Programming Efficiency:

- Efficiency is all about Time & space Complexity

→ Greedy Dynamic programming :(DP)

- In Greedy approach, what ever is the solution for the problem given at the First Go is fixed as the final solution.

Note:- This is not the best approach for all the scenarios however it also works for some cases.

- In DP, we will be find out all the possible solutions for the given problem out of which the best will be picked. → Time & Space Complexity:-

Asymptotic Notations:

1. Big O(n) Notation
2. Omega NOtation
3. Theta Notation

⇒ Swajith is Having one lakh is in bank account in the rate of interest is 12% per annum. In the 5th month swajith is withdrawing 25000rs in order to buy gift for his loved one. In 9th month 100000rs is been deposited in his account by his 2nd loved one. End of the Financial year how much swajith is having in his account? Note: simple interest not Compound acct.

Sol:

$\frac{3 \times 8 \times 15}{100} = \frac{27}{2}$

$\Rightarrow 25000 \times \frac{27}{25}$

$= 91800$

A: 95400

100000 → 12%

100

12000 — per month

$\frac{PTR}{100}$

$\Rightarrow \frac{1 \times 12}{100} = \frac{12\phi\phi}{100} = 12\%$

$\frac{75000 \times 850 \times 12 \times 1}{100}$

```
def rateofinterest(amount):
    rateofinterest = amount/100
    return rateofinterest;
amount=100000
b=amount-25000
new=b+100000
print(new+(rateofinterest(amount)*4)+
(rateofinterest(new*4)+(rateofinterest(new)*12))
```

Struct

{
int... 4
char... 1
}

space: 4+1+(highest data type)
= 4+1+3
= 8

=> {
int... 8
int... 2
}

space => 10

=> {
int... 8
double... 2
}

space => 10

=> combines two data types
datatype+datatype

=> {
double 8
char 1
}
}

space = 16 bytes
=> 8+1+7=16

Union

{
int... 4
int...
}

space = 4.

=> {
int = 6
double = 8
}
space: 8

=> {
int... 4
char... 1byte
}
space:

=> ~~In consists only 1~~ es
=> only 1 highest data type
=> lowest datatype is allocated in highest data types
=> lowest is occupied in highest.

In C

=> short
int. 4
double. 8
char. 1

space => 4.

= 4+4+8+1+1

=> How many times statement is getting executed why?

efficiency of the program.

Type 1:

```
For (i=0; i<n; i++)
{
statements:
}
```

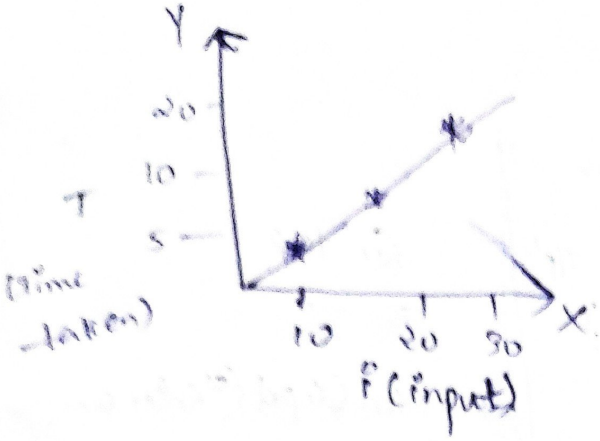
ex: n=5
- n+1 =
5+1=6

=> n+1 times

=> Ω (or) g(0) n. - (or) n order

=> polynomial theorem = n+1.

Time complexity:



① #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;
}

Time Complexity = O(n)

②

=> for(i=1; i<n; i=i*2)
{
statements
}

n/2

f(n) = n/2

Degree of polinomial is n

So n/anything is n

So here also

O(n)

=> anything/n gives same

Time complexity O(n).

∴ n & n ⇒ O(n)

Q) Get one number int as i/p & find sum of digits of given number -

```
#include <stdio.h>
int main()
{
int a=0
b=input("enter number")
while (a<5);
  a=a+1
  print b
```

=> multiple scanf i/p's

for loop

```
num = int(input("enter an integer number:"))
sum = 0
num = abs(num)
for digit in str(num):
  digit = int(digit)
  sum += digit
print("Sum of digits:", sum)
```

```
=> #include <stdio.h> int main()
{
int number = "enter number"
```

```
=> int input("enter number")
  rem =
```

=> while loop :-

```
#include <stdio.h>
int main()
{
int rem, i, n;
printf("Enter the value:");
scanf("%d", &n);
int sum=0
while (n>0)
{
rem = %10
n = n/10;
  sum = sum + rem;
}
printf("%d \n", sum);
return 0;
}
```