**DAY 1**

------------------------------------------------------------------------------------------------------------------------------

Optimization:

1.finding the optimum solution

2.lower the time complexity

\*itarative method

1.repeated approach

2.recurssion

\*dynamic programming

//DYNAMIC PROGRAMMING

1.In greedy approach whatever is the solution for the problem given at the first go is fixed

as the final solution

NOTE:

1.This is not the best approach for all the scenarios,however it also works for some cases

2.In dynamic programming ,we will be finding out all the possible solutions for the given

problem out of which the best will be selected

TIME AND SPACE COMPLEXITY:

--------------------------------------------------------------------------------------------------------------------

AYMPTOTIC NOTATIONS:

Three types of notations are:

[O(n)]

omega

theta

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

IMP POINTS:

Seive algorithm used to eliminate duplicate values

Id():command used to declare memory allocation in python

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Polynomial theorem f(n)=n+1

which is O(n) big o of n (or) order of n

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<stdio.h>

int main()

{

int i,n=5;

for(i=0;i<n;i++)

printf("%d\n",i);

printf("final i:%d",i);

return 0;

}

output:

0

1

2

3

4

final i=5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for(i=1;i<n;i+2)

{

statements

}

n/2

f(n)=n/2

Degree of polynomial is n

so n/anything is n

so here also o(n)

\*so irrespective of iteration time is gonna be same o(n)

==========================================================================

**DAY-2**

==========================================================================

TIME COMPLEXITY

for(i=0;i<n;i++)....//n+1 times

{

for(j=0;j<n;j++) //n\*(n+1)

{

Statements; //n\*n----n square

}+

}

;

Time complexity=o(n square)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#IMP POINTS:

---> Primitive data types

1.int

2.float

3.char

4.double

-Python supports non-primitive data type, whereas C and java support primitive data types

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for(i=0;i<n;i++)

{

for(i=0;i<n;i++)

{

statements;

}

}; // i=0 then 0<0 no i j

0 nothing

so when i 0....0 times

1..... 1 time 1 0 will execute

2...2 times

so 1+2+3+4+5 1 will stop

2 0,1 executes

2 will stop

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

p=0,

for(i=1;p<=n;i++)

{

p=p+1;

}

1 0+1 bcs p=p+1..when N is 0 now assuming n times when will s

2 1+2=3....N is 1

3 1+2+3=6

.

.

.

k 1+2+...k

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ANALYSE

i=1 1 time

i=2 2 times (1\*2)

i=3 4 times (1\*2)\*2=2 power 2

i=4 8 times (1\*2)\*2)\*2= 2 power 3

so when stops i>=n

i=2 power k

2 power k>=n

k log n base 2

so time complexity O(log n base 2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DERIVED FORMULAE:

for(i=0;i<n;i++) ----O(n)

for(i=0;i<n;i+2) ----O(n)

for(i=n;i>1;i--) ----O(n)

for(i=1;i<n;i=i\*2)--- O(log n base 2)

for(i=1;i>1;i=i\*3)--- O(log n base 3)

for(i=n;i>1;i=i/2)--- O(log n base 2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* Constant Time complexity: O(1)...

\* Linear Time complexity: O(n)...

\* Logarithmic Time complexity: O(log n)...

\* Quadratic Time complexity: O(n^2)...

\* Exponential Time complexity: O(2^n)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FINAL SUMMARY:

i++ i-- i+2 O(n)

i\*2 i/2 log(n) base 2

p=0

for p<n i++

p=p+i

O(sqrt(n)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SPACE COMPLEXITY :

1.array of size n will take n space and 2d array will take n\*n order of n^2 space

constant space complexity order of n

2.same amount of space regardless of the input size of n is called constant complexity

ex: sum of array elements and linear search

3.it is order of 1 beacause space is not depending upon values

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.linear compplexity:O(n)

2.log-linear complexity:O(nlogn)

-> when the space complexity of an algorithm grows proportionally to the input size

linear search:O(1)

polynomial complexity:o(n2):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <stdio.h>

struct a

{

char x;

double y;

int z;

};

int main()

{

struct a yes;

printf("%d",sizeof(yes));

return 0;

}

explanation:

char x --> 1+3 double char--> 1+7

int y --> 4 16 int 16 double--> 8 24

double -->8 char int --> 4+4

int double

char 16 char 16

double +int

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\\sum of array elements

#include<stdio.h>

int main()

{

int n;

printf("Enter arry size: ");

scanf("%d",&n);

int arr[n];

int s=0;

printf("Enter elements: ");

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

s+=arr[i];

}

printf("%d",s);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//generate list:

def generate\_lists\_of\_lists(n):

table\_list=[]

for num in range(n):

row=[]

for i in range(n):

row.append(i)

table\_list.append(row)

return table\_list

print(generate\_lists\_of\_lists(10))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_