5 Program Efficiency

5.1 INTRODUCTION

The most important thing to think about while designing and implementing a program is that it should produce results that can be relied upon. We want our bank balances to be calculated correctly. We want the fuel injectors in our automobiles to inject appropriate amount of fuel. We would prefer that neither airplanes nor operating systems crash.

Sometimes performance is an important aspect of correctness. This is most importantly observed in real-time applications where prompt action is required from the program's end. A program that warns airplanes of potential obstructions needs to issue a warning before an obstruction is encountered. Performance can also affect the utility of many non-real-time programs. The number of transactions completed per minute is an important factor to be taken into account while evaluating the utility of database systems. Users care about the time required to start an application on their phone, which should be instant.

Writing efficient programs is not easy. The most straightforward solution is often not the most efficient. Computationally-efficient algorithms often implement the logics and concepts that are usually difficult to understand. Consequently, programmers often increase the conceptual complexity of a program in an effort to reduce its computational complexity.

A good and efficient program starts with an efficient and simple algorithm. We have already discussed Algorithms in Class XI. We know that an algorithm is a step-by-step method of solving a problem. It is also defined as a sequence of instructions written in an English-like language. An efficient algorithm lays the foundation for an efficient program. Thus, it requires a balance to be maintained between the computational and conceptual complexity of a program, which has been addressed in this chapter.

> **Learning Tip:** A step is an operation that takes a fixed amount of time, such as binding a variable to an object, making a comparison, executing an arithmetic operation, or accessing an object in memory.

5.2 WHAT IS ALGORITHM ANALYSIS

It is very common for the beginners in Computer Science to compare their programs with each other. You must have noticed that computer programs, especially the simple ones, look similar. Now, an interesting question arises: When two programs solve the same problem but look different, is one program better than the other?

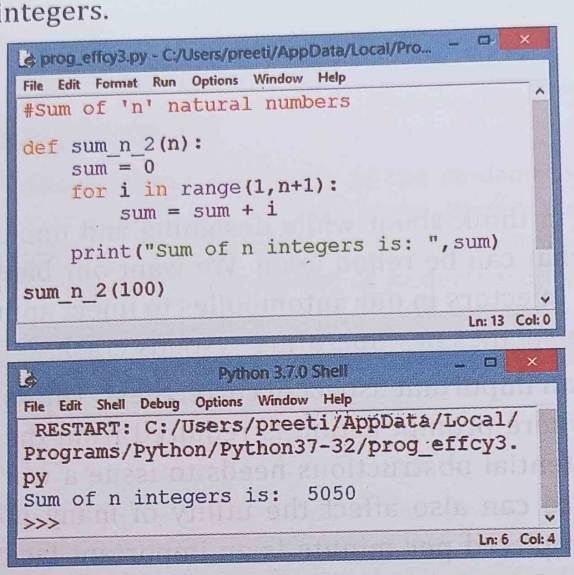
In order to answer this question, we need to remember that there is an important difference between a program and the underlying algorithm that the program is representing. As mentioned above, an algorithm is a method of solving any instance of the problem in such a way that, given

a particular input, the algorithm produces the desired result. A program, on the other hand, is an algorithm that has been encoded into some programming language. There may be multiple programs for the same algorithm, depending on the programmer and the programming language used.

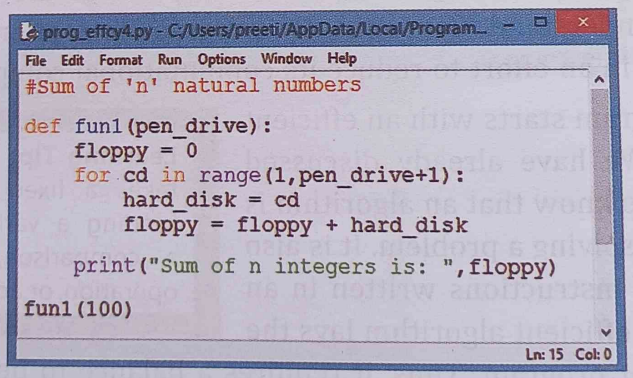
To explore this difference further, consider the function shown below. This function solves a familiar problem—computing the sum of the first '*n*' integers. The algorithm uses the idea of a variable 'sum' that is initialized to 0. The solution then iterates through the *n* integers, adding each to the value of this variable 'sum'.

Practical Implementation–1

To compute the sum of 'n' integers.



In the above example, we are calculating the sum of 'n' inputted integers, for n=100 in this case. With each iteration, the value of counter-variable 'i' is added to the sum and printed in the end. The same task can be performed through another program given below. Consider another function **fun1()**.



At first glance, the above function may look strange but, upon further inspection, you can see that this function is essentially doing the same thing as the previous one. The reason is not poor coding. We did not use good **identifier names** to **assist** with **readability**, and we used an extra assignment statement—hard_disk = cd—a step that was not really necessary.

Now, again the question arises whether one function is better than the other. The answer depends on your criteria. The function **sum_n_2 ()** is certainly better than the function **fun1()** if you are concerned with readability. The motive behind this explanation is to help you write programs that are easy to read and understand. In the above program, the identifier names that have been

used are quite weird, such as pen_drive, cd, floppy, etc., that seem to be nowhere associated with the program and create confusion in the minds of the readers/users. This hampers the overall execution and implementation efficiency of a program. Therefore, the first consideration to be kept in mind is the correct and appropriate use of identifier names. They should be proper and content-specific.

Algorithm analysis is concerned with comparing algorithms based upon the amount of computing resources that each algorithm uses. We consider two algorithms and say that one is better than the other because it is more efficient in its use of those resources or we simply say it uses fewer resources. From this perspective, the above two functions seem very similar. They both use essentially the same algorithm to solve the summation problem.

At this point, it is important to think more about what we really mean by computing resources.

5.3 WHAT IS COMPUTATIONAL COMPLEXITY

Computational Complexity consists of two words—computation and complexity.

Computation means to solve problems using an algorithm whereas complexity involves the study of factors to determine how much resource is sufficiently necessary for this algorithm to run efficiently (performance).

Resources generally include time and space:

- Time to run the algorithm (temporal complexity)
- The space (Memory) needed to run the algorithm (Space Complexity)

Determining the efficiency and effectiveness of the algorithm is known as complexity. It is also observed that an algorithm is giving maximum output in minimum time. All these concepts are inter-related.

Effectiveness means that the algorithm carries out its intended function correctly. On the other hand, efficiency means that the algorithm should be correct with the best possible performance. To measure efficiency, it is required to determine complexity.

Calculating the complexity of different algorithms involves mathematical calculations and detailed analysis, which is beyond the scope of this textbook. However, we will discuss some basics of complexity to get some ideas.

The following tips will guide us in estimating the time complexity of an algorithm:

1. Any algorithm that does not have any loop will have time complexity as 1 since the number of instructions to be executed will be constant, irrespective of the data size. Such algorithms are known as Constant time algorithms.
2. Any algorithm that has a loop (usually 1 to n) will have time complexity as n because the loop will execute the statement inside its body n number of times. Such algorithms are known as Linear time algorithms.
3. A loop within a loop (nested loop) will have time complexity as n^2. Such algorithms are known as Quadratic time algorithms. If there is a nested loop and also a single loop, time complexity will be estimated on the basis of the nested loop only.

How to Evaluate Efficiency of Programs

Efficient programming is a manner of programming that, when the program is executed, uses a low amount of overall resources, especially pertaining to computer hardware. A program is designed by a human being and different human beings may use different algorithms or sequences of codes to perform particular tasks. So, the efficiency of such programs varies depending upon the number of resources being used. Thus, efficiency refers to the quality of algorithm, *i.e.*, code must complete the task in the shortest possible time using minimum resources. It is important to measure efficiency of the algorithm before applying it on a large scale, *i.e.*, on bulk of data.

Nowadays, most of the applications are online where prompt response is required. If efficiency of the algorithm is not checked, the site will crash and the organization may lose their customers/ business because of the slow speed.

Practising to create a low-size (number of lines of codes) and low-resource algorithm results in an efficient program.

- **Wall clock time/elapsed time:** It is defined as the time taken to complete a task as seen by the user. The wall clock time constitutes all the overheads, *e.g.*, operating system overheads, potentially interfering with other applications, etc.
- **CPU time:** It does not include time slices introduced by external sources (*e.g.*, running other applications).

Calculating Performance

Performance of a program is inversely proportional to the wall clock time. So, to maximize the performance of a program, the execution time should be minimum.

$$\text{Performance}_x = \frac{1}{\text{execution\_time}_x}$$

If "X is n times faster than Y", then the execution time on Y is n times longer than that on X.

$$\frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Executiontime}_y}{\text{Executiontime}_x} = n$$

For example,

If a particular desktop runs a program in 60 seconds and a laptop runs the same program in 90 seconds, how much faster is the desktop than the laptop?

Ans. Efficiency = Performancedesktop/Performancelaptop

= (1/60)/(1/90) = 1.5

So, the desktop is 1.5 times faster than the laptop.

The efficiency of a program can be evaluated on the basis of two major factors:

- In terms of time
- On the basis of number of operations

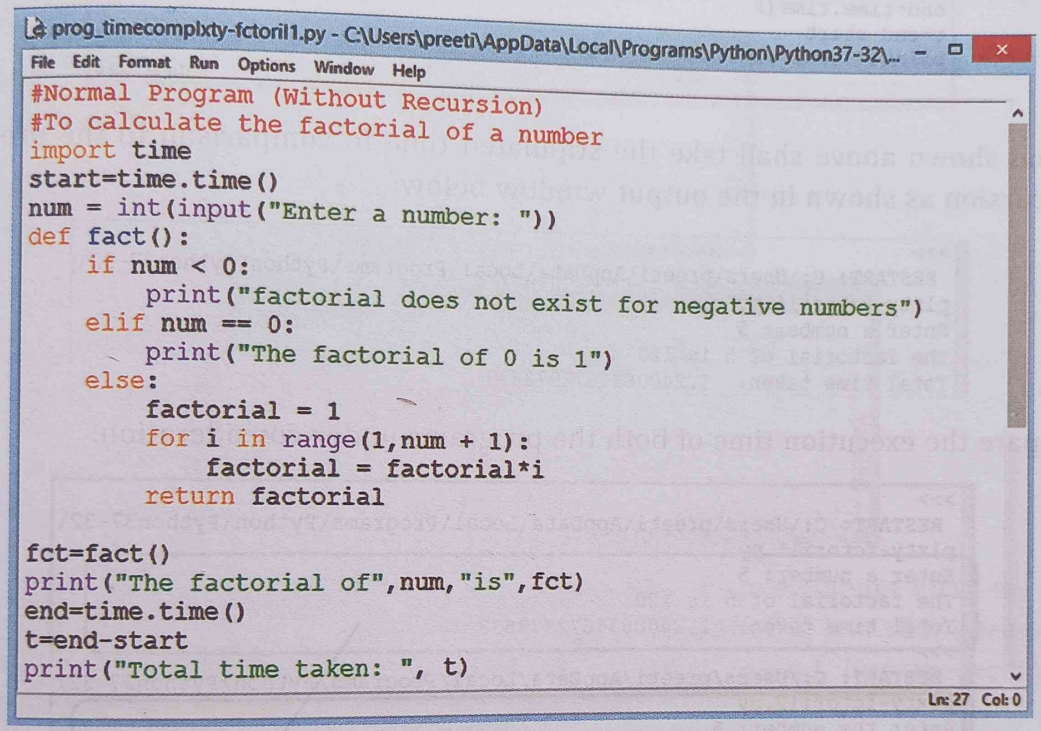
5.4 PROGRAM EFFICIENCY IN TERMS OF TIME

When we speak about program efficiency, a major factor that defines the overall efficiency of a program is the amount of space or memory (known as space complexity) an algorithm requires to solve a problem.

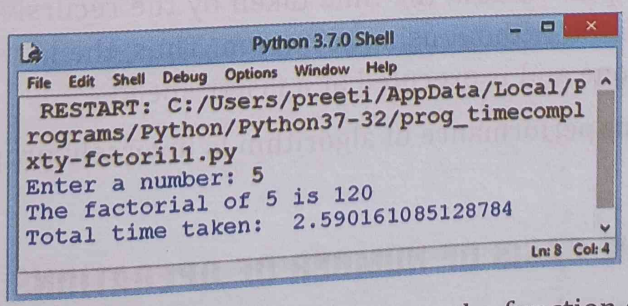
Another resource alternative to space requirements is the amount of time required to execute the algorithms, known as **time complexity**. This measure is also referred to as the "**execution time**" or "**running time**" of the algorithm. One way to measure the execution time for the function **fact()** (to calculate the factorial of an inputted number) is to do a benchmark analysis. This means that we will track the actual time required for the program to compute its result. In Python, we can trace back a function by taking into account the starting time and ending time with respect to the system we are using. In the **time** module, there is a function called **time()** that will return the current system clock time in seconds.

Practical Implementation–2

Illustrate programming efficiency of the program to calculate factorial of an inputted number on the basis of execution time, *i.e.*, using the function time().



In the above program, time library is instantiated by importing it using the statement import time. By calling time() function twice, at the beginning and at the end, and then computing the difference, we can get an exact number of seconds (fractions in most cases) for execution. The function displays the result and the amount of time (in seconds) required for the calculation of the factorial for the inputted number as shown below:



Now the same program is done recursively, which means the function shall keep on calling itself again and again and the time taken to execute this program is considerably less in comparison to the previous program implemented without using recursion.

```
prog_timecomplxty-fctoril2.py - C:/Users/preeti/AppData/Local/Programs/Python/Pytho...
File Edit Format Run Options Window Help
#Recursive Program
#To calculate the factorial of a number

import time
def factorial(n):
    if n == 1:
        return n
    else:
        return n*factorial(n-1)
start=time.time()
num=int(input("enter the number: "))
if num < 0:
    print("factorial does not exist for negative numbers")
elif num==0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of ",num," is ",factorial(num))
end=time.time()
t=end-start
print("Total time taken: ",t)
                                                    Ln: 22 Col: 0
```

The program shown above shall take the stipulated time in comparison to the program done without recursion as shown in the output window below:

```
>>>
 RESTART: C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\
plxty-fctoril1.py
Enter a number: 5
The factorial of 5 is 120
Total time taken:  1.2400636672973633
```

Let us compare the execution time of both the programs under consideration.

```
>>>
 RESTART: C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\
plxty-fctoril1.py
Enter a number: 5
The factorial of 5 is 120
Total time taken:  1.2400636672973633
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/
plxty-fctoril2.py
enter the number: 5
The factorial of  5  is  120
Total time taken:  2.590161085128784
```

Time taken by the factorial program without using recursion

Time taken by the factorial program using recursion

As is evident from the output window, the time taken by the recursive function is considerably less than the program written without using recursion. Thus, the latter program becomes more efficient than the former one and is executed at a faster speed.

Thus, we can conclude that performance of algorithm is inversely proportional to the wall clock time.

5.5 EFFICIENCY ON THE BASIS OF NUMBER OF OPERATIONS

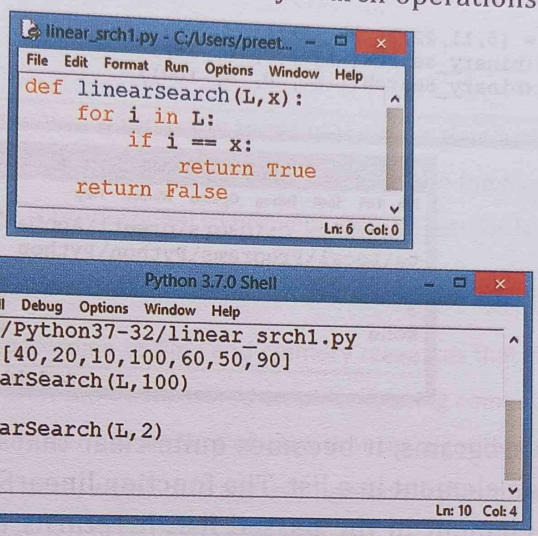
Although we do not see this in the summation example, sometimes the performance of an algorithm depends on the exact values of the data and also on the number of operations performed to carry out a particular task or in the execution of an algorithm, rather than on simply the size of the problem. For these kinds of algorithms, we need to characterize their performance in terms of

best-case, worst-case, or average-case performance. The **worst-case** performance refers to a particular data set where the algorithm performs poorly whereas a different data set for the same algorithm might have extraordinarily good performance. However, in most cases, the algorithm performs somewhere in between these two extremes (average case). Thus, performance-based algorithms are dependent upon the number of operations that are carried out by an algorithm and the time taken by the program to carry out these operations.

To compare the efficiency of the algorithms on the basis of number of operations, we shall consider the example of searching an element in a list, *i.e.*, linear search as well as binary search, and find out which is more efficient.

Practical Implementation–3

To compute the efficiency of linear search and binary search operations on a linear list.



In respect of the above program, we have a list containing seven elements. First, we will perform search operation using linearSearch(L, 100). If the first element in L is 100, linearSearch() will return true almost immediately. On the other hand, if 100 is not in L, linearSearch() will have to examine all seven elements before returning false.

In general, there are three broad cases to think about:

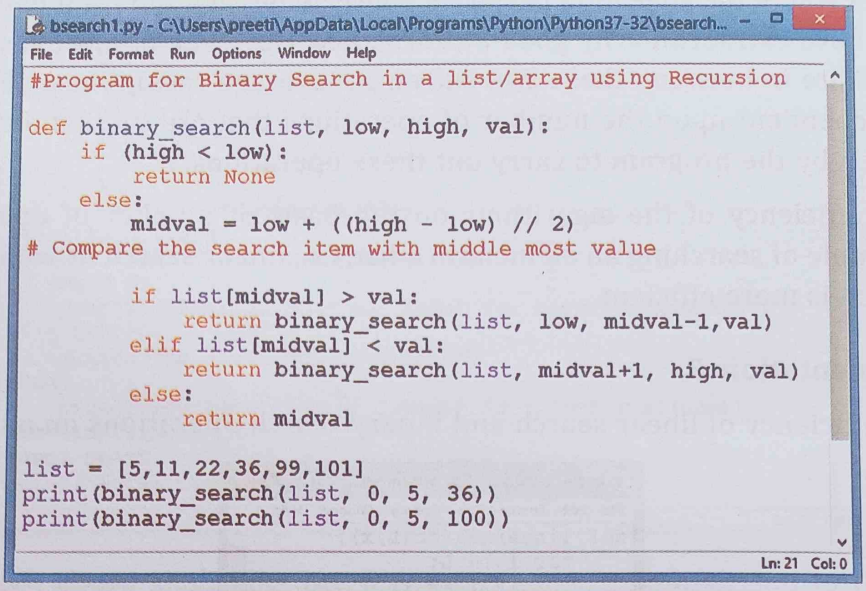
The **best-case** running time is the running time of the algorithm when the inputs are as favourable as possible, *i.e.*, the best-case running time is the minimum running time over all the possible inputs of a given size. For linear search, the best-case running time is independent of the size of L.

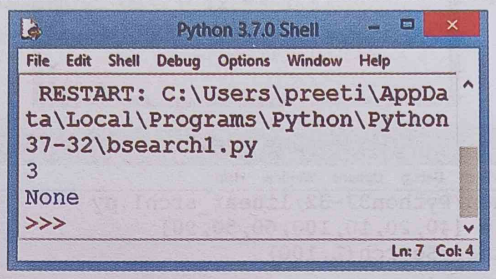
Similarly, the **worst-case** running time is the maximum running time over all the possible inputs of a given size. For linear search, the worst-case running time is linear in the size of the list.

By analogy, with the definitions of the best-case and worst-case running time, the **average-case** (also called expected-case) running time is the average running time over all possible inputs of a given size.

People usually focus on the worst case. The worst case provides an upper bound (maximum value) on the running time. This is critical in situations where there is a time constraint on how long a computation can take. It is not good enough to know that "most of the time" the air traffic control system warns of impending collisions before they occur.

Now, consider the program for Binary Search using recursion.





On comparing both the programs, it becomes quite clear that binary search method is a better approach for searching an element in a list. The function linearSearch() receives two parameters. It searches for a given element in the passed list. It returns true if the element is found, else returns false. The linear search technique will prove worst if the element to be searched is one of the last elements in the list as so many comparisons would take place and the entire process will be time-consuming with several of the operations carried out to determine whether the search is successful or not.

To save on the number of comparisons or operations performed, binary search is very useful. This technique searches an element in minimum possible comparisons. Its only requirement is that the list in which the element is to be searched should be a sorted list. The element to be searched is always compared with the middle element of the entire list. If the value to be searched is more than the middle element, the latter part of the segment becomes the new segment to be looked into. If the value is less than the middle element, then the first segment of the list is taken as the new segment and is looked into. This results in the breakdown of the entire list into half the number of elements in the first instance itself. Thus, the number of comparisons to be made is half of what are carried out in the case of linear search with the same number of elements in the list.

Therefore, we say that the program for binary search is far more efficient than linear search in terms of the number of computations or comparisons to be made. This is also defined as **Code Optimization**, which is a way of enhancing code productivity and quality. It is to be achieved by reducing the code size and memory in order to execute statements quickly. In a nutshell, the efficiency of a program depends a lot on the number of operations to be carried out to execute a particular task. Fewer operations means higher efficiency.

In order to increase the efficiency of a Python program, the following points are to be kept in mind:

1. Keep the code as compact and simple as possible.
2. Looping constructs should be kept to the bare minimum.
3. Recursive operations are more efficient than sequential programs with a large number of instructions to be executed.
4. Correctness and robustness of the program should be considered first.
5. Proper identifier names should be used to increase the readability of a program.
6. Best-case, worst-case and average-case should be taken into consideration before writing a program/algorithm for a task.
7. Use of built-in functions should be preferred for imparting code reusability.

MEMORY BYTES

- Analysis of algorithms: A way to compare algorithms in terms of their run-time and/or memory requirements.
- Worst case: The input that makes a given algorithm run slowest (or require the most space or time).
- Linear: An algorithm whose run-time is proportional to problem size, at least for large problem sizes.
- Search: The problem of locating an element from a collection (like a list or dictionary) or determining whether it is present or not.
- Different algorithms can be ranked according to time and memory resources that they require.
- Running/execution time of an algorithm can be measured empirically using computer's clock time.
- Counting instructions provide measurement of amount of work that the algorithm does.
- An algorithm can have different best-case, worst-case and average-case behaviours.
- Binary search is faster than linear search (but data must be ordered).
- Binary search looks for the given item in a sorted list. The search segment reduces to half at every successive stage.
- In linear search, each element of the array is compared with the given item to be searched one by one.
- Complexity analysis is performed to explain how an algorithm will perform when the input size increases.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.
   (a) An efficient ..... lays the foundation for an efficient program.
   (b) An ..... is a step-by-step method of solving a problem.
   (c) A ..... is an algorithm encoded into some programming language.
   (d) ..... is concerned with comparing algorithms based on the amount of computing resources used.
   (e) Efficiency of a program can be evaluated on the basis of ..... and ..... .
   (f) The overall efficiency of a program depends on the amount of ..... an algorithm requires to solve a program.
   (g) ..... is inversely proportional to the wall clock time.
   (h) Classification of algorithm is made on the basis of ....., ... or ..... performance.
   (i) For linear search, the worst-case running time is ..... in the size of the list.
   (j) ..... is a measure of efficiency of algorithm.
   (k) ..... is a way to enhance code productivity and quality.

In order to increase the efficiency of a Python program, the following points are to be kept in mind:

1. Keep the code as compact and simple as possible.
2. Looping constructs should be kept to the bare minimum.
3. Recursive operations are more efficient than sequential programs with a large number of instructions to be executed.
4. Correctness and robustness of the program should be considered first.
5. Proper identifier names should be used to increase the readability of a program.
6. Best-case, worst-case and average-case should be taken into consideration before writing a program/algorithm for a task.
7. Use of built-in functions should be preferred for imparting code reusability.

MEMORY BYTES

- Analysis of algorithms: A way to compare algorithms in terms of their run-time and/or memory requirements.
- Worst case: The input that makes a given algorithm run slowest (or require the most space or time).
- Linear: An algorithm whose run-time is proportional to problem size, at least for large problem sizes.
- Search: The problem of locating an element from a collection (like a list or dictionary) or determining whether it is present or not.
- Different algorithms can be ranked according to time and memory resources that they require.
- Running/execution time of an algorithm can be measured empirically using computer's clock time.
- Counting instructions provide measurement of amount of work that the algorithm does.
- An algorithm can have different best-case, worst-case and average-case behaviours.
- Binary search is faster than linear search (but data must be ordered).
- Binary search looks for the given item in a sorted list. The search segment reduces to half at every successive stage.
- In linear search, each element of the array is compared with the given item to be searched one by one.
- Complexity analysis is performed to explain how an algorithm will perform when the input size increases.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.
   (a) An efficient ..... lays the foundation for an efficient program.
   (b) An ..... is a step-by-step method of solving a problem.
   (c) A ..... is an algorithm encoded into some programming language.
   (d) ..... is concerned with comparing algorithms based on the amount of computing resources used.
   (e) Efficiency of a program can be evaluated on the basis of ..... and ..... .
   (f) The overall efficiency of a program depends on the amount of ..... an algorithm requires to solve a program.
   (g) ..... is inversely proportional to the wall clock time.
   (h) Classification of algorithm is made on the basis of ....., ... or ..... performance.
   (i) For linear search, the worst-case running time is ..... in the size of the list.
   (j) ..... is a measure of efficiency of algorithm.
   (k) ..... is a way to enhance code productivity and quality.

2. State whether the following statements are True or False.

(a) An algorithm is a sequence of instructions written in English-like language.

(b) The best-case running time is the maximum running time for all the possible inputs.

(c) For a binary search, the best case for an input size of N is when it is sorted.

(d) Efficiency of linear search is better than that of binary search algorithm when the array is sorted.

(e) External factors such as size of input, speed of the computer, compiler that is used, etc., may affect an algorithm's efficiency.

(f) Complexity is the only absolute measure for evaluating program efficiency.

(g) In a program execution, more the number of steps, more is the time taken and lower is the performance.

(h) As the input size of a program grows, it affects the efficiency of an algorithm.

(i) In linear search, the worst case is when the target item is at the end of the list or not in the list at all.

(j) Recursive operations are less efficient than sequential programs with a large number of instructions to be executed.

(k) In the below code, number of operations done by the compiler is 10.

```
x=eval(input("Enter a number: "))
y= eval(input("Enter another number: "))
sum=x+y
print(sum)
```

3. Multiple Choice Questions (MCQs)

(a) The input that makes a given algorithm run slowest (or requires the most space or time) is termed as:
(i) Linear (ii) Best-case (iii) Worst-case (iv) Performance

(b) An algorithm whose run-time is proportional to problem size, at least for large problem sizes, is called:
(i) Best-case (ii) Linear (iii) Worst-case (iv) Efficient algorithm

(c) The process of locating an element from a collection (like a list or dictionary) or determining whether it is present or not is called:
(i) Searching (ii) Locating (iii) Addressing (iv) Binding

(d) Which of the following algorithms searches for the given item in a sorted array?
(i) Linear search (ii) Sequential search (iii) Bubble sort (iv) Binary search

(e) In a computer system, the ratio of the output to the input given with successful completion of tasks without wasting time or energy is described as:
(i) Efficiency (ii) Execution (iii) Behaviour (iv) Searching

(f) The performance of a program is inversely proportional to which factor?
(i) User time (ii) Execution time (iii) CPU time (iv) Memory

(g) Which technique reduces the amount of time taken by an algorithm to run as a function of the length of the input?
(i) while loop (ii) for loop
(iii) conditional constructs (iv) recursion

(h) In which process is each element of the array compared with the given item to be searched one by one?
(i) Linear search (ii) Binary search (iii) Bubble sort (iv) Insertion sort

(i) Which among the following is the best sorting technique when the number of elements is lesser?
(i) Selection sort (ii) Bubble sort (iii) Merge sort (iv) Insertion sort

(j) Under which situation does binary search algorithm turn out to be a worst case?
(i) Sorted array (ii) Unsorted array (iii) Linear array (iv) Linear list

SOLVED QUESTIONS

1. What do you mean by efficiency of a computer?

Ans. The efficiency of a computer is the ratio of output to input in a given system. A computer or a program is efficient if it is able to do tasks successfully, without wasting time or system resources such as memory.

2. What is algorithm?

Ans. It is a step-by-step process to solve a problem.

3. What is an algorithm efficiency?

Ans. Algorithm efficiency is related to the time taken by a program for execution and space used by the algorithm.

4. What is worst case in terms of algorithms?

Ans. The input that makes a given algorithm run slowest.

5. What is the best case in terms of algorithms?

Ans. The input that makes a given algorithm run fastest.

6. What are the internal factors on which performance of an algorithm depends?

Ans. (i) Time required to run
(ii) Space (or memory) required to run
(iii) Number of operations

7. What are the external factors on which performance of an algorithm depends?

Ans. (i) Size of input
(ii) Speed of computer on which it will run
(iii) Quality of compiler

8. Out of internal and external factors, which factors are of major concern for the performance of an algorithm and why?

Ans. Internal factors are of major concern for the performance of an algorithm. In internal factors, time is the major concern. We can control external factors to some extent, for example, we can replace the computer with a high-speed one or we can switch to any other compiler. Even space factor can be controlled by purchasing more memory but time factor cannot be controlled.

9. Write down any two simple suggestions that can be useful in designing algorithms.

Ans. (i) Redundant computations or unnecessary use of storage should be avoided.
(ii) Inefficiency due to late termination should be taken care of.

10. State condition(s) under which binary search is applicable so that it turns out to be most efficient.

Ans. For binary search, (i) the list must be sorted, (ii) lower bound, upper bound and the sort order of the list must be known.

11. Which is faster—linear search or binary search?

Ans. Binary search is faster, but the data needs to be in sorted order for binary search to work.

12. In what situations would you prefer linear searching over binary searching in terms of effective algorithm?

Ans. Linear search is preferred when one is doing a very small number of searches and the data is not sorted, or when the data set is so small that the overhead of a binary search is not worth the effort.

13. Given the following list/array, which search will find the value 3 in the least number of steps?

3 10 18 22

Ans. Linear/sequential search.

14. Define time complexity.

Ans. Time taken by the code to execute, *i.e.*, the number of iterations or statements, is known as time complexity.

15. Given the following array, which search will find the value 18 in the least number of steps?

3 10 18 22 35

Ans. Binary search.

16. Determine the memory consumption (memory efficiency) by the following list:

(a) Excluding the data's memory consumption. (b) Including the data's memory consumption.

```
L1 = ["Mr", "Puri", 5642, 4.5]
```

Consider it for Python 3.6, 64-bit implementation where the reference pointer is 4 bytes and list overheads are 36 bytes.

Ans. Length of given list L1 = 4

List overheads in Python = 36 bytes.

(a) Total memory consumption by list excluding data

= List overheads + reference-pointer size x length of list

= 36 + 4 x 4 = 52 bytes

(b) Memory consumption by list L1's data:
By string "Mr" = string overheads + 1 × length = 21 + 1 × 2 = 23 bytes
By string "Puri" = string overheads + 1 × length = 21 + 1 × 4 = 25 bytes
By integer 5642 = 12 bytes (12 bytes for an integer)
By float 4.5 = 16 bytes (16 bytes for a float)
Total memory consumption by data = 23 + 25 + 12 + 16 = 76 bytes
Total memory consumption by list L1 including memory consumption by data = 52 + 76 = 128 bytes.

17. Comment on the efficiency of linear and binary search in relation to the number of elements in the list being searched.

Ans. The linear search compares the search item with each element of the array one by one. If the search item happens to be in the beginning of the array, the comparisons are low; however, if the element to be searched is one of the last elements of the array, this search technique proves the worst as so many comparisons take place. The binary search, on the other hand, tries to locate the search item in minimum possible comparisons, provided the array is sorted. This technique proves efficient in almost all cases.

18. You have studied sorting methods in Class XI. Which of the following sorting—Selection sort, Bubble sort and Insertion sort—is more efficient?

Ans. Insertion sort is generally preferred for a small number of elements. The programming effort in this technique is trivial. However, sorting does depend upon the distribution of element values.
Selection sort is easy to use but performs more transfers and comparisons as compared to Bubble sort. Also, the memory requirement of Selection sort is more as compared to Insertion and Bubble sorting. With lesser memory available, Insertion and Bubble sorting proves useful.

19. On what factors is the quality of an algorithm judged?

Ans. The quality of an algorithm can be judged by simple points like:
(i) Simplicity of the solution, conciseness (ii) Ease of modification
(iii) Independence of a particular computer programming language or development environment

20. What factors does the efficiency depend upon?

Ans. Efficiency of an algorithm depends on its design and implementation. Since every algorithm uses computer resources to run, execution time and internal memory usage are important considerations to analyze an algorithm.

21. Write the code to find whether the inputted number is a prime number or not using two different approaches, with for loop and describe the most efficient approach of doing it using time(). (For Practical Implementation only)

```
import time
start = time.time()
a=int(input("Enter number:"))
k=0
for i in range(2,a//2+1):
  if(a%i==0):
    k=k+1
if(k<=0):
  print("Number is prime")
else:
  print("Number isn't prime")
end = time.time()
print(end - start)
```

Output:

```
Enter number:5
Number is prime
1.689096450805664
```

```
import time
start = time.time()
number = int(input("Enter any number: "))
if number > 1:
  for i in range(2, number):
    if(number%i)==0:
      print(number, "Not a prime no")
      break
  else:
    print("number is a prime number")
else:
  print(number, "is not a prime number")
end = time.time()
print(end - start)
```

Output:

```
Enter any number:5
5 is a prime number
1.909109115600586
```

Ans. As observed from the above snippets, the first approach is better and more efficient than the second one as executing the above task using for loop is less time consuming than giving for loop inside if...else construct and, thus, is more efficient.

22. Consider the given code and answer the following questions:

```
n=3
x=1
while x<=n:
  y=1
  while y<=n:
    z=1
    while z<=n:
      print(x,y,z)
      z=z+1
    y=y+1
  x=x+1
```

(a) What is the run-time efficiency of the above code segment?

Ans. There are three nested loops, each loop gets executed 3 times (as n=3). So, the run-time efficiency is $3 \times 3 \times 3 = 27$ or $n \times n \times n = n^3$.

(b) Give its output.

Ans.
```
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
2 3 1
2 3 2
2 3 3
3 1 1
3 1 2
3 1 3
3 2 1
3 2 2
3 2 3
3 3 1
3 3 2
3 3 3
```

(c) What will be the run-time efficiency if n=2?

Ans. If n=2, then run-time efficiency will be $2 \times 2 \times 2 = 8$.

23. What is the best and worst case in sorting techniques?

Ans. When the array or list is in ascending order and we are arranging the data in the same order, then that is the best case.

When the array or list is in ascending order and we are arranging the data in the reverse order, *i.e.*, descending order or vice versa, then that is the worst case.

24. Using time() function to count the number of seconds elapsed since the numbers are taken in the range of (1,10), find the sum of the product of numbers taken in this range. Also, calculate the time taken to execute it. (For Practical Implementation only.)

Ans.
```
import time
start = time.time()
r=0
for i in range(10):
    for n in range(10):
        r = r+(i*n)
        print(r)
end = time.time()
print(end - start)
```
Output:
```
1944
2025
0.46056270599365234
```

UNSOLVED QUESTIONS

1. Define efficiency of a program.
2. How can you relate efficiency with algorithm?
3. What is algorithm analysis?
4. Write a program to show efficiency of a program in terms of time.
5. How can you say that binary search program is more efficient than linear search?
6. Consider the following code. How many times will it execute? In the context of the given code, mention the methods used to find out the total time taken by the program to execute?
```
import time
start = time.time()
a = range(100000)
b = []
for i in a:
    b.append(i*2)
end = time.time()
print(end - start)
```
7. Write two different programs for the same problem and explain how efficient one program is over another and which program takes lesser time.
8. Explain (a) Linear search method, (b) Binary search method. Which of the two is more efficient for searching an element in the given data set?
9. Determine memory allocated to following sequences:
   (a) `(3, 4.0, '5')` (b) `["I", "love", "my", "India"]`
   (c) `[(1), (22.15,), (None,)]`
10. If through Python 64-bit implementation, the reference-pointer size is 8 bytes and list-overheads is 36 bytes, then determine the memory allocated to the following sequences. (Do not consider memory allocated to actual data.)
    (a) `[3, 4, 5, 6]` (b) `[3.0, 4.0, 5.0, 6.0]`
    (c) `["3.0", "4.0", "5.0", "6.0"]` (d) `[3, 5]`
    (e) `[3,4,5]` (f) `["3.0, "4.0", 6.0"]`
    (g) `(2, )` (h) `(3, 4.5)`
    (i) `("Hello",)` (j) `(3.0, 4.0, 5.0, 6.0, 7.0)`
11. Differentiate between time complexity and space complexity.