Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES
23CS03HF

Topic:

String Matching Algorithms

Session - 29

AIM OF THE SESSION

To familiarize students with the concept of String Matching algorithm.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate :- String Matching algorithm.
2. Describe :- Types of String Matching algorithms.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define :- String Matching algorithm
2. Describe :- Types of String Matching algorithms.
3. Summarize:- Naïve string matching algorithm.

String Matching

- Text-editing programs frequently need to find all occurrences of a pattern in the text.

- Given a text is an array T[1..n] of length n and a pattern P[1..m] of length m<=n.
- We say that pattern P occurs with shift **s** in text T if **$0 \le s \le n-m$** and **T[s+1..s+m] = P[1..m].**

- If P occurs with shift **s** in T , then we call s a **valid** shift; otherwise, we call **s** an **invalid** shift.

Definition

- The string-matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T .

- Example: T = ababcabdabcaabc and P = abc, the occurrences are:
  - first occurrence starts at T[3]
  - second occurrence starts at T[9]
  - third occurrence starts at T[13]

Applications of String Matching

- Searching keywords in a file
- Searching engines (like Google and Openfind)
- Database searching (GenBank)

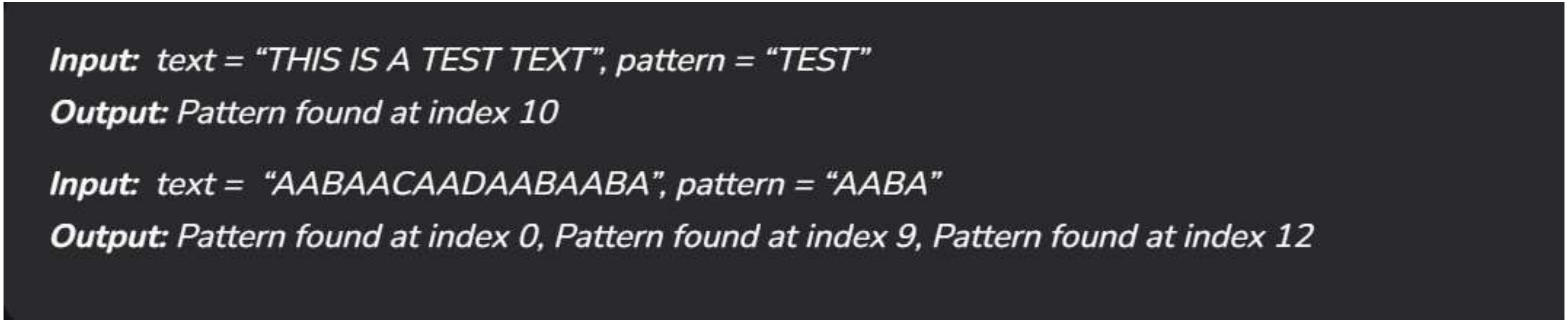
Types of String Matching Algorithms

- Naive String matching algorithm
- Robin-Krap algorithm
- Knuth–Morris–Pratt algorithm
- String matching with Finite automata

The naive string-matching algorithm

The naive algorithm finds all valid shifts using a loop that checks the condition.

P [1..m] = T[s+1..s+m] for each of the n-m+1 possible values of s.



Input: *text = "THIS IS A TEST TEXT", pattern = "TEST"*

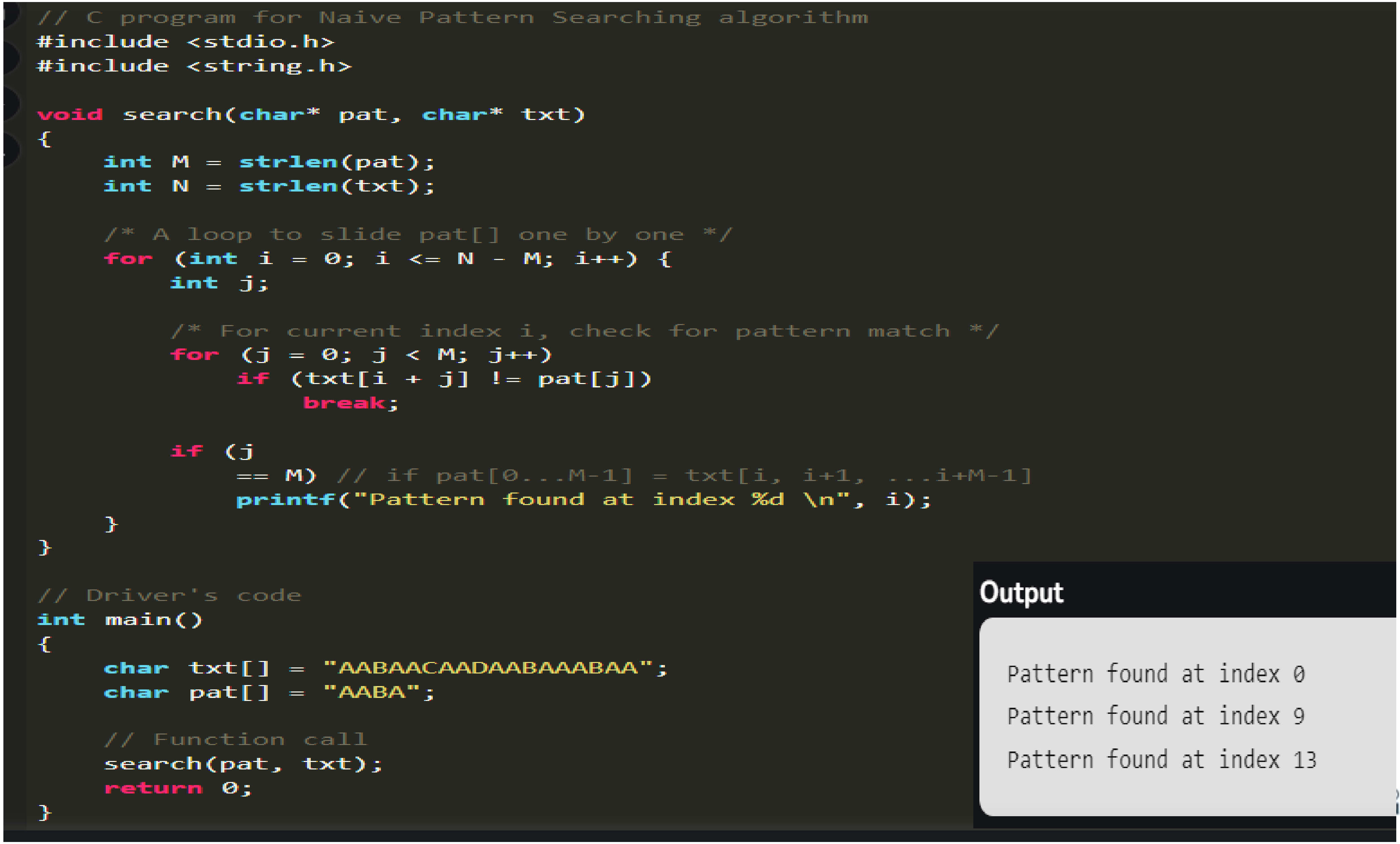
Output: *Pattern found at index 10*

Input: *text = "AABAACAADAABAABA", pattern = "AABA"*

Output: *Pattern found at index 0, Pattern found at index 9, Pattern found at index 12*

Algorithm

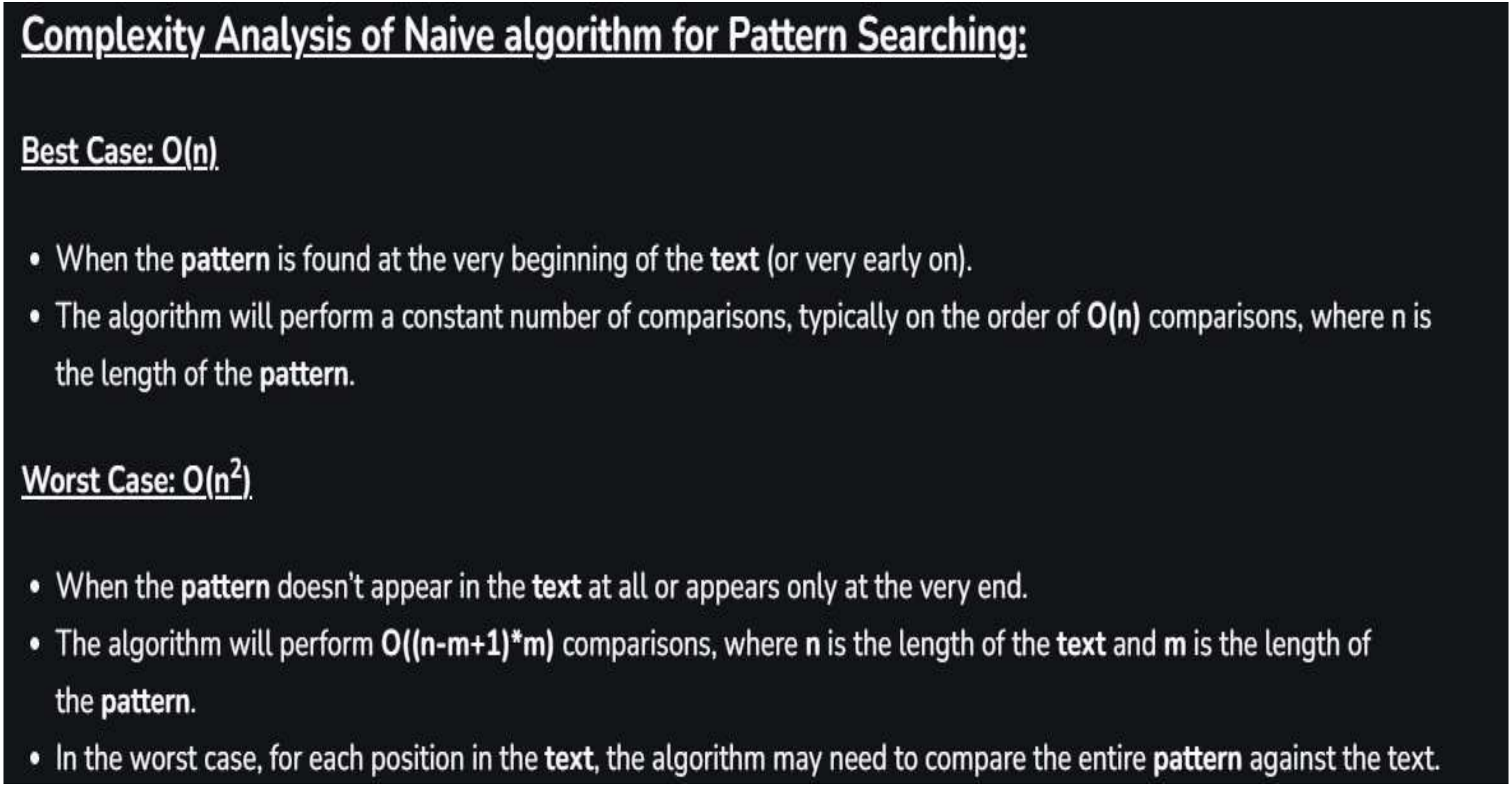
```
Naïve-string-Matcher(T,P)
   n=T. length
   m=P. length
   for s=0 to n-m
    if P[1..m]==T[s+1..s+m]
    print "Pattern occurs shift s"
```



```c
// C program for Naive Pattern Searching algorithm
#include <stdio.h>
#include <string.h>

void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++) {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j
            == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            printf("Pattern found at index %d \n", i);
    }
}

// Driver's code
int main()
{
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";

    // Function call
    search(pat, txt);
    return 0;
}
```

Output

```
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
```



Complexity Analysis of Naive algorithm for Pattern Searching:

Best Case: O(n)

- When the **pattern** is found at the very beginning of the **text** (or very early on).
- The algorithm will perform a constant number of comparisons, typically on the order of **O(n)** comparisons, where n is the length of the **pattern**.

Worst Case: $O(n^2)$

- When the **pattern** doesn't appear in the **text** at all or appears only at the very end.
- The algorithm will perform **O((n-m+1)*m)** comparisons, where **n** is the length of the **text** and **m** is the length of the **pattern**.
- In the worst case, for each position in the **text**, the algorithm may need to compare the entire **pattern** against the text.

Example

T :

a	b	c	a	b	d	a	a	b	c	d	e

P :

a	b	d

Example (Step – 1)

T :	a	b	c	a	b	d	a	a	b	c	d	e

P :	a	b	d

Mismatch after 3 Comparisons

(Step – 2)

T :	a	b	c	a	b	d	a	a	b	c	d	e
P :		a	b	d								

Mismatch after 1 Comparison

Example (Step – 3)

T :	a	b	c	a	b	d	a	a	b	c	d	e
P :			a	b	d							

Mismatch after 1 Comparison

Example (Step – 4)

T :	a	b	c	a	b	d	a	a	b	c	d	e
P :				a	b	d						

Match found after 3 Comparisons.

Thus, after total 8 comparisons the substring P is found in T.

Find whether pattern exists in the text or not using naïve string matching algorithm

T :	a	b	c	a	a	d	c	a	b	a	d	e

P :	b	a	d

Advantages

- The comparison of the pattern with the given string can be done in any order
- There is no extra space required
- The most important thing that it doesn’t require the pre-processing phase, as the running time is equal to matching time

Drawbacks

- There is only one disadvantage of the naïve string matching approach, which is that it is **inefficient**.
- This is because when it has found a position, it does not use it again to find the other position. It goes back to the starting point and looks for the pattern over again.
- And so, it does not use the information from the previous shift again

SUMMARY

String matching is the method to find a place where one is several strings are found within the larger string.

SELF-ASSESSMENT QUESTIONS

Which of the following is an application of string algorithms?

(a) Text editing documents

(b) Knapsack problem6

(c) Graph coloring

(d) Sorting

(a) (5,3,8,4,7,1,6,2)

(b) (1,6,3,8,3,2,4,7)

(c) (4,1,5,8,6,3,7,2)

(d) (6,2,7,1,4,8,5,3)

TERMINAL QUESTIONS

1. What is the time complexity of Naive String Matching algorithm.
2. What are the advantages and limitations of Naive String Matching algorithm?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.
2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.
3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.
4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=
2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

THANK YOU