1/12/22

Design and Analysis of Algorithm

Computers works on IPO cycle i.e input process output
Any product is design and typed 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, $A_1, A_2, A_3, \ldots$
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.

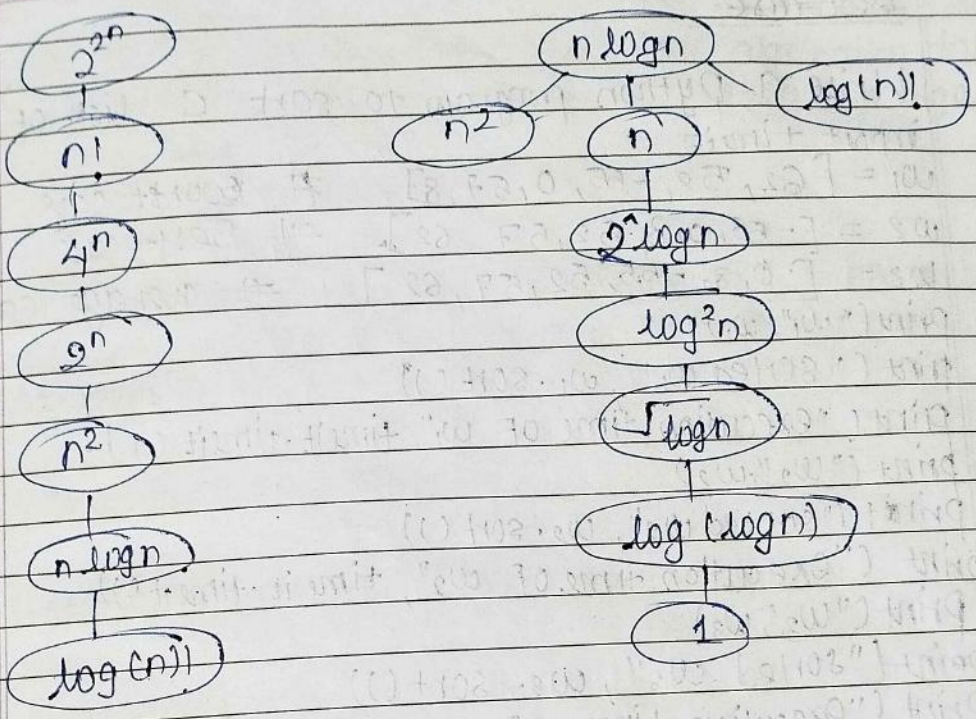
⇒ Running time analysis

- 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

47, 52, 55, 57, 62, 69



3/12/22

Types of analysis of algorithm

1) Best case 2) worst case 3) average case.

Write a python code to print factorial of a numbers

```
import timeit
result = 1
i = 1
a = int (input ("enter a number"))
while  i <= a:
        result = result * i
        i = i + 1
print ("factorial of", a, "is", result)
print ("Running time is") timeit.timeit()
```

O/p. enter a number : 6
factorial of 6 is 720
running time is : 0.007679000066897

Best case

Write a python program to sort a list of data

```
import timeit
w1 = [62, 52, -55, 0, 57, 8]    # Worst 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())
```

Types of
w1 = 0.007520001832433064
w2 = 0.007457200012267 38
w3 = 0.020531999902632 1

- 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 run is fastest

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.

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)

Best 1) O(n)
worst 2) $\Theta(n)$ → Big theta notation for f(n)
It gives tight lower bound for f(n)

Big Ω notation
It gives average case of f(n)

6/12/22

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())
```

0.00762772

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

0.00 7444

List processing

```
Program to 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())
```

0.00756060

/* */

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

Execute the sum of 5 elements & 20 elements

Program to count even and add no

```
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,-7,6,8,44,-56,-67,89]
print("the maximum value is:", max(x))
print("the minimum value is:", min(x))
print("running time", timeit.timeit())
```

```
the maximum value is: 89
the minimum value is: -67
running time: 0.057665
```

Method 2

```
x = [1, -9, -7, 6, 88, 114, -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())
```

3/2/22

Method 2

Search:

1) Linear search
2) Binary search

write a python 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

Method1 using built in for loop function

```
#Searching a list
import timeit
a = [66,123,0,-45,53,111]
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

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

method 3 for loop:

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

10/12/22

Two dimensional Array

Write a python program to ~~take~~ obtain sum of 2 matrix

```
m1 = [[1, 2] [3,4] [5,6]]
m2 = [[7,8], [12,1], [4, 9]]
r = len(m1)
c = len(m, [0])
result = 0
m3 = [[0,0], [0,0], [0,0]]
For i in range (0,r):
    for j in range(0, c):
        m3[i][j] = m1[i][j] + m2[i][j]
for i in range(0,r):
    for j in range (0,c):
        print(m3[i][j])
        print(" ")
print()
```

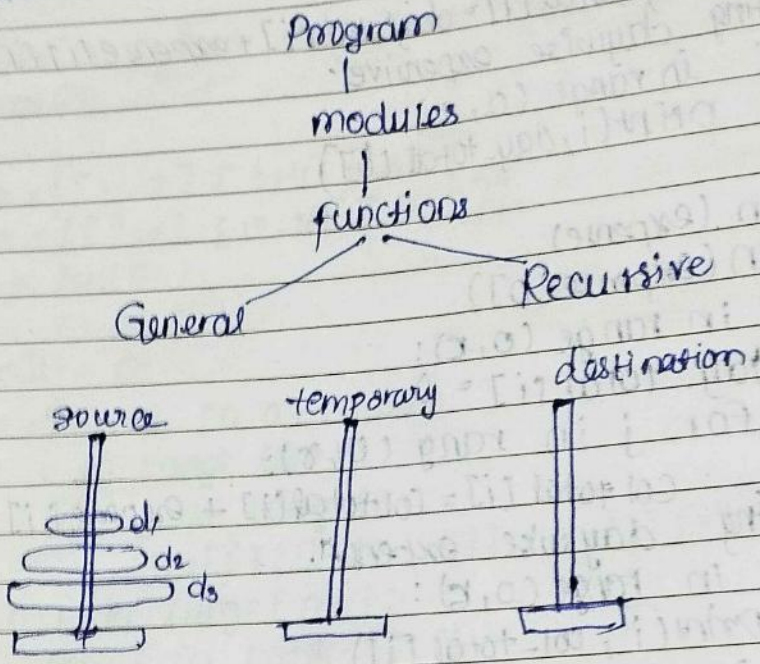
Write a python code to obtain row sum and column sum of 2D ~~matrix~~ array

```
expense = [[30,30,70], [20, 40, 35], [10, 50, 60], [50, 70, 80], [40, 40, 2]]
r = len(expense)
c = len(expense[0])
day_total = 0
For i in range(0,r):
    D[i]
```

```
for i in range (0,r):
    day_total[i]=0
    day_total[i]=daytotal[i]+expense[i][j]
# printing daywise expensive.
for i in range (0, r):
    print(i, day_total[0])

r=len(expense)
c=len(expense[0])
for i in range (0,c):
    day_total[i] = 0
    for j in rang (0, r):
        col_total[i]=col_total[i] + expense[i][j]
# printing daywise expense.
for i in range (0,c):
    print(i, col_total[i])
for j in range (0,r)
    print(j, row_total[i]+=expense[i][j]
    print(j, row_total[i]]
final=0
for i in range (0,r):
    for j in range (0,c):
        final=final+ expense[i][j]
print("total expense", final)
```

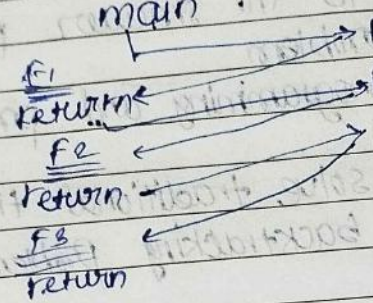
Recursion



d_1	source	destination	
d_2	source	temporary	
d_3	temporary	defination	temp
d_3	S	d	
d_1	dest	S	
d_2	temp	S	
d	S	d	

return to main

Transfer of control



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:
        return (n * rfactorial(n-1))
#main
print("Recursive factorial", rfactorial(5))
```

- Reccursion is the process of solving a problem by subdividing the problem into smaller cases.
- we obtain the sol" 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 problem