Unit I

CHAPTER 4

ALGORITHMIC STRATEGIES

8Y1KQC

Learning Objectives

At the end of this chapter the students will be able to:

- Know the basics and technical perspective of algorithms.
- Understand the efficiency, time and space complexity of an algorithm.
- Develop and analyze algorithms for searching and sorting.
- Learn about dynamic programming through algorithmic approach.

4.1 Introduction to Algorithmic strategies

An algorithm is a finite set of instructions to accomplish a particular task. It is a step-by-step procedure for solving a given problem. An algorithm can be implemented in any suitable programming language.

Algorithms must have input, output and should satisfy the following characteristics such as definiteness, correctness and effectiveness. Data are maintained and manipulated effectively through data structures. Algorithms can be developed to store, manipulate and retrieve data from such data structures. Examples for data structures are arrays, structures, list, tuples, dictionary etc.

Search	To search an item in a data structure using linear and binary search.
Sort	To sort items in a certain order using the methods such as bubble sort, insertion sort, selection sort, etc.
Insert	To insert an item (s) in a data structure.
Update	To update an existing item (s) in a data structure.
Delete	To delete an existing item (s) in a data structure.

The way of defining an algorithm is called algorithmic strategy. For example to calculate factorial for the given value **n** then it can be done by defining the function to calculate factorial once for the iteration-1 then it can be called recursively until the number of required iteration is reached.

DO YOU KNOW?

The word Algorithm comes from the name of a Persian author, Abu Jafar Mohammed ibn Musa al Khowarizmi(c. 825 AD(CE)), who wrote a textbook on mathematics. The word Algorithm has come to refer to a method to solve a problem.

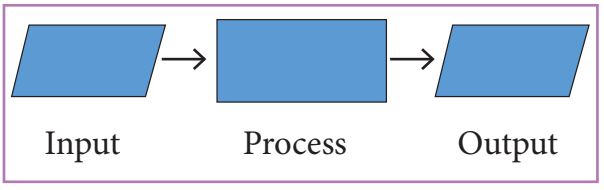
4.1.1Characteristics of an Algorithm

An algorithm should have the following characteristics:

Input	Zero or more quantities to be supplied.
Output	At least one quantity is produced.
Finiteness	Algorithms must terminate after finite number of steps.
Definiteness	All operations should be well defined. For example operations involving division by zero or taking square root for negative number are unacceptable.
Effectiveness	Every instruction must be carried out effectively.
Correctness	The algorithms should be error free.
Simplicity	Easy to implement.
Unambiguous	Algorithm should be clear and unambiguous. Each of its steps and their inputs/outputs should be clear and must lead to only one meaning.
Feasibility	Should be feasible with the available resources.
Portable	An algorithm should be generic, independent of any programming language or an operating system able to handle all range of inputs.
Independent	An algorithm should have step-by-step directions, which should be independent of any programming code.

4.1.2 Writing an Algorithm

Algorithms are generic and not limited to computer alone. It can be used in various real time activities also. Knowingly or unknowingly we perform many algorithms in our daily life such as packing books in school bag, finding shortest path to search a place, scheduling day-to-day activities, preparation for examination, etc. As we know that all programming languages share basic code constructs like conditions and iterations can be used to write an algorithm. A typical algorithm is shown in the following Figure 4.1.



A Typical Algorithm

Example

Consider the example of Coffee preparation. To make coffee, we need to have the following ingredients: Water, milk, coffee powder and sugar. These ingredients are the inputs of an algorithm. Preparing a cup of coffee is called process. The output of this process is coffee.

The procedure for preparing coffee is as follows:

1. Take a bowl with coffee powder
2. Boil the water and pour it into the bowl

3. Filter it
4. Boil milk
5. Mix sugar and filtered coffee along with boiled milk
6. Pour the coffee into the cup to serve

This kind of procedure can be represented using an algorithm. Thus, the algorithm consists of step-step-by instructions that are required to accomplish a task and helps the programmer to develop the program.

Problem: Design an algorithm to find square of the given number and display the result.

The algorithm can be written as:

Step 1 – start the process

Step 2 – get the input x

Step 3 –calculate the square by multiplying the input value ie., square ← x* x

Step 4 – display the result square

Step 5 – stop

Algorithm could be designed to get a solution of a given problem. A problem can be solved in many ways. Among many algorithms the optimistic one can be taken for implementation.

An algorithm that yields expected output for a valid input is called an algorithmic solution.

Algorithm	Program
• Algorithm helps to solve a given problem logically and it can be contrasted with the program	• Program is an expression of algorithm in a programming language
• Algorithm can be categorized based on their implementation methods, design techniques etc	• Algorithm can be implemented by structured or object oriented programming approach
• There is no specific rules for algorithm writing but some guidelines should be followed.	• Program should be written for the selected language with specific syntax
• Algorithm resembles a pseudo code which can be implemented in any language	• Program is more specific to a programming language

Table 4.1 Algorithm Vs Program

4.1.3. Analysis of Algorithm

Computer resources are limited. Efficiency of an algorithm is defined by the utilization of time and space complexity.

Analysis of an algorithm usually deals with the running and execution time of various operations involved. The running time of an operation is calculated as how many programming instructions executed per operation.

Analysis of algorithms and performance evaluation can be divided into two different phases:

1. **A Priori estimates:** This is a theoretical performance analysis of an algorithm. Efficiency of an algorithm is measured by assuming the external factors.

2. **A Posteriori testing:** This is called performance measurement. In this analysis, actual statistics like running time and required for the algorithm executions are collected.

An estimation of the time and space complexities of an algorithm for varying input sizes is called algorithm analysis.

4.2 Complexity of an Algorithm

Suppose A is an algorithm and n is the size of input data, the time and space used by the algorithm A are the two main factors, which decide the efficiency of A.

Time Factor -Time is measured by counting the number of key operations like comparisons in the sorting algorithm.

Space Factor - Space is measured by the maximum memory space required by the algorithm.

The complexity of an algorithm f (n) gives the running time and/or the storage space required by the algorithm in terms of n as the size of input data.

4.2.1 Time Complexity

The Time complexity of an algorithm is given by the number of steps taken by the algorithm to complete the process.

4.2.2. Space Complexity

Space complexity of an algorithm is the amount of memory required to run to its completion. The space required by an algorithm is equal to the sum of the following two components:

A fixed part is defined as the total space required to store certain data and variables for an algorithm. For example, simple variables and constants used in an algorithm.

A variable part is defined as the total space required by variables, which sizes depends on the problem and its iteration. For example: recursion used to calculate factorial of a given value n.

4.3 Efficiency of an algorithm

Computer resources are limited that should be utilized efficiently. The efficiency of an algorithm is defined as the number of computational resources used by the algorithm. An algorithm must be analyzed to determine its resource usage. The efficiency of an algorithm can be measured based on the usage of different resources.

For maximum efficiency of algorithm we wish to minimize resource usage. The important resources such as time and space complexity cannot be compared directly,

so time and space complexity could be considered for an algorithmic efficiency.

4.3.1 Method for determining Efficiency

The efficiency of an algorithm depends on how efficiently it uses time and memory space.

The time efficiency of an algorithm is measured by different factors. For example, write a program for a defined algorithm, execute it by using any programming language, and measure the total time it takes to run. The execution time that you measure in this case would depend on a number of factors such as:

- Speed of the machine
- Compiler and other system Software tools
- Operating System
- Programming language used
- Volume of data required

However, to determine how efficiently an algorithm solves a given problem, you would like to determine how the execution time is affected by the nature of the algorithm. Therefore, we need to develop fundamental laws that determine the efficiency of a program in terms of the nature of the underlying algorithm.

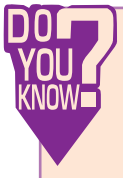
A way of designing algorithm is called algorithmic strategy

4.3.2 Space-Time tradeoff

A space-time or time-memory tradeoff is a way of solving in less time by using more storage space or by solving a given algorithm in very little space by spending more time.

To solve a given programming problem, many different algorithms may be used. Some of these algorithms may be extremely time-efficient and others extremely space-efficient.

Time/space trade off refers to a situation where you can reduce the use of memory at the cost of slower program execution, or reduce the running time at the cost of increased memory usage.



The best algorithm to solve a given problem is one that requires less space in memory and takes less time to execute its instructions to generate output.

4.3.3 Asymptotic Notations

Asymptotic Notations are languages that uses meaningful statements about time and space complexity. The following three asymptotic notations are mostly used to represent time complexity of algorithms:

(i) Big O

Big O is often used to describe the worst-case of an algorithm.

(ii) Big Ω

Big Omega is the reverse Big O, if Bi O is used to describe the upper bound (worst - case) of a asymptotic function, Big Omega is used to describe the lower bound (best-case).

(iii) Big Θ

When an algorithm has a complexity with lower bound = upper bound, say that an algorithm has a complexity O (n log n) and Ω (n log n), it's actually has the complexity Θ (n log n), which means the running time of that algorithm always falls in n log n in the best-case and worst-case.

4.3.4 Best, Worst, and Average ease Efficiency

Let us assume a list of n number of values stored in an array. Suppose if we want to search a particular element in this list, the algorithm that search the key element in the list among n elements, by comparing the key element with each element in the list sequentially.

The best case would be if the first element in the list matches with the key element to be searched in a list of elements. The efficiency in that case would be expressed as O(1) because only one comparison is enough.

Similarly, the worst case in this scenario would be if the complete list is searched and the element is found only at the end of the list or is not found in the list. The efficiency of an algorithm in that case would be expressed as O(n) because n comparisons required to complete the search.

The average case efficiency of an algorithm can be obtained by finding the average number of comparisons as given below:

Minimum number of comparisons = 1

Maximum number of comparisons = n

If the element not found then maximum number of comparison = n

Therefore, average number of comparisons = (n + 1)/2

Hence the average case efficiency will be expressed as O (n).

4.4 Algorithm for Searching Techniques

4.4.1 Linear Search

Linear search also called sequential search is a sequential method for finding a particular value in a list. This method checks the search element with each element in sequence until the desired element is found or the list is exhausted. In this searching algorithm, list need not be ordered.

Pseudo code

1. Traverse the array using for loop
2. In every iteration, compare the target search key value with the current value of the list.

- If the values match, display the current index and value of the array
- If the values do not match, move on to the next array element.

3. If no match is found, display the search element not found.

To search the number 25 in the array given below, linear search will go step by step in a sequential order starting from the first element in the given array if the search element is found that index is returned otherwise the search is continued till the last index of the array. In this example number 25 is found at index number 3.

index	0	1	2	3	4
values	10	12	20	25	30

Example 1:

Input: values[] = {5, 34, 65, 12, 77, 35}
target = 77
Output: 4

Example 2:

Input: values[] = {101, 392, 1, 54, 32, 22, 90, 93}
target = 200
Output: -1 (not found)

4.4.2. Binary Search

Binary search also called half-interval search algorithm. It finds the position of a search element within a sorted array. The binary search algorithm can be done as divide-and-conquer search algorithm and executes in logarithmic time.

Pseudo code for Binary search

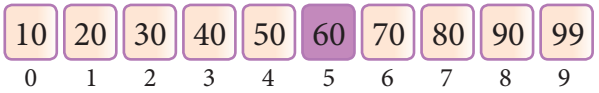
1. Start with the middle element:

- If the search element is equal to the middle element of the array i.e., the middle value = number of elements in array/2, then return the index of the middle element.
- If not, then compare the middle element with the search value,
- If the search element is greater than the number in the middle index, then select the elements to the right side of the middle index, and go to Step-1.
- If the search element is less than the number in the middle index, then select the elements to the left side of the middle index, and start with Step-1.

2. When a match is found, display success message with the index of the element matched.
3. If no match is found for all comparisons, then display unsuccessful message.

Binary Search Working principles

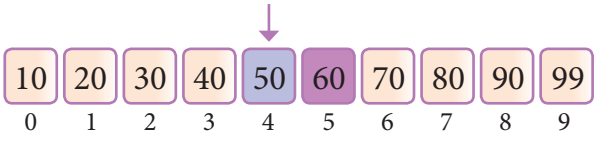
List of elements in an array must be sorted first for Binary search. The following example describes the step by step operation of binary search. Consider the following array of elemnts, the array is being sorted so it enables to do the binary search algorithm. Let us assume that the search element is 60 and we need to search the location or index of search element 60 using binary search.



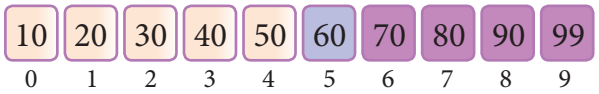
First, we find index of middle element of the array by using this formula :

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0) / 2 = 4 (fractional part ignored). So, 4 is the mid value of the array.



Now compare the search element with the value stored at mid value location 4. The value stored at location or index 4 is 50, which is not match with search element. As the search value 60 is greater than 50.

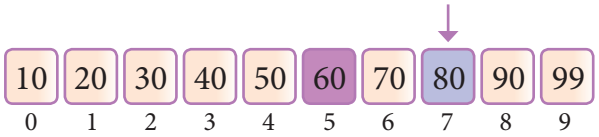


Now we change our low to mid + 1 and find the new mid value again using the formula.

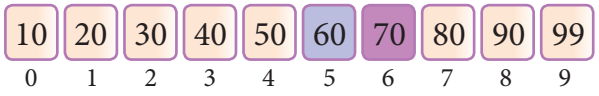
low = mid + 1

mid = low + (high - low) / 2

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



The value stored at location or index 7 is not a match with search element, rather it is more than what we are looking for. So, the search element must be in the lower part from the current mid value location

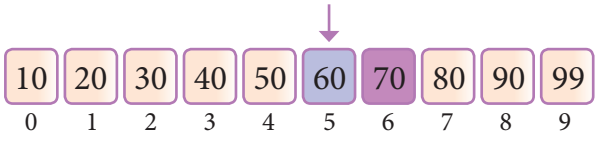


The search element still not found. Hence, we calculated the mid again by using the formula.

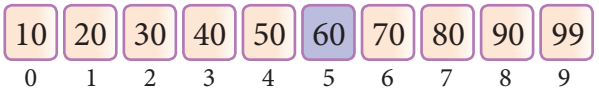
high = mid -1

mid = low + (high - low)/2

Now the mid value is 5.



Now we compare the value stored at location 5 with our search element. We found that it is a match.



We can conclude that the search element 60 is found at lcoation or index 5. For example if we take the search element as 95, For this value this binary search algorithm return unsuccessful result.

4.5 Sorting Techniques

4.5.1 Bubble sort algorithm

Bubble sort is a simple sorting algorithm. The algorithm starts at the beginning of the list of values stored in an array. It compares each pair of adjacent elements and swaps them if they are in the unsorted order. This comparison and passed to be continued until no swaps are needed, which indicates that the list of values stored in an array is sorted. The algorithm is a comparison sort, is named for the way smaller elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and less efficient when compared to insertion sort and other sorting methods.

Assume list is an array of n elements. The swap function swaps the values of the given array elements.

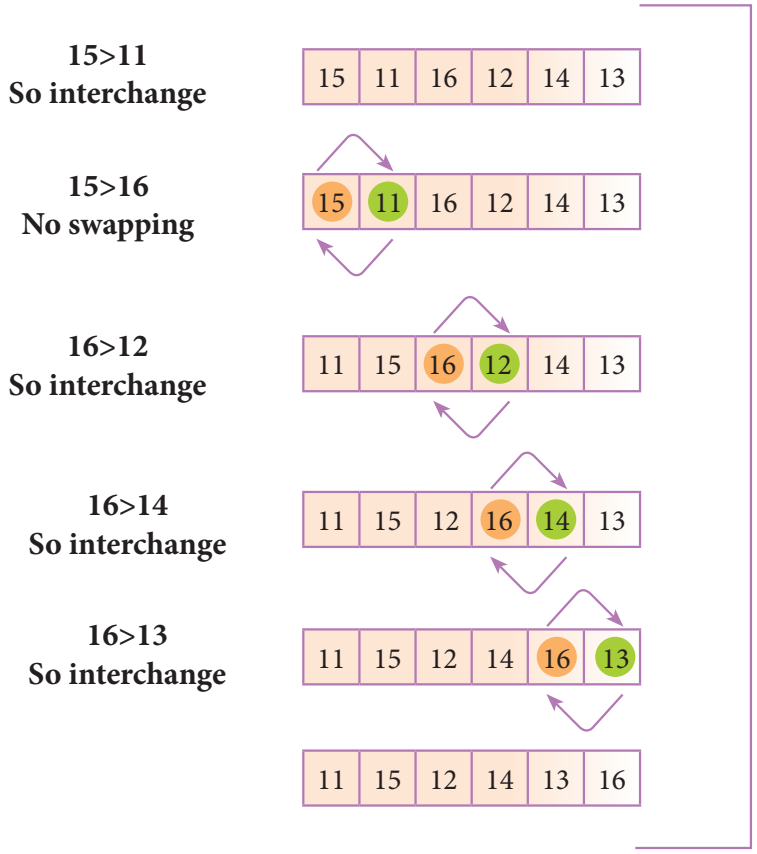
Pseudo code

1. Start with the first element i.e., index = 0, compare the current element with the next element of the array.

2. If the current element is greater than the next element of the array, swap them.

3. If the current element is less than the next or right side of the element, move to the next element. Go to Step 1 and repeat until end of the index is reached.

Let's consider an array with values {15, 11, 16, 12, 14, 13} Below, we have a pictorial representation of how bubble sort will sort the given array.



The above pictorial example is for iteration-1. Similarly, remaining iteration can be done. The final iteration will give the sorted array.

At the end of all the iterations we will get the sorted values in an array as given below:

11	12	13	14	15	16

4.5.2 Selection sort

The selection sort is a simple sorting algorithm that improves on the performance of bubble sort by making only one exchange for every pass through the list. This algorithm will first find the smallest elements in array and swap it with the element in the first position of an array, then it will find the second smallest element and swap that element with the element in the second position, and it will continue until the entire array is sorted in respective order.

This algorithm repeatedly selects the next-smallest element and swaps in into the right place for every pass. Hence it is called selection sort.

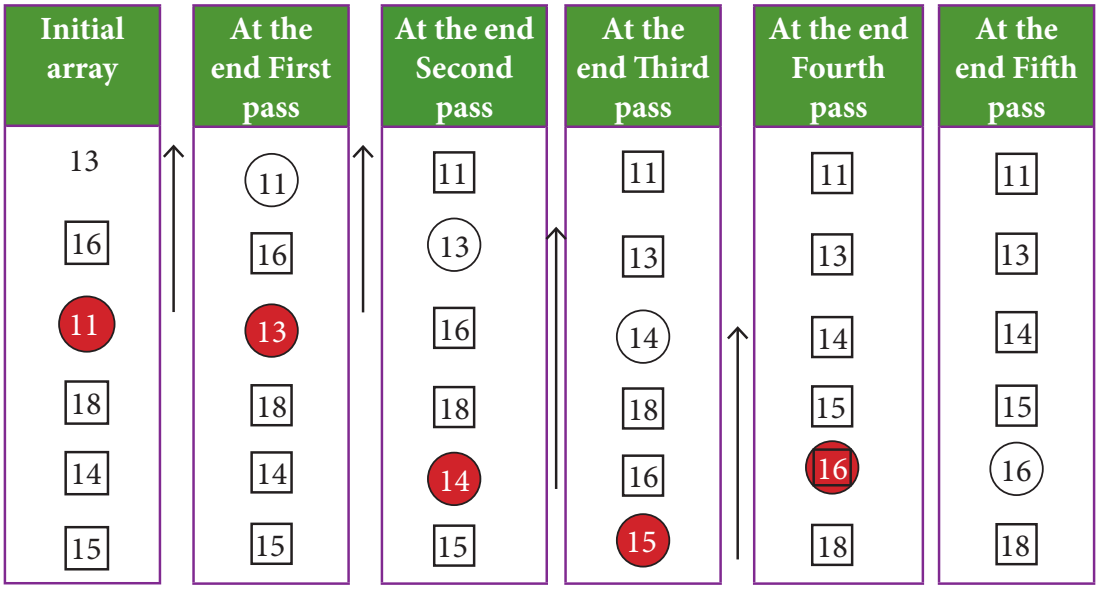
Pseudo code

1. Start from the first element i.e., index-0, we search the smallest element in the array, and replace it with the element in the first position.

2. Now we move on to the second element position, and look for smallest element present in the sub-array, from starting index to till the last index of sub - array.
3. Now replace the second smallest identified in step-2 at the second position in the or original array, or also called first position in the sub array.
4. This is repeated, until the array is completely sorted.

Let's consider an array with values {13, 16, 11, 18, 14, 15}

Below, we have a pictorial representation of how selection sort will sort the given array.



In the first pass, the smallest element will be 11, so it will be placed at the first position.

After that, next smallest element will be searched from an array. Now we will get 13 as the smallest, so it will be then placed at the second position.

Then leaving the first element, next smallest element will be searched, from the remaining elements. We will get 13 as the smallest, so it will be then placed at the second position.

Then leaving 11 and 13 because they are at the correct position, we will search for the next smallest element from the rest of the elements and put it at third position and keep doing this until array is sorted.

Finally we will get the sorted array end of the pass as shown above diagram.

4.5.3 Insertion sort

Insertion sort is a simple sorting algorithm. It works by taking elements from the list one by one and inserting then in their correct position in to a new sorted list. This algorithm builds the final sorted array at the end. This algorithm uses n-1 number of passes to get the final sorted list as per the pervious algorithm as we have discussed.

Pseudo for Insertion sort

Step 1 – If it is the first element, it is already sorted.

Step 2 – Pick next element

Step 3 – Compare with all elements in the sorted sub-list

Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted

Step 5 – Insert the value

Step 6 – Repeat until list is sorted

44	16	83	07	67	21	34	45	10	Assume 44 is a sorted list of 1 item
16	44	83	07	67	21	34	45	10	inserted 16
16	44	83	07	67	21	34	45	10	inserted 83
07	16	44	83	67	21	34	45	10	inserted 07
07	16	44	67	83	21	34	45	10	inserted 67
07	16	21	44	67	83	34	45	10	inserted 21
07	16	21	34	44	67	83	45	10	inserted 34
07	16	21	34	44	45	67	83	10	inserted 45
07	10	16	21	34	44	45	67	83	inserted 10

At the end of the pass the insertion sort algorithm gives the sorted output in ascending order as shown below:

07	10	16	21	34	44	45	67	83

4.6. Dynamic programming

Dynamic programming is an algorithmic design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions. Dynamic programming approach is similar to divide and conquer. The given problem is divided into smaller and yet smaller possible sub-problems.

Dynamic programming is used whenever problems can be divided into similar sub-problems. so that their results can be re-used to complete the process. Dynamic programming approaches are used to find the solution in optimized way. For every inner sub problem, dynamic algorithm will try to check the results of the previously solved sub-problems. The solutions of overlapped sub-problems are combined in order to get the better solution.

Steps to do Dynamic programming

- The given problem will be divided into smaller overlapping sub-problems.
- An optimum solution for the given problem can be achieved by using result of smaller sub-problem.
- Dynamic algorithms uses Memoization.

Note

Memoization or memoisation is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

4.6.1 Fibonacci Series – An example

Fibonacci series generates the subsequent number by adding two previous numbers. Fibonacci series starts from two numbers – Fib 0 & Fib 1. The initial values of Fib 0 & Fib 1 can be taken as 0 and 1.

Fibonacci series satisfies the following conditions :

$Fibn = Fib_{n-1} + Fib_{n-2}$

Hence, a Fibonacci series for the n value 8 can look like this

Fib_8 = 0 1 1 2 3 5 8 13

4.6.2 Fibonacci Iterative Algorithm with Dynamic programming approach

The following example shows a simple Dynamic programming approach for the generation of Fibonacci series.

Initialize f0=0, f1 =1

step-1: Print the initial values of Fibonacci f0 and f1

step-2: Calculate fibanocci fib ← f0 + f1

step-3: Assign f0← f1, f1← fib

step-4: Print the next consecutive value of fibanocci fib

step-5: Goto step-2 and repeat until the specified number of terms generated

For example if we generate fibobnacci series upto 10 digits, the algorithm will generate the series as shown below:

The Fibonacci series is : 0 1 1 2 3 5 8 13 21 34 55

Points to remember:

- An algorithm is a finite set of instructions to accomplish a particular task.
- Algorithm consists of step-step-by instructions that are required to accomplish a task and helps the programmer to develop the program.
- Program is an expression of algorithm in a programming language.
- Algorithm analysis deals with the execution or running time of various operations involved.
- Space complexity of an algorithm is the amount of memory required to run to its completion.
- Big Oh is often used to describe the worst-case of an algorithm.
- Big Omega is used to describe the lower bound which is best way to solve the space complexity.
- The Time complexity of an algorithm is given by the number of steps taken by the algorithm to complete the process.
- The efficiency of an algorithm is defined as the number of computational resources used by the algorithm.

Points to remember:

- A way of designing algorithm is called algorithmic strategy.
- A space-time or time-memory tradeoff is a way of solving a problem or calculation in less time by using more storage space.
- Asymptotic Notations are languages that uses meaningful statements about time and space complexity.
- Bubble sort is a simple sorting algorithm. It compares each pair of adjacent items and swaps them if they are in the unsorted order.
- The selection sort improves on the bubble sort by making only one exchange for every pass through the list.
- Insertion sort is a simple sorting algorithm that builds the final sorted array or list one item at a time. It always maintains a sorted sublist in the lower positions of the list.
- Dynamic programming is used when the solutions to a problem can be viewed as the result of a sequence of decisions.

Evaluation

9DJFRN

Part - I

Choose the best answer: **(1 Marks)**

1. The word comes from the name of a Persian mathematician Abu Ja'far Mohammed ibn-i Musa al Khowarizmi is called?

   (A) Flowchart (B) Flow (C) Algorithm (D) Syntax

2. From the following sorting algorithms which algorithm needs the minimum number of swaps?

   (A) Bubble sort (B) Quick sort (C) Merge sort (D) Selection sort

3. Two main measures for the efficiency of an algorithm are

   (A) Processor and memory (B) Complexity and capacity

   (C) Time and space (D) Data and space

4. The complexity of linear search algorithm is

   (A) O(n) (B) O(log n) (C) O(n2) (D) O(n log n)

5. From the following sorting algorithms which has the lowest worst case complexity?

   (A) Bubble sort (B) Quick sort (C) Merge sort (D) Selection sort

6. Which of the following is not a stable sorting algorithm?

(A) Insertion sort (B) Selection sort (C) Bubble sort (D) Merge sort

7. Time complexity of bubble sort in best case is

(A) θ (n) (B) θ (nlogn) (C) θ (n2) (D) θ (n(logn) 2)

8. The Θ notation in asymptotic evaluation represents

(A) Base case (B) Average case (C) Worst case (D) NULL case

9. If a problem can be broken into subproblems which are reused several times, the problem possesses which property?

(A) Overlapping subproblems (B) Optimal substructure

(C) Memoization (D) Greedy

10. In dynamic programming, the technique of storing the previously calculated values is called ?

(A) Saving value property (B) Storing value property

(C) Memoization (D) Mapping

Part - II

Answer the following questions **(2 Marks)**

1. What is an Algorithm?
2. Define Pseudo code.
3. Who is an Algorist?
4. What is Sorting?
5. What is searching? Write its types.

Part - III

Answer the following questions **(3 Marks)**

1. List the characteristics of an algorithm.
2. Discuss about Algorithmic complexity and its types.
3. What are the factors that influence time and space complexity.
4. Write a note on Asymptotic notation.
5. What do you understand by Dynamic programming?

Part - IV

Answer the following questions **(5Marks)**

1. Explain the characteristics of an algorithm.
2. Discuss about Linear search algorithm.
3. What is Binary search? Discuss with example.
4. Explain the Bubble sort algorithm with example.
5. Explain the concept of Dynamic programming with suitable example.

Reference Books

1. *Fundamentals Computer Algorithms, Ellis Horowitz, Sartaj Sahni, Sanguthevar, Rajasekaran, Second Edition, University press (India) Limited, 2013.*
2. *Design and Analysis of Algorithms, S. Sridhar, Oxford University Press, 2015*

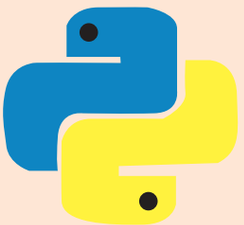
Web References

www.wickipedia.org

CASE STUDY/ STUDENT'S ACTIVITY

1. Create an algorithm for grading systems of your class student's Quarterly examination marks by satisfying all necessary conditions.

Why Python in standard XII curriculum?



This is the question that is fretting the minds of teachers and students.

The present book is organized in such a way that even a novice reader can grasp and work on python programming.

Testimonies

- "Python has been an important part of Google since the beginning and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language." -- **Peter Norvig, director of search quality at Google, Inc.**
- "Python is fast enough for our site and allows us to produce maintainable features in record times, with a minimum of developers," -- **Cuong Do, Software Architect, YouTube.com**

Python's popularity has seen a steady and unflagging growth over the recent years. Today, familiarity with Python is an advantage for every programmer, as Python has infiltrated every niche and has useful roles to play in any software solution.

Python has experienced an impressive growth as compared to the other languages. The **IEEE Spectrum magazine** published by **the Institute of Electrical & Electronics Engineers, New York**, ranks Python as the top language for 2018, for the second consecutive year.

The above statistical data has maintained its grip with Python scoring 100 and C++ language stands second nipping at its heels with a 99.7 score.

Python being popular is used by a number of tech giants like **Google, Instagram, Pinterest, Yahoo, Disney, IBM, Nokia etc.**

1. Python	100.0
2. C++	99.7
3. Java	97.5
4. C	96.7
5. C#	89.4
6. PHP	84.9
7. R	82.8
8. JavaScript	82.6
9. Go	76.4
10. Assembly	74.1

Many businesses are advised to choose Python for the following reasons:-

- Easy syntax and readability
- High level scripting language with oops
- famous for enormous functions, add-on modules, libraries, frameworks and tool-kits.
- Built-in functions supports scientific computing.

With the advent of computers, there have been significant changes in the way we work in almost all the fields. The computerization has helped to improve productivity and accelerate decision making in every organization. Even for individuals, be it engineers, doctors, chartered accountants or homemakers, the style of working has changed drastically. So as we go, let us accept the change and move towards a brighter day ahead.