

1/12/22

Date _____
Page _____

Design and Analysis of Algorithm

Computers works on IPO cycle i.e input process output
Any product is design analysed based on an algorithm

Algorithm: It is defined as the step by step procedure required to produce or complete a task.

Algorithm to display sum of three numbers

Accept a, b, c

Total = a + b + c

Display Total

There are many types of algorithms like, A₁, A₂, A₃, ... we have to select best type of algorithm

Best means

- i) It must take less time to execute/run'
- ii) It must take less space / memory

To check the time, use system libraries / modules

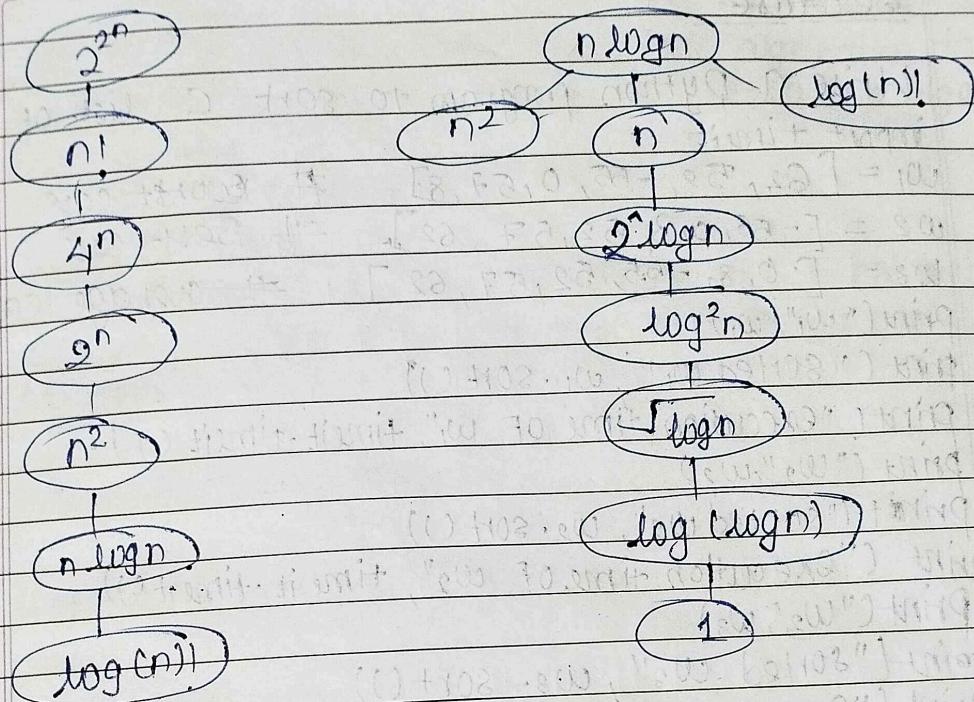
$$m_1 = \begin{bmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$m_2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 8 & 9 \end{bmatrix}$$

- An algorithm is defined as a step by step instructions to solve a particular problem
- There are many solution to a given problem
- The main goal of analysis of algorithm is to compare the algorithm in terms of running time, memory and other factors.
- Running time analysis
- Input to a given problem can be of many type
- some of the common type of input are :
 - i) elements of matrix
 - ii) elements in an array
 - iii) edges and vertices of a graph
 - iv) equations of polynomial degree
 - v) Binary representation of numbers
 - vi) Draw image.
 - vii) Video
 - viii) Audio etc.
- Processing time increases as the size of the problem increases / input increases
- algorithms can be compared based on execution time and no. of instruction to be executed / amount of space or memory required to execute
- The rate of growth of an algorithm is a function of input.

Amortised

Commonly used rate of growth



3/12/22

Types of analysis of algorithm

i) Best case ii) Worst case iii) Average case.

Write a python code to print factorial of a number.

import timeit

result = 1

i = 1

q = int(input("enter a number"))

while i <= q:

result = result * i

i = i + 1

print("factorial of", q, "is", result)

print("Running time is") timeit.timeit()

O/p. enter a number : 6

factorial of 6 is 720

running time is : 0.007679600006667897

Best case

Write a Python program to sort a list of data

```

import timeit
w1 = [62, 52, -55, 0, 57, 8]    # Best case
w2 = [-55, 0, 8, 52, 57, 62]    # Best case
w3 = [0, 8, -55, 52, 57, 62]    # average case
print("w1", w1)
print("sorted w1", w1.sort())
print("execution time of w1", timeit.timeit())
print("w2", w2)
print("sorted w2", w2.sort())
print("execution time of w2", timeit.timeit())
print("w3", w3)
print("sorted w3", w3.sort())
print("Execution time of w3", timeit.timeit())

```

w1 = 0.00752000183243304
 Types of w2 = 0.00745720001226738
 w3 = 0.0205319999026321

- In order to analyse the algorithm we need to know on what input the algorithm takes less time and on what input the algorithm takes long time algorithms are given as simple instructions
- A complete algorithm multiple instructions
- Accordingly we have 3 types of analysis

1) Best case

It defines the input for which the algorithm takes lowest time it has the input where the algorithm runs fastest

Best

Worst

Date _____
Page _____

Date _____
Page _____

2) Worst case

It defines the input for which the algorithm takes long time. It has the input for which the algorithm runs slower.

3) Average case

It provides a prediction about the running time of the algorithm. It assumes that the input is random.

11 Asymptotic notation

Consider the eqn $n^2 + 500$

$$n + 100n + 500$$

In all the three cases of best, worst and average cases we need to find the upper bound and the lower bound.

Lower bound is < average time > upper bound.

Every algorithm is represented in the form of a function. Let $f(n)$ be a function.

Based on the function $f(n)$ we have a big-O notation i.e. $O(n)$ which gives the tight upper bound of the given function $f(n)$.

lowest

Best $\Rightarrow O(n)$

$O(n) \rightarrow$ Big theta notation for $f(n)$

It gives tight lower bound for $f(n)$

import

a = [1]

total

x = 1

i = 0

while

i < len(a):

for i in

a:

sum = sum + 1

print

sum

0.00 7444

print

(running time) + timeit.timeit()

0.00 7444

6/1/19

Date _____

Page _____

~~Big O notation~~
~~It gives average case of f(n)~~

~~Write a python code to print the sum of elements in a list~~

~~import timeit~~

~~a = [1, 2, 3, 4, 5, 6]~~

~~print (sum(a))~~

~~print("running time") + timeit.timeit()~~

~~0.00 7444~~

~~list processing~~

~~print~~

~~print("running time") + timeit.timeit()~~

~~0.00 7444~~

~~Program -> Import timeit~~

~~a = [1, 2, 3, 4, 5, 6]~~

~~total = a[0] + a[1] + a[2] + a[3] + a[4] + a[5]~~

~~print ("the total is!", total)~~

~~print ("running time") + timeit.timeit()~~

~~import timeit~~

~~a = [1, 2, 3, 4, 5, 6]~~

~~sum = 0~~

~~while i <= len(a):~~

~~sum = sum + 1~~

~~i = i + 1~~

~~print (sum)~~

~~print ("running time") + timeit.timeit()~~

~~0.00 7444~~

```

import timeit
a = [1, 2, 3, 4, 5, 6]
total = 0
n = len(a)
x = x - 1
i = 0
while i <= x:
    total = total + a[i]
    i = i + 1
print("sum of elements", total)
print("running time") timeit.timeit()
print("running time") timeit.timeit()

import timeit
a = [1, 2, 3, 4, 5, 6]
t = 0
x = len(a)
for i in range(0, x):
    t = t + a[i]
print("sum =", t)
print("running time") timeit.timeit()

execute the sum of 5 elements & 20 elements

```

Program to count even and odd no. using for loop

a = [1, 2, 3, 4, 5, 6, 7, 8]

for i in len(a):

if i % 2 == 0:

e = 0

o = 0

for i in len(a):

if i % 2 == 0:

e = e + 1

else:

o = o + 1

print("Odd no. count:", o)

print("Even no. count:", e)

import timeit

x = [-1, -9, -7, 6, 8, 4, -5, -6, 7, 8, 9]

print("the maximum value is:", max(x))

print("the minimum value is:", min(x))

print("running time", timeit.timeit(x))

The maximum value is: 9
the minimum value is: -9

running time: 0.0076653

Method 2:

$x = [1, -9, -4, 6, 18, 14, -56, -67, 89]$

maximum = $x[0]$

i = 1

while i < len(x):

if $x[i] >$ maximum:

maximum = $x[i]$

i += 1

print("the maximum value is:", maximum)

print("running time:", timeit.timeit())

minimum = $x[0]$

i = 1

while i < len(x):

if $x[i] <$ minimum:

minimum = $x[i]$

i += 1

print("the minimum value is:", minimum)

print("running time:", timeit.timeit())

9/12/22

Date _____
Page _____

II Search:

1) Linear search

2) Binary search

python

write a program to search for a specific element in a given list

1) Declare list

2) accept search element

3) Begin the loop

4) traverse the list

5) compare list element with search element

6) if equal search is successful otherwise

compare

searching a list

import timeit

a = [66, 123, 0, -45, 53, 111]

Method
using built
in for loop
function

found = False

search = int(input("Enter the value you want to search:"))

for data in a or not found :

if data == search:

found = True

break

if found == True:

print(search, "is present")

else:

print(search, "is not present")

print("Running time", timeit.timeit())

Method 2

using count function

```
my_list = [66, 123, 0, -45, 53, 111]
search = int(input("enter the value you want to search:"))
x = my_list.count(search)
if x == 0:
    print(search, "is present")
else:
    print(search, "is not present")
print("running time:", timeit.timeit())
```

method 3 for loop:

```
my_list = [66, 123, 0, -45, 53, 111]
search = int(input("enter search element:"))
x = len(my_list)
for i in range(0, x):
    if my_list[i] == search:
        print("search is successful")
        print("running time:", timeit.timeit())
    else:
        print("search is not successful")
        print("running time:", timeit.timeit())
```

10/12/22

Date _____
Page _____

Two dimensional Array

Write a python program to take matrix obtain sum of 2

$m_1 = [[1, 2], [3, 4], [5, 6]]$
 $m_2 = [[7, 8], [12, 11], [4, 9]]$

$r = \text{len}(m_1)$

$c = \text{len}(m_1[0])$

result = 0

$m_3 = [[0, 0], [0, 0], [0, 0]]$

for i in range(0, r):

 for j in range(0, c):

$m_3[i][j] = m_1[i][j] + m_2[i][j]$

for i in range(0, r):

 for j in range(0, c):

 print(m3[i][j])

 print(" ")

print()

Write a python code to obtain row sum and column sum of 2D matrix array

$\text{expense} = [[30, 30, 70], [20, 40, 35], [10, 50, 60], [50, 70, 80], [40, 40, 50]]$

$r = \text{len}(\text{expense})$

$c = \text{len}(\text{expense}[0])$

day_total = 0

for i in range(0, r):

 day_total

```

For i in range (0, r):
    day_total[i] = 0
    day_total[i] = day_total[i] + expense[i][i]
# printing daywise expenditure
for i in range (0, r):
    print(i, day_total[i])

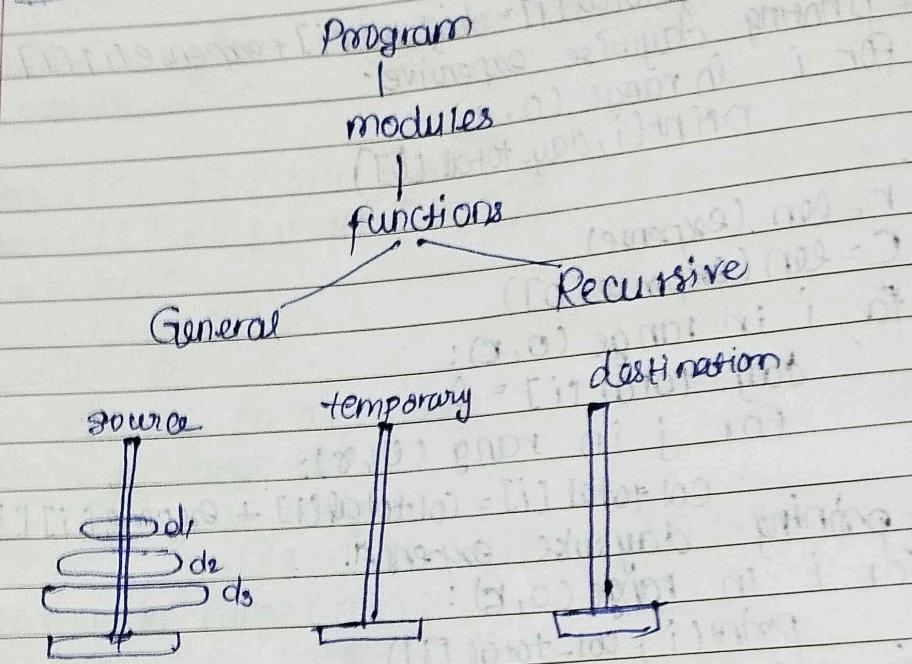
```

```

r = len(expense)
c = len(expense[0])
for i in range (0, r):
    day_total[i] = 0
    for j in range (0, c):
        col_total[i] = col_total[i] + expense[i][j]
        col_total[i] = col_total[i] + expense[i][j]
# printing day wise expenditure
for i in range (0, r):
    print(i, col_total[i])
for j in range (0, c):
    print(j, row_total[j])
    print(j, row_total[j])
final = 0
for i in range (0, r):
    for j in range (0, c):
        final = final + expense[i][j]
print("Total expense", final)

```

Recursion



d ₁	source	destination
d ₂	source	temporary
d ₃	temporary	definition temp
d ₄	s	d
d ₅	dest	s
d ₆	temp	s
d ₇	s	d

Transfer of control

main

f1() → f1()

return

f2() → f2()

f2 ←

f3() → f3()

f3 ←

return

General function to find factorial

def factorial(n):

f = 1

i = 1

while i <= n:

f = f * i

i += 1

print("Factorial of", n, "is", f)

def rFactorial(n):

assert n >= 0, "Factorial of -ve no. doesn't exist"

if n < 2:

return 1

else: ← rFactorial = 1

return (n * rFactorial(n-1))

main

print("Recursive factorial", rFactorial(5))

- Recursion is the process of solving a problem by subdividing the problem into smaller cases.
- we obtain the soln to the smaller problem to solve the larger problem
- It is a powerful programming and problem solving tool
- It can be used to solve traditional iterative problems and also advanced backtracking problems

Learning bit or not not solved

(a) learning not

$$1 = 2$$

$$1 = 3$$

$\therefore 0 \rightarrow 1$ given

$$1 * 2 = 3$$

$$1 = 1$$

(b) "if(i == 0) return 1"

(c) learning not

"the result is 0 if i == 0" & 0 < i <= 0

$\therefore 0 > 0 \neq 1$

Condition

= bnot01

((i-1) > not01 + 1 * 0) given

((a) learning not, given, writing ratio)