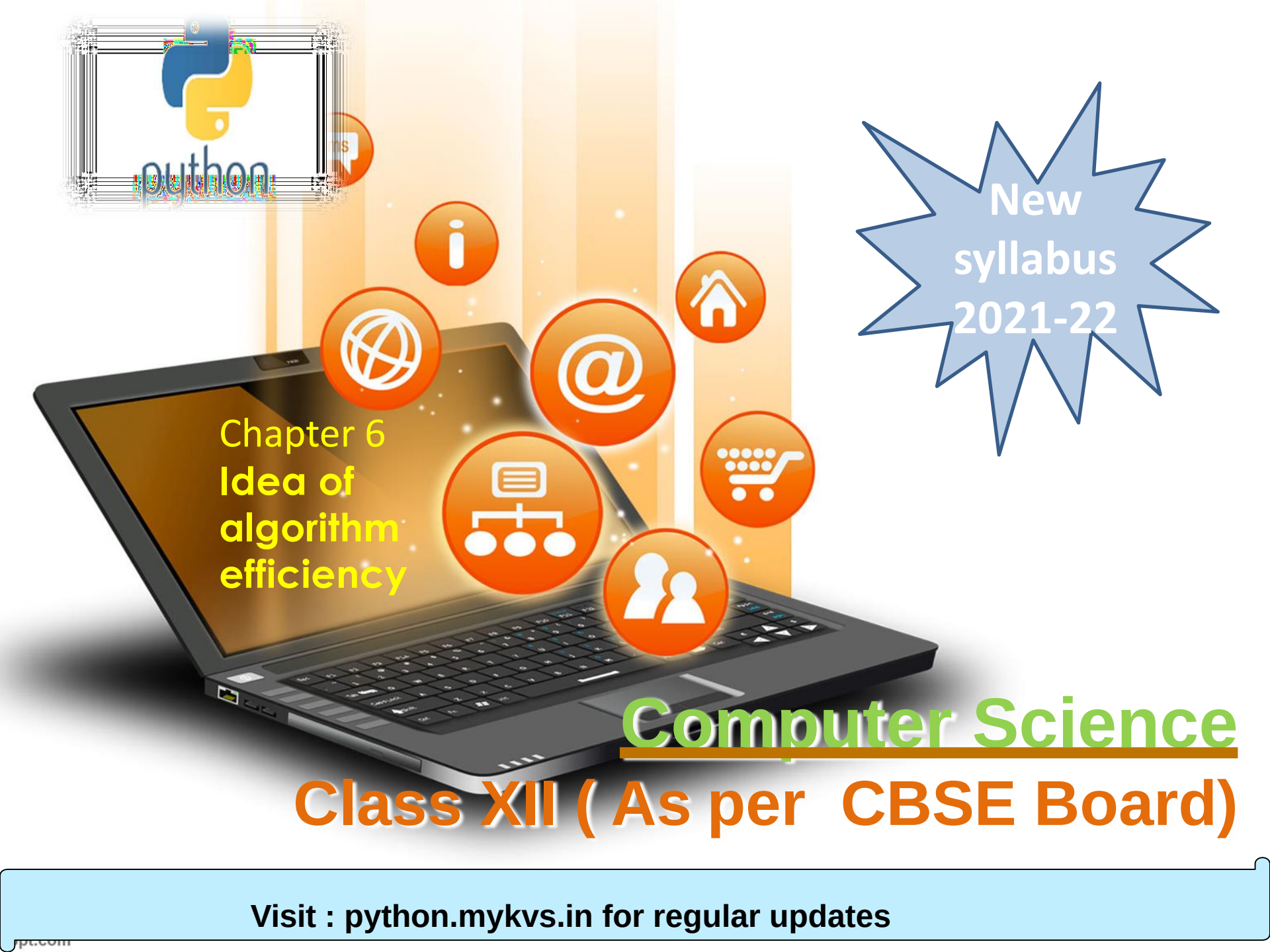
python
New syllabus 2021-22
Chapter 6
Idea of algorithm efficiency
Computer Science
Class XII (As per CBSE Board)
Visit : python.mykvs.in for regular updates

Idea of Efficiency - algorithm

Efficient programming is a manner of programming that, when the program is executed, it uses a low amount of overall resources pertaining to specially 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 different programs/algorithm varies, depend upon the number of resources being used. Practicing to create a low size(number of line of codes/number of operations) and low resource algorithm results in an efficient program.

Idea of Efficiency - algorithm

Performance defined as inversely proportional to the wall clock time:-

- Wall clock time/elapsed time: time to complete a task as seen by the user. In wall clock timing all kind of time is included ,e.g. operating system overhead or potentially interfering other applications etc.
- CPU time: does not include time slices introduced by external sources (e.g. running other applications).

Idea of Efficiency - algorithm

Performance defined as inversely proportional to the wall clock time:-To maximize performance, minimize execution time

performance = 1 / execution_timeX

"X is n times faster than Y"

– Execution time on Y is n times longer than on X

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

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?

= Performancedesktop/ Performancelaptop

= (1/60)/(1/90) = 1.5 So, the desktop is 1.5 times faster than the laptop

Idea of Efficiency - algorithm

Performance of algorithm depends on many internal and external factors.

Internal factors- time required to run and memory required to run

External factors – size of input to the algorithm , speed of computer

External factors are controllable, so many internal factors are studied an measured for algorithms efficiency.

We will determine efficiency of algorithm in terms of computational complexity. Computational complexity – computation+complexity

Computation involves the problems to be solve and algorithm to solve them.Complexity involves study of factors to determine how much resource(time to run+memory) is necessary for algo to run efficiently.

Big o notation- allow us to measure the time and space Complexity of our code.

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations:-

To compute the number of operations in a piece of code,then simply count the number of arithmetic operations+other operation that code is performing. All operations (addition, subtraction, multiplication, and division) are usually counted to be the same, which is not exactly true, since multiplication includes several additions and division includes several multiplications when actually executed by a computer. However, we are looking for an estimate here, so it is reasonable to assume that on average, all operations count in the same manner.

Here is an example (just for illustration):

```
r=0
for i in range(4):
    for n in range(4):
        r = r+(i*n)
print(r)
```

For each r there is 1 multiplications, 1 addition and 1 assignment resulting in 3 operations. This loop is executed 4X4 times, so there are (4X4)r operations. This is the the order of the code. In this example, its is $O(4^2r)$.

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations – (How to calculate)

1. Loop

```
for i in range(n):
        m=m+2
```

(All the steps in loop take constant time c) & (loop is executed n times)

Total time = c*n =cn -> O(n)

2. Nested Loop

```
for i in range(n):
        for j in range(n):
                m=m+2
```

(Steps in red will take cn times)(outer loop executed n times)

Total time= cn*n = cn^2 -> $O(n^2)$

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations – (How to calculate)

3. Consecutive statements

```
x=x+1                       #constant time=a
for i in range(n):          #constant time=cn
        m=m+2
```

```
for i in range(n):          #constant time=bn^2
        for j in range(n):
                m=m+2
```

Total time = a+cn+bn2 = O(n2)[Considering only the dominant term]

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations – (How to calculate)

4. If then else statements

```
If len(x) !=len(y)                          #a
        return false                        #b
else
        for i in range (n)                          #(c+d)*n
                if x[i]!=y[i]:              #c
                        return false        #d
```

Total time=a+b+(c+d)*n=O(n)

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations – (How to calculate)

5. Logarithmic running time (O(log n)) essentially means that the running time grows in proportion to the logarithm of the input size - as an example, if 10 items takes at most some amount of time x , and 100 items takes at most, say, 2x , and 10,000 items takes at most 4x , then it's looking like an O(log n) time.Example program is binary search.Such alogrihthm may return result in best case,average, and wors case

- The worst-case complexity of the algorithm is the function defined by the maximum number of steps taken on any instance of size n. It represents the curve passing through the highest point of each column.
- The best-case complexity of the algorithm is the function defined by the minimum number of steps taken on any instance of size n. It represents the curve passing through the lowest point of each column.
- Finally, the average-case complexity of the algorithm is the function defined by the average number of steps taken on any instance of size n.

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations –

5. Logarithmic running time (O(log n)) of binary search

Best case - O (1) comparisons -In the best case, the item X is the middle in the array A. A constant number of comparisions (actually just 1) are required.

Worst case - O (log n) comparsions -In the worst case, the item X does not exist in the array A at all. Through each recursion or iteration of Binary Search, the size of the admissible range is halved. This halving can be done ceiling(lg n) times. Thus, ceiling(lg n) comparisons are required.

Average case - O (log n) comparsions - To find the average case, take the sum over all elements of the product of number of comparsions required to find each element and the probability of searching for that element. To simplify the analysis, assume that no item which is not in A will be searched for, and that the probabilities of searching for each element are uniform.

The difference between O(log(N)) and O(N) is extremely significant when N is large: for any practical problem it is crucial that we avoid O(N) searches. For example, suppose your array contains 2 billion (2 * 10**9) values. Linear search would involve about a billion comparisons; binary search would require only 32 comparisons!

Idea of Efficiency - algorithm

performance measurement in terms of the number of operations –

5. Logarithmic running time(O(log n)) of binary search (How to calc.)

The most commonly used Big O descriptions are

O(1) always terminates in about the same amount of time, regardless of the input size.

O(logN) takes a fixed additional amount of time each time the input size doubles.

O(N) takes twice as long to finish if the input size doubles.

O(N2) takes four times as long if the input size doubles.

O(2N) increases exponentially as the input size increases.

You can see from the table below that the difference is small for small input sizes, but it can become tremendous as the input size increases even a little bit.

Input Size	Time to Complete				
	O(1)	O(logN)	O(N)	O(N2)	O(2N)
1	1	1	1	1	1
2	1	2	2	4	4
4	1	3	4	16	16
8	1	4	8	64	256
16	1	5	16	254	65536

Idea of Efficiency - algorithm

Measure the time taken by a Python Program

To measure the script execution time is simply possible by using *time* built-in Python module. time() function is used to count the number of seconds elapsed since the epoch.

e.g.program

```
import time
start = time.time()
r=0
for i in range(400):
    for n in range(400):
        r = r+(i*n)
print(r)
end = time.time()
print(end - start)
```

OUTPUT

```
6368040000
0.12480020523071289 #TIME TAKE TO EXECUTE THE PYTHON SCRIPT
```

Idea of Efficiency - algorithm

Compare programs for time efficiency

With the help of time() function,we can compare two/more programs with different algo for same problem that which one take less time.Below two code scripts are for prime no time efficiency purpose.

```
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

Idea of Efficiency - algorithm

number of comparisons in Best, Worst and Average case for linear search:

Program

```
def search(arr, x):
    for index, value in enumerate(arr):
        if value == x:
            return index
    return -1

# Driver Code
arr = [11, 10, 30, 15]
x = 30
print(x, "is present at index",
      search(arr, x))
```

Idea of Efficiency - algorithm

number of comparisons in Best, Worst and Average case for linear search:

Best case-

The element being searched may be found at the first position.

In this case, the search terminates in success with just one comparison.Thus in best case, linear search algorithm takes O(1) operations.

Worst Case-

The element being searched may be present at the last position or not present in the array at all.

In the former case, the search terminates in success with n comparisons.In the later case, the search terminates in failure with n comparisons.Thus in worst case, linear search algorithm takes O(n) operations.

Idea of Efficiency - algorithm

number of comparisons in Best, Worst and Average case for linear search:

Average case-

The element being searched may be found at the center position.

In this case, the search terminates in success with just half of elements comparison.Thus in best case, linear search algorithm takes O(n/2) operations.

.