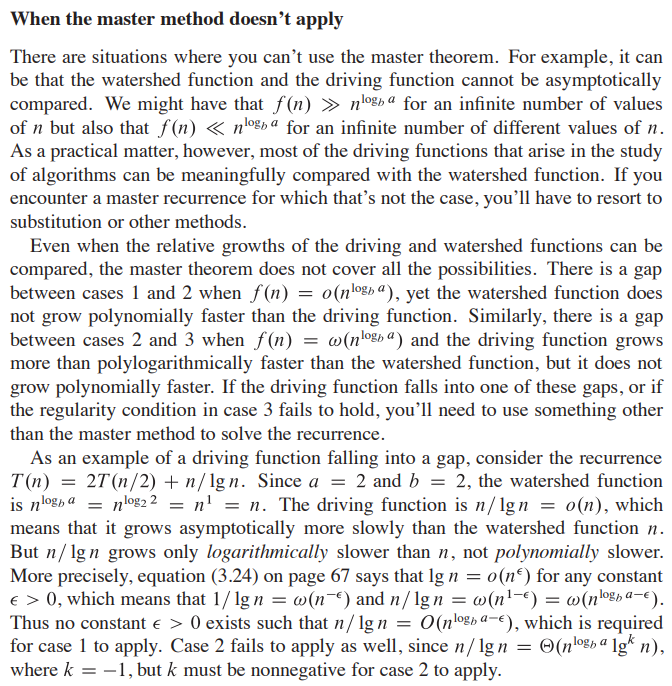
* Solving the recurrence of the form T(n) = aT(n/b) + f(n),
* Master method

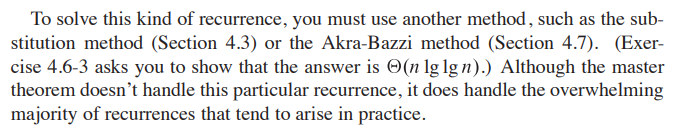
[[1]](#footnote-1)

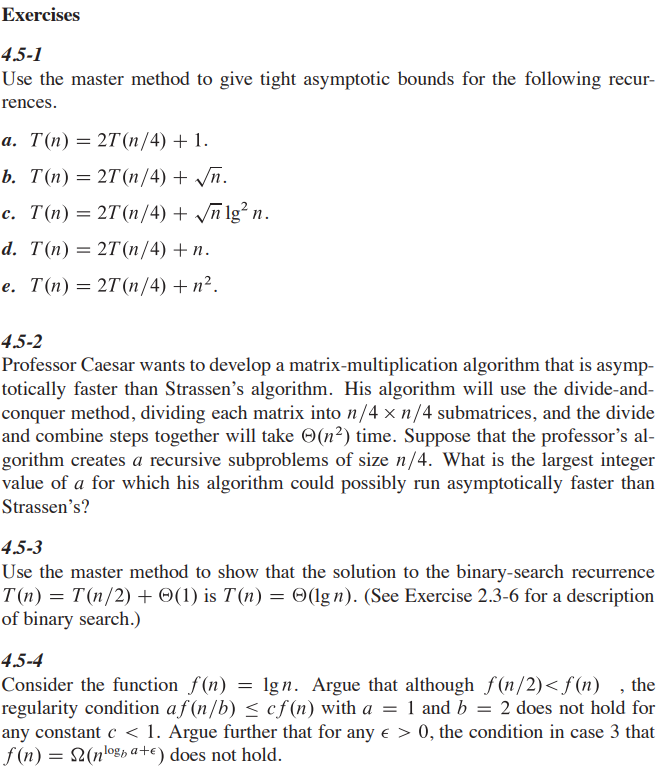


[[2]](#footnote-2)



[[3]](#footnote-3)



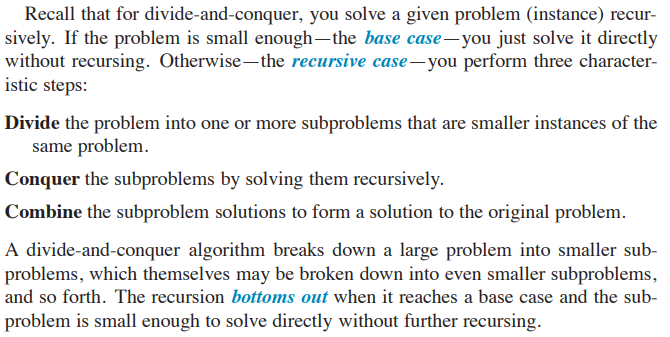
[[4]](#footnote-4)

**Read section 4.6 for better understanding.**

## Lecture 07: Divide & Conquer

Topics to be Covered:

* Divide and Conquer: Design Paradigm
* Binary Search
* Powering a Number

[[5]](#footnote-5)

Ink Drawings
Binary Search
Binary Search
Divide :- Check Middle Element
Conquer :- Recursively search one sub array
Combine :- Trivial (Not needed in case of Binary Search)
Recurrence for Binary Search
𝑻﷐𝒏﷯=𝟏 ∗𝑻﷐﷐𝒏﷮𝟐﷯﷯+ 𝜽﷐𝒏﷯
﷐𝑛﷮﷐﷐log﷮𝑏﷯﷮𝑎﷯﷯ = ﷐𝑛﷮﷐﷐log﷮2﷯﷮1﷯﷯ = ﷐𝑛﷮0﷯ =  1 ⇒ CASE II ( k = 0 )
𝑻﷐𝒏﷯= 𝜽﷐﷐𝐥𝐨𝐠﷮𝒏﷯﷯
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


# Week – 03: Quick Sort and Heap Sort, Decision Tree

# Week – 04: Linear Time Sorting, Order Statistic

# Week – 05: Hash Function, Binary Search Tree (BST) Sort

# Week – 06: Randomly Build BST, Red Black Tree, Augmentation of Data Structure

# Week – 07: Van Emde Boas, Amortized Analysis, Computational Geometry

# Week – 08: Dynamic Programming, Graph, Prim’s Algorithm

# Week – 09: BFS & DFS, Shortest Path Problem, Dijktra, Bellman-Ford

# Week – 10: All Pair Shortest Path, Floyd-Warshall, Jhonson Algorithm

# Week – 11: More Amortized Analysis, Disjoint Set Data Structure

# Week – 12: Network Flow, Computational Complexity

# Appendix – 01: Test of 2025

## Week – 01

# Appendix – 02: Important Links

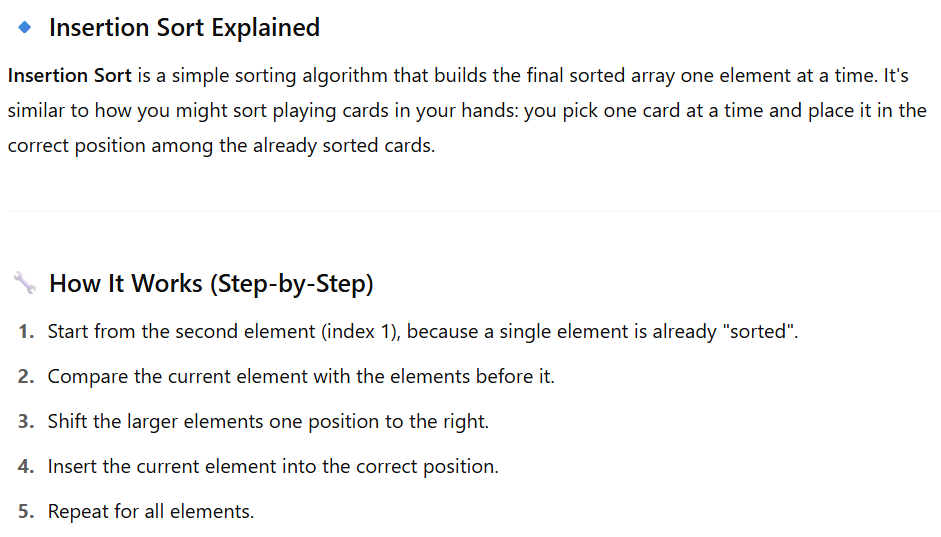
Current: - <https://onlinecourses.nptel.ac.in/noc25_cs150/preview>

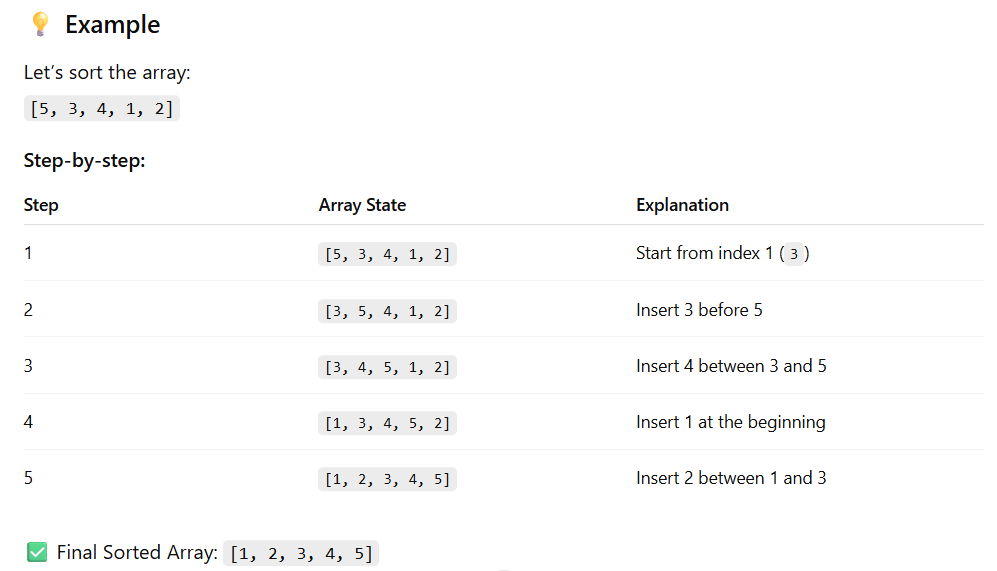
2023: - <https://onlinecourses.nptel.ac.in/noc23_cs88/preview>

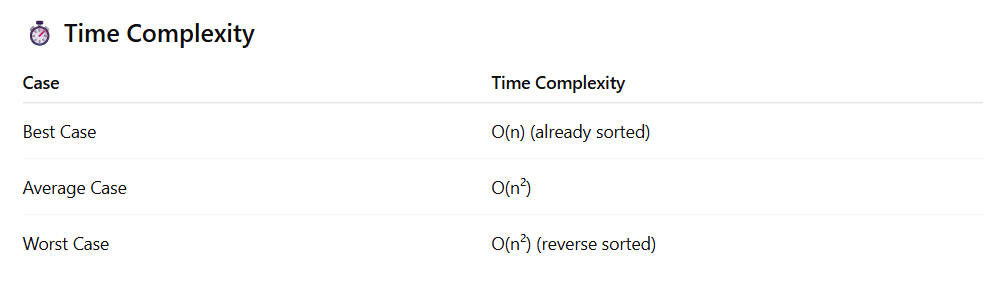
NPTEL: - <https://nptel.ac.in/courses/106105164>

# Appendix – 03: Chat GPT and Deep Seek

## Insertion Sort







|  |
| --- |
| # Python Code for Insertion Sort  def insertion\_sort(arr):  for i in range(1, len(arr)):  key = arr[i]  j = i - 1    # Move elements greater than key to one position ahead  while j >= 0 and arr[j] > key:  arr[j + 1] = arr[j]  j -= 1    arr[j + 1] = key  # Example  arr = [5, 3, 4, 1, 2]  insertion\_sort(arr)  print("Sorted array:", arr) |

Walk through the **step-by-step logic to code Insertion Sort**, assuming you understand loops, conditionals, and lists.

**🧩 Step-by-Step Plan to Code Insertion Sort**

Let’s say we want to sort the list arr = [5, 3, 4, 1, 2].

✅ **Step 1: Understand the Goal**

We want to sort arr in ascending order using the insertion sort technique.

**✅ Step 2: Loop Through the Array**

Start the loop from the second element (index 1), because the first element is trivially sorted.

**for i in range(1, len(arr)):**

**✅ Step 3: Store the Current Value**

Pick the current value (to be inserted into the sorted part on the left).

**key = arr[i]**

**✅ Step 4: Set Pointer for Comparison**

Set a pointer j to the element just before i. We'll use this to compare with the elements on the left side.

**j = i - 1**

**✅ Step 5: Shift Elements Greater Than key**

While j is not out of bounds and the current left-side element is greater than key, shift it to the right.

while j >= 0 and arr[j] > key:

arr[j + 1] = arr[j] # shift right

j -= 1 # move left

**✅ Step 6: Insert the key at the Correct Position**

Now insert the key at j + 1 because the last checked position (j) was either out of bounds or smaller than key.

arr[j + 1] = key

**✅ Step 7: Full Code Together**

See above

**✅ Step 8: Test It**

arr = [5, 3, 4, 1, 2]

insertion\_sort(arr)

print("Sorted array:", arr)

**Output:**

Sorted array: [1, 2, 3, 4, 5]

**🎯 Optional Tip for You**

Add print() statements to see what's happening at each step:

def insertion\_sort(arr):

for i in range(1, len(arr)):

key = arr[i]

j = i - 1

print(f"\nInserting {key}")

while j >= 0 and arr[j] > key:

arr[j + 1] = arr[j]

j -= 1

print(f"Moved {arr[j + 1]} to position {j + 2}")

arr[j + 1] = key

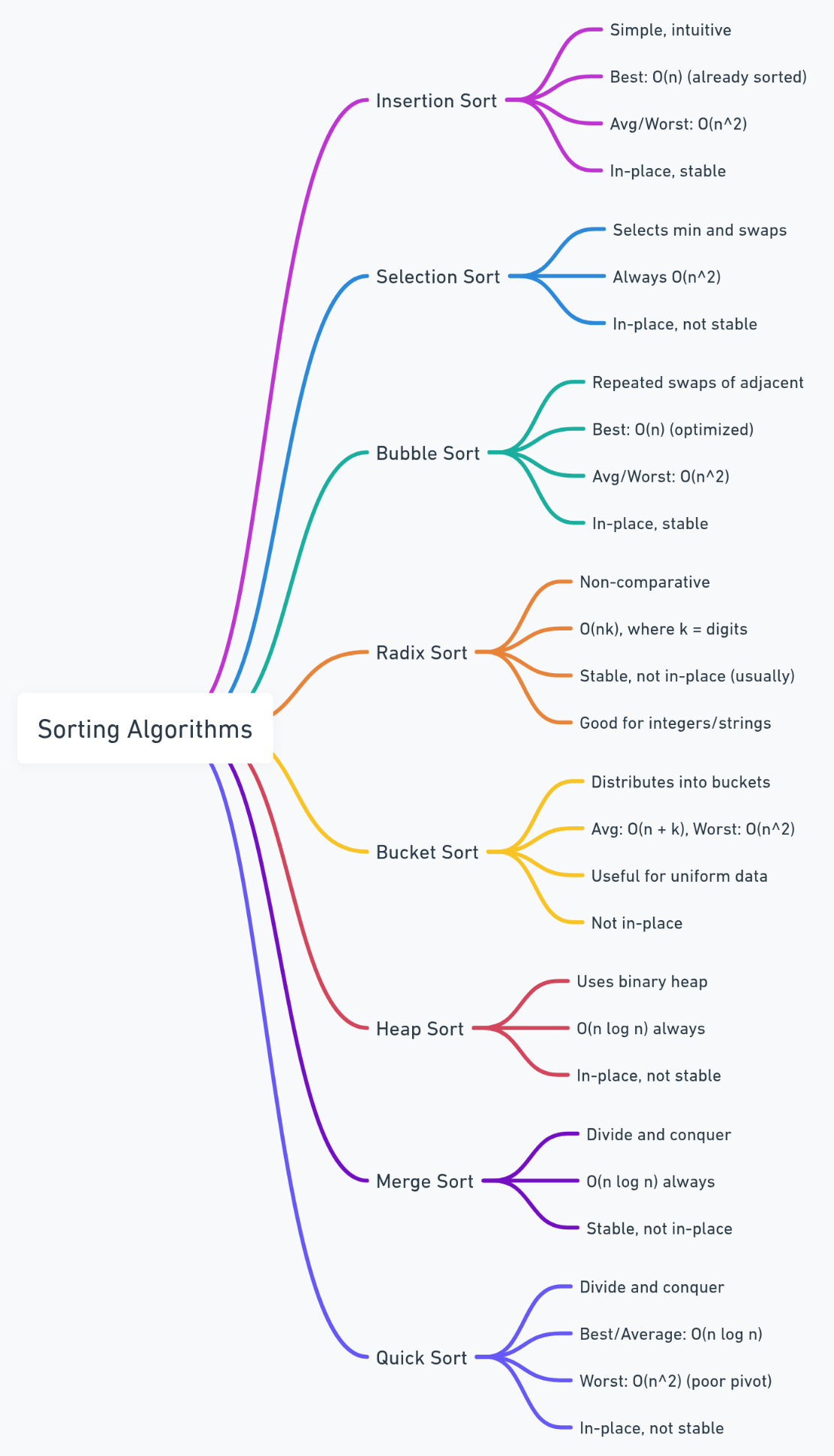
print(f"Inserted {key} at position {j + 1}")

print("Current array state:", arr)

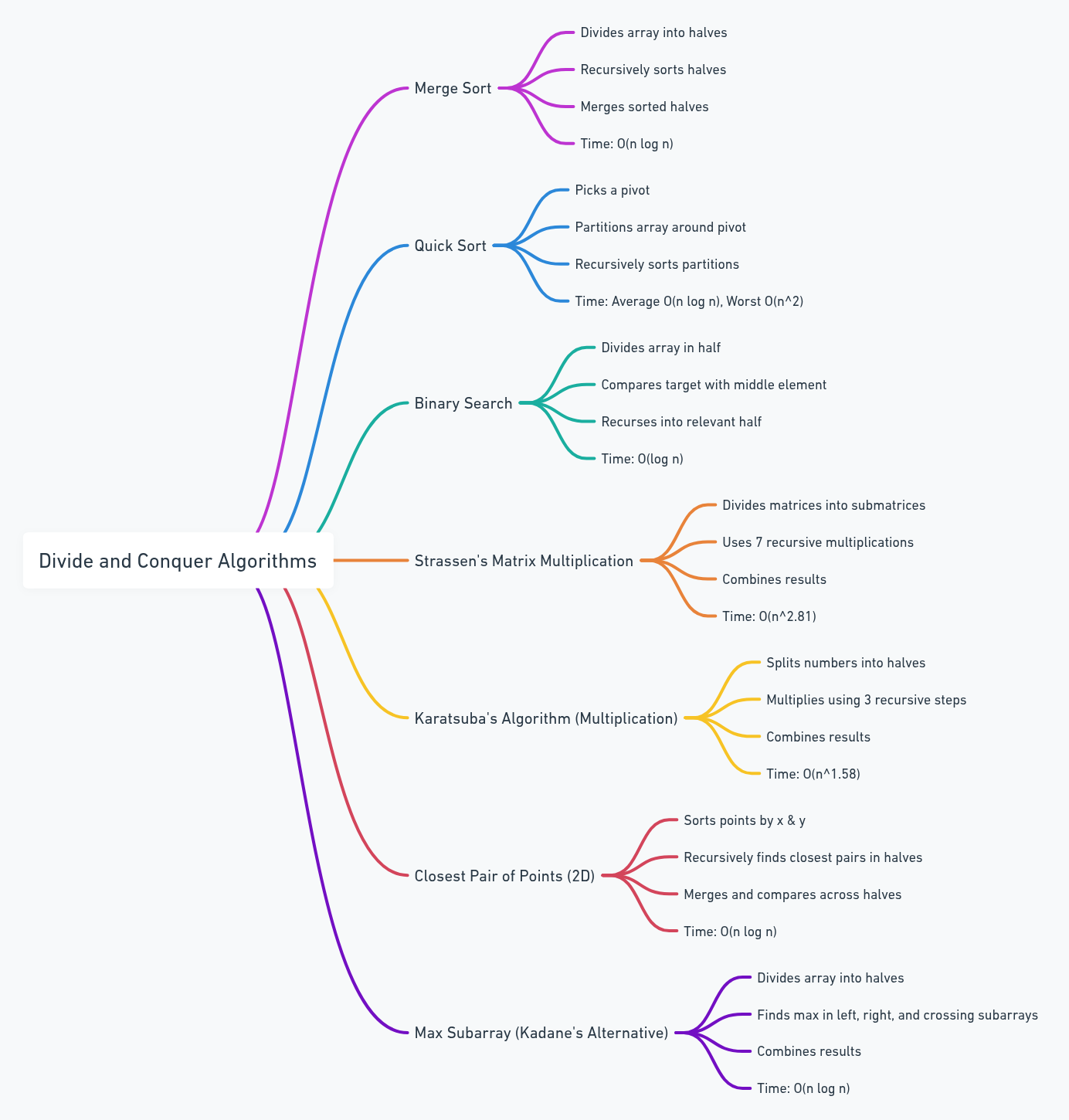
**Create Insertion Sort in Decreasing order.**

## Whimsical Diagrams

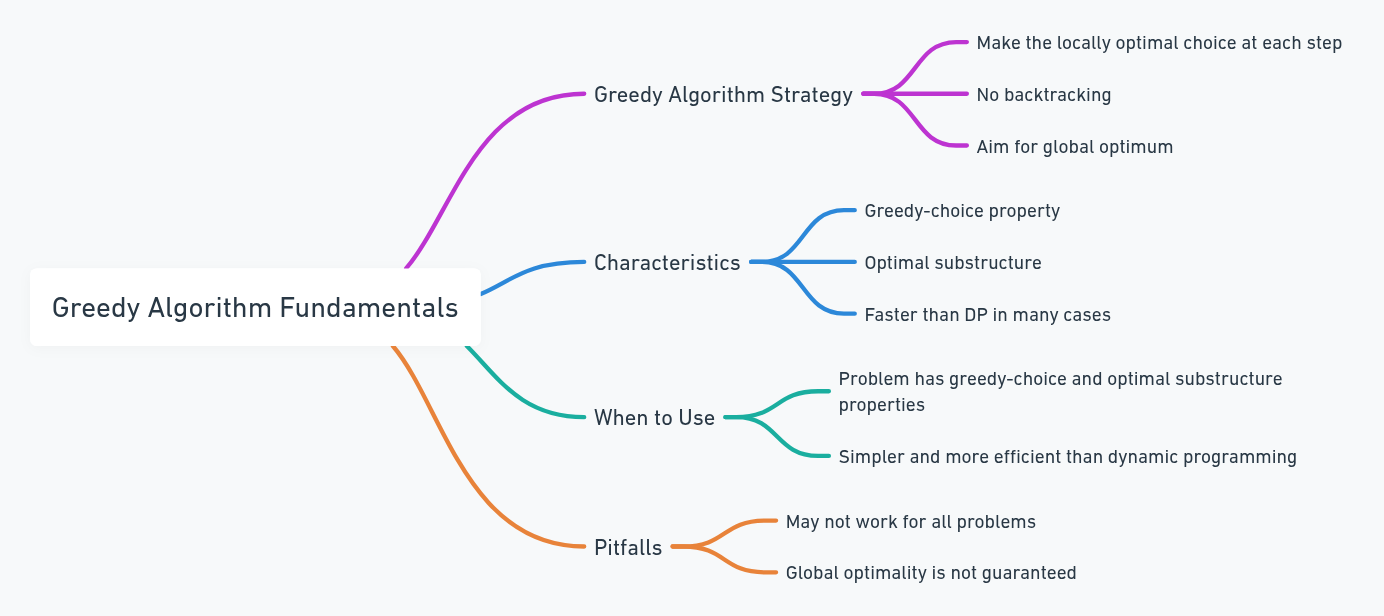
### Sorting Technique



### Divide and Conquer Algorithm

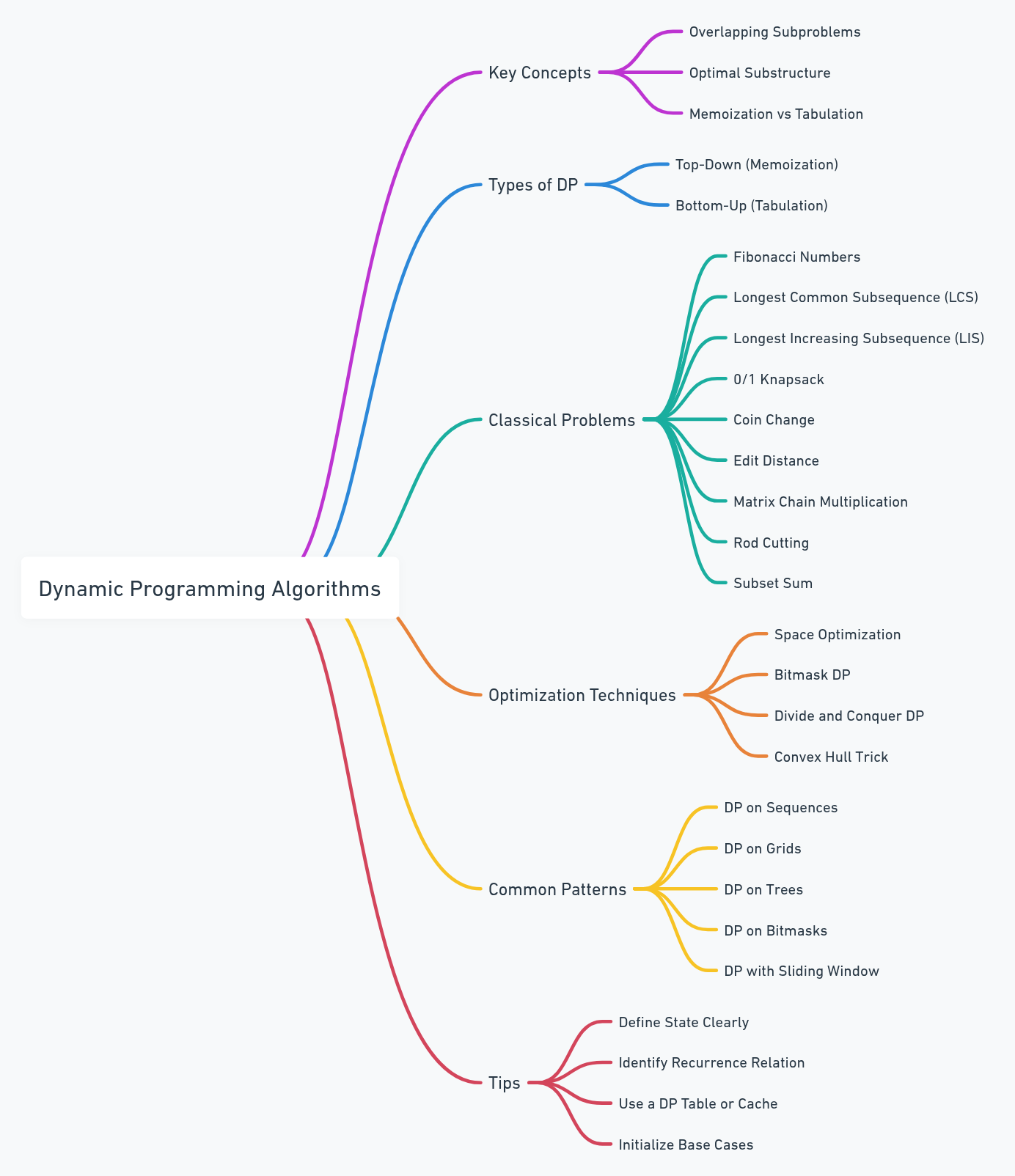


### Greedy Algorithm



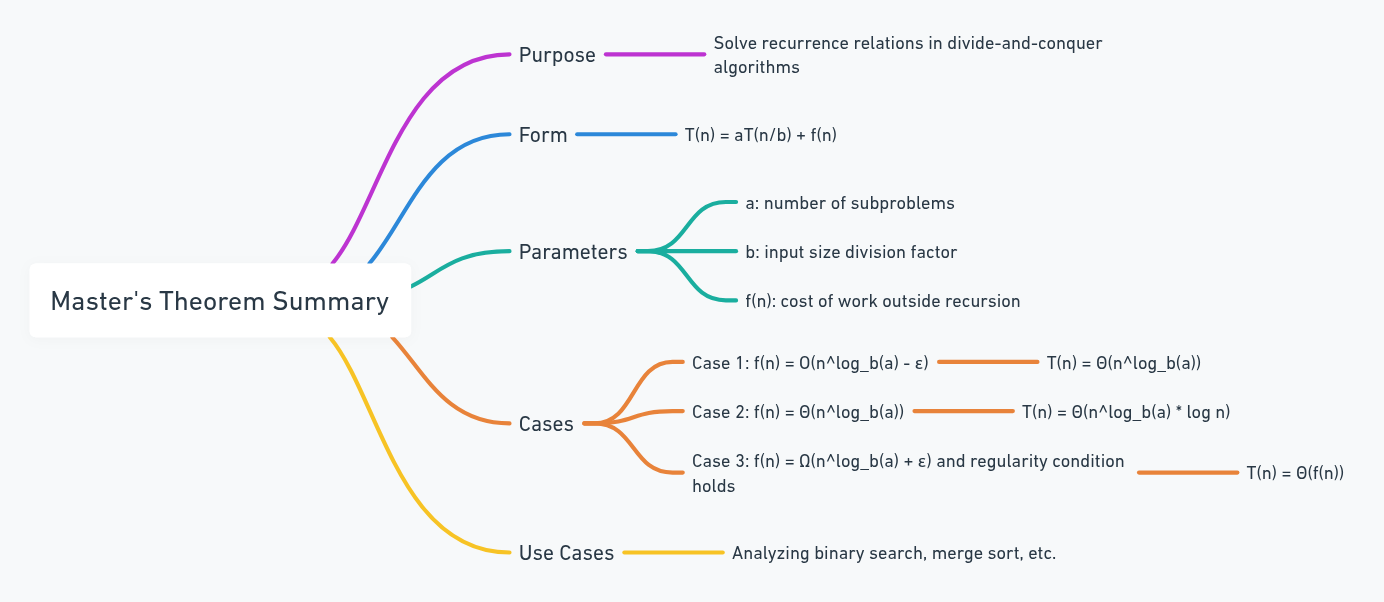


### Dynamic Programming





### Master’s Theorem



# Appendix – 04: Python Setup Guide

**🖥️ 1. Installing Python**

**🔹 Windows:**

1. Download the installer from: <https://www.python.org/downloads>
2. Run the installer:
   * ✅ Check **“Add Python to PATH”**
   * Click **Install Now**
3. Verify installation:

|  |
| --- |
| python –version |

**🔹 Linux (Ubuntu/Debian):**

|  |
| --- |
| sudo apt update  sudo apt install python3 python3-pip python3-venv  # Verify Installation  python –version |

**🧪 2. Create a Virtual Environment**

**🔹 Windows:**

|  |
| --- |
| python -m venv algo-env  .\algo-env\Scripts\activate |

**🔹 Linux:**

Bash

|  |
| --- |
| python3 -m venv algo-env  source algo-env/bin/activate |

**Deactivate with:**

|  |
| --- |
| deactivate |

**📦 3. Install Essential Packages**

Once the virtual environment is activated, install these recommended packages:

|  |
| --- |
| pip install jupyter matplotlib numpy pandas networkx rich |

🔍 Purpose of These Packages:

|  |  |
| --- | --- |
| **Package** | **Purpose** |
|  | Run notebooks for interactive coding |
|  | Visualization and plotting |
|  | Numerical computing |
|  | Data structures and manipulation |
|  | |  | | --- | |  |  |  | | --- | | Graph theory & algorithm practice | |
|  | |  | | --- | |  |  |  | | --- | | Beautiful CLI output (optional) | |

**📓 4. Launching Jupyter Notebook**

In your project folder:

|  |
| --- |
| jupyter notebook |

A browser will open. You can create ‘.ipynb’ files and run code cells interactively — great for learning and testing algorithms.

**You can create a ‘requirements.txt’to share your setup:**

|  |
| --- |
| pip freeze > requirements.txt |

**🛠️ 6. Optional Tools (Highly Recommended)**

|  |  |  |
| --- | --- | --- |
| **Tool** | **Use Case** | **Install Command** |
|  | Auto-code formatter | pip install black |
|  | Testing algorithms | pip install pytest |
|  | Enhanced interactive shell | pip install ipython |
|  | Advanced graph visualizations | see note below |

⚠️ pygraphviz may require additional system libraries:  
On Ubuntu:

|  |
| --- |
| sudo apt install graphviz libgraphviz-dev |

# Appendix – 05: Step-by-Step Guide of Various Algorithm with Python Code

## 01 – Implementation of Dijkstra’s Algorithm

## 02 - Implementation of Bellman-Ford Algorithm

## 03 - Implementation of Kahn’s Algorithm

## 04 - Implementation of Dinic’s Algorithm

## 05 - Implementation of Ford-Fulkerson Algorithm

## 06 - Implementation of Prim’s Algorithm

## 07 - Implementation of Kruskal’s Algorithm

## 08 - Implementation of Basic Operation Associated with B+ Tree

## 09 - Implementation of K – Dimensional Tree

## 10 - Implementation of Rabin-Krap Algorithm

## 11 - Implementation of KMP Algorithm

## 12 - Implementation of Union by Rank Algorithm

## 13 - Implementation of Various Sorting Algorithm

## 14- Implementation of Quick Sort Algorithm

## 15- Implementation of Merge Sort Algorithm

## 16 - Implementation of Heap Sort Algorithm

# Appendix – 06: Working with Graph using NetworkX

(https://chatgpt.com/c/68490260-0394-800c-a581-9d6389235c43)

**🧾 Introduction**

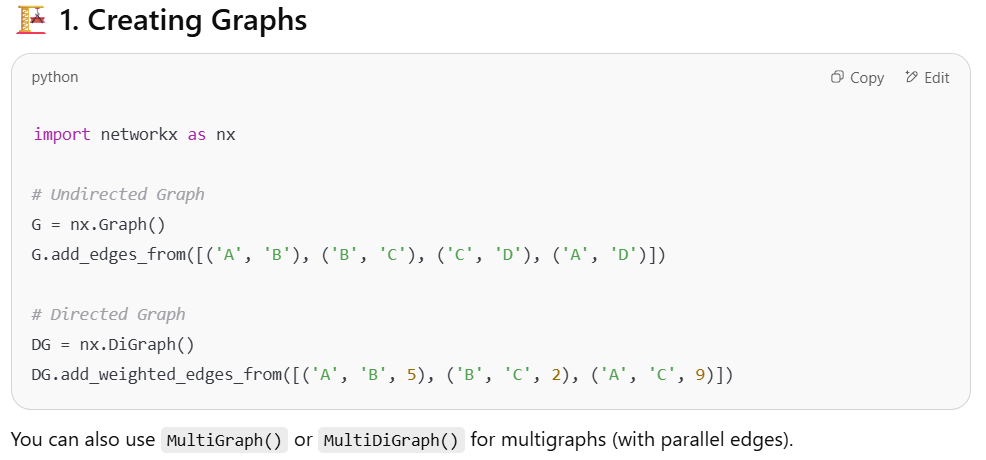
Graphs are fundamental in computer science and algorithm design. They model relationships between entities—like cities on a map, web pages, social networks, and more.

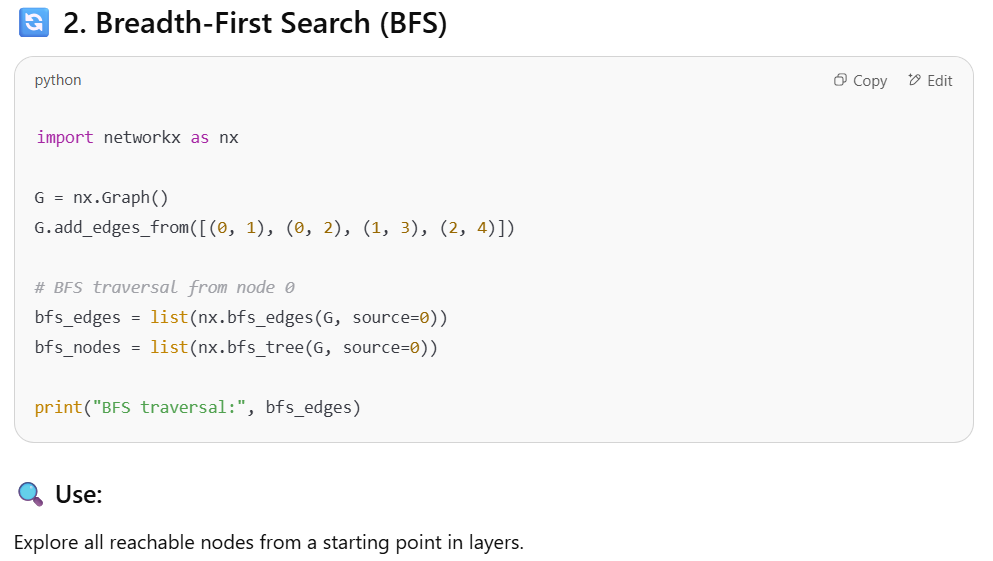
This appendix introduces [**NetworkX**](https://networkx.org/)(https://networkx.org/)—a Python library for creating, manipulating, and visualizing complex networks—to help you **experiment interactively** with graph algorithms while learning them.

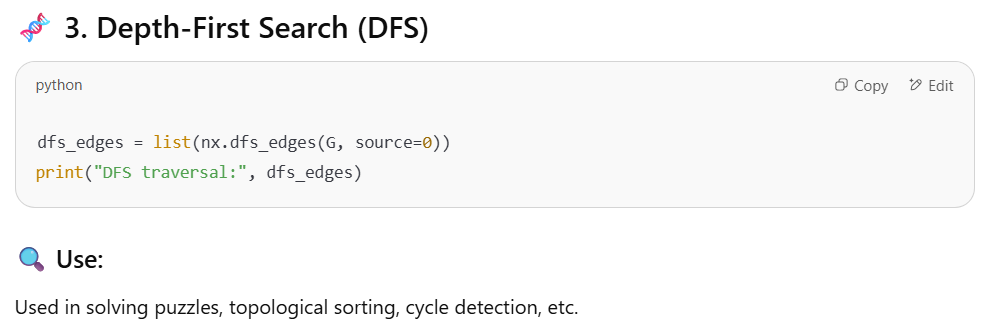
**✅ Why** [**NetworkX**](https://networkx.org/) **is Useful**

* Easy to create and visualize graphs
* Supports directed, undirected, weighted, and multigraphs
* Built-in implementations of many classic graph algorithms
* Useful for both **learning concepts** and **experimenting interactively**

**Here are some examples, please develop some more by yourself.**

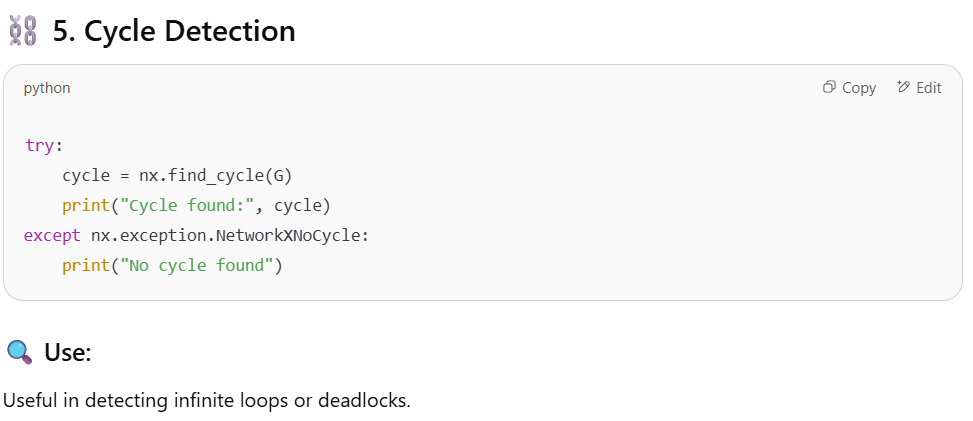


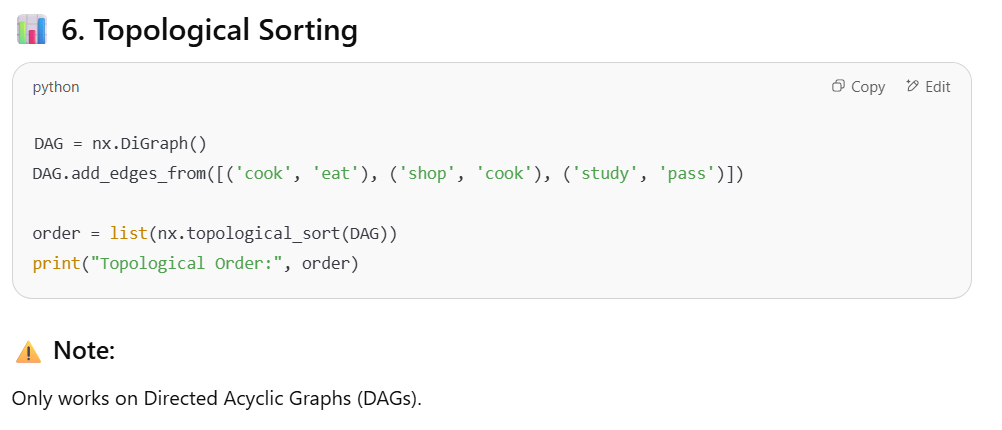


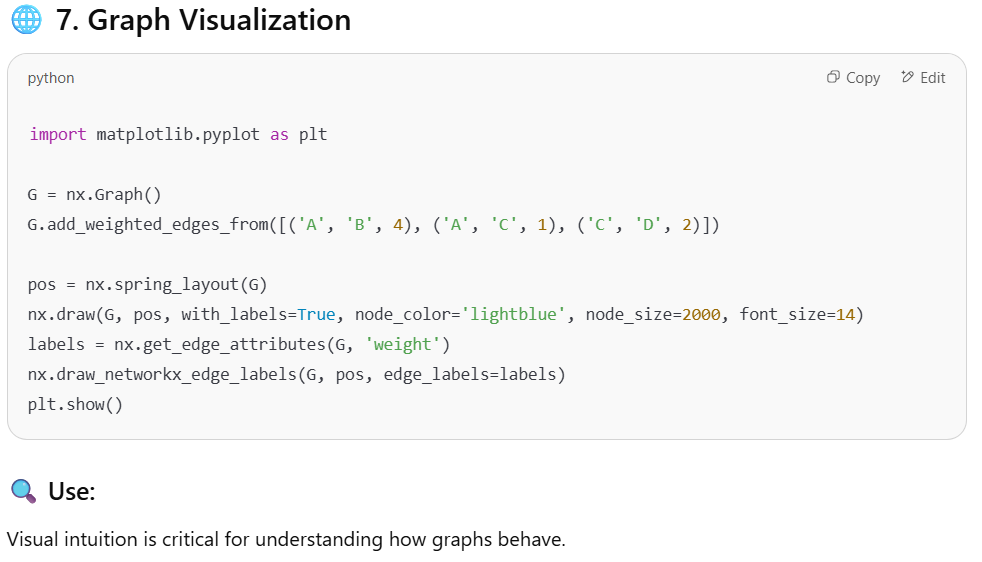






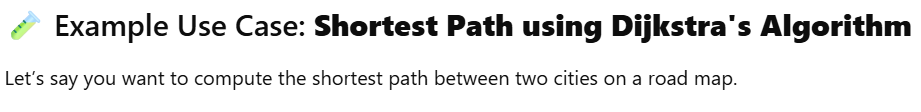












|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  # Step 1: Create a directed weighted graph  G = nx.DiGraph()  # Step 2: Add edges (node1, node2, weight)  G.add\_weighted\_edges\_from([  ('A', 'B', 4),  ('A', 'C', 2),  ('B', 'C', 5),  ('B', 'D', 10),  ('C', 'D', 3)  ])  # Step 3: Compute shortest path from A to D  path = nx.dijkstra\_path(G, source='A', target='D')  length = nx.dijkstra\_path\_length(G, source='A', target='D')  print("Shortest path:", path)  print("Path length:", length)  # Step 4: Visualize the graph  pos = nx.spring\_layout(G)  nx.draw(G, pos, with\_labels=True, node\_color='skyblue', node\_size=2000, font\_size=16)  edge\_labels = nx.get\_edge\_attributes(G, 'weight')  nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels)  plt.show() |

# Appendix – 07: Essential Problems from CLRS

**🧠 Essential Problems from CLRS (by Chapter)**

**Chapter 2: Getting Started**

* Insertion Sort (2.1)
* Merge Sort – including loop invariants (2.3)
* Binary Search (Exercise 2.3-5)
* Inversions in an array (Problem 2-4)

**Chapter 3: Growth of Functions**

* Asymptotic notation comparison problems
* Exercise 3.1-1 to 3.1-6 – proving O, Θ, and Ω relationships
* Exercise 3.2-3 – use of limits in asymptotic behaviour

**Chapter 4: Divide-and-Conquer**

* Maximum Subarray Problem (4.1)
* Recurrence Tree Method and Master Theorem (4.3)
* Strassen’s Matrix Multiplication (4.2)

**Chapter 6: Heapsort**

* Build-Max-Heap and Max-Heapify (6.3)
* Implement Priority Queue with Heap
* Median maintenance with two heaps (advanced)

**Chapter 7–8: Quicksort & Sorting Lower Bounds**

* Randomized Quicksort (7.3)
* Worst-case for Quicksort (Problem 7-1)
* Counting Sort and Radix Sort (8.2, 8.3)
* Lower bounds for comparison sorts (8.1)

**Chapter 9: Medians and Order Statistics**

* Randomized-Select algorithm (9.2)
* Deterministic Select – Median of Medians (9.3)

**Chapter 10–11: Elementary Data Structures & Hashing**

* Stack, Queue, Linked List operations (10.1–10.3)
* Hash Table with chaining and open addressing (11.2–11.4)
* Universal hashing (11.3)

**Chapter 12–13: Binary Search Trees & Red-Black Trees**

* In-order Traversal
* Search, Min, Max, Successor, Predecessor (12.2)
* Insert and Delete in BST
* Red-Black Tree Insertion & Deletion (13.3)

**Chapter 15: Dynamic Programming**

* Matrix Chain Multiplication (15.2)
* Longest Common Subsequence (15.4)
* Rod Cutting (15.1)
* Optimal BST (15.5, advanced)

**Chapter 16: Greedy Algorithms**

* Activity Selection Problem (16.1)
* Huffman Coding (16.3)
* Fractional Knapsack (Problem 16-1)

**Chapter 22–24: Graph Algorithms**

* BFS and DFS (22.2, 22.3)
* Topological Sort (22.4)
* Strongly Connected Components (22.5)
* Dijkstra’s Algorithm (24.3)
* Bellman-Ford Algorithm (24.1)
* Floyd-Warshall Algorithm (25.2)
* Minimum Spanning Trees: Prim’s and Kruskal’s (23.1, 23.2)

**Chapter 26–27: Max Flow**

* Ford-Fulkerson Algorithm (26.2)
* Bipartite Matching using flow (26.1, 26.3)
* Push-Relabel Algorithm (27.2)

1. “Introduction to Algorithm – CLRS, 4TH ed”, Page 102 [↑](#footnote-ref-1)
2. “Introduction to Algorithm – CLRS, 4TH ed”, Page 104 [↑](#footnote-ref-2)
3. “Introduction to Algorithm – CLRS, 4TH ed”, Page 105 [↑](#footnote-ref-3)
4. “Introduction to Algorithm – CLRS, 4TH ed”, Page 106 [↑](#footnote-ref-4)
5. “Introduction to Algorithm – CLRS, 4TH ed”, Page 76 [↑](#footnote-ref-5)