DESIGN AND ANALYSIS OF ALGORITHM

Design => method {methods are different}

Recursion => jis nai km chalna hai
Analysis => critical thinking {time, memory}

procedure => sequence of instructions
Algorithm well defined computational procedure that takes some value or set of values as input and produces set of values as output in a finite amount of time

=> sequence of Steps imp {for a procedure}

Sequence of computational steps that perform the input into output at a finite amount of time

Instance => input sequence

Algorithm's properties
→ must be correct {kyu k problem ko solve krta}
→ halts (finished)
→ finite time
→ correct output / solution

```
n = 1;
while (n! = 0)
{   cout << n;
    n = n - 2;
}
```

→ non-terminating

int = 2 Bytes => 2^{16}

-2^{15} $2^{15}-1$

{-32768, ... 0, ... 32767}

set of values

set of operations
→ +, -, /, %, *

Real numbers → Mantissa and Exponent

Data type is a set of values that defines on them by some / certain no of operations

→ {-32768, ... 0, ... 32768}

a = 32768

a++

cout << a

Output
=> -32768

→ ضروری نہیں ہے کہ algorithm غلط ہے اور وہ رک رہا ہے execute
ہو رہا ہوتا۔ ⇐ غلط algorithm سے بھی کبھی صحیح ans آجاتا ہے

Why do we need to study algorithm Imp!

- foundation of programming
- structural approach
- problem solving
- observe efficiency (resoure storage efficient and time)
- optimization
  { hammad والا program کام کر رہا لیکن اگر ہم اس کے اندر اچھا function / طریقے سے optimize کر سکیں }
- Data handling { زیادہ data آگیا تو وہی algorithm اس پر نہیں لگ سکتے جیسے searching }

Binary search => direct access
② sorted
for large data, binary search fail ⇐ kyu k hardisk se RAM mai lana hai اتنا نہیں آسکتا

- Innovation {google}
- Career opportunities { progress krte hum, purani chzo ko ~~dekh kr new banate~~, progress hota
- Understanding complexity => O(n)

Loop, Recursion
→ is complex but imp to understand this

Mohi

DAA

Computational model => choteh se algorithms ko mila kr banana

What kinds of problems are solved by algorithms?

- Search problems
- Sorting — Bubble sort (On^2), merge ($O\ n\log n$)

2^n => very fast growth.

- Optimization => $2^n \longrightarrow n$ => اس میں convert کرنا
- Graph Problems => complex nature, social media (in graph form)
- Data processing => image color enhance↑, noise↓ aggregate (data ikhta krna)
- Cryptography => crypt => meaning change krna {code words}
- Mathematical problems => designs.
- AI
- Simulation and Modeling => Car driving simulation k through, accidents dikhana
- Automated Decision Making => aeroplanes => auto, security system auto

=> $\log_2(-)$

Algorithm Technology

Computer A => 1000 times faster than B

Input size = 10 million => sort

Comp A	Comp B
$= \frac{2 \times 10^7 \text{ inst}}{10^{10} \text{ inst/sec}}$	$= \frac{50 \times 10^7 \log 10^7 \text{ inst}}{10^7 \text{ inst/sec}}$
= 20,000 secs => 5.5 hrs (more than)	= 1163 sec = less than 20 minutes

GCD => greatest common divisor

Representation of algorithm is multiple

Properties:-

- Input, Output — input ہمیں پتا ہونی چاہیے اور output بھی۔
- Precision — steps کو define precisely، sequence ہونی steps کی
- Finiteness — کہیں نہ کہیں رکنا
- Definiteness — جو define کریں وہ proper ہو، unclear نہ ہو
- Correctness — statement correct ہونی چاہیے
- Generality — کسی نہ کسی خاص data کے لیے ہو تاکہ hand free نہیں ہونے limits میں رہ کر algorithm لکھیں
- Non-static — صرف خاص inputs کے لیے نہ ہو۔

An Art and a Science

problem → thinking {mind} → solution

this is art {algorithm is art and science}

S - state the problem {problem کیا ہے}
T - tools for the job
A - algorithm development {picture dry run}
I - implementation of algorithm
R - Refinement {اس کو change یا ٹھیک کرنے}

DAA

Computational problem
jis mai calculations hon.
→ step by step, input, output, constraints.

→ is specified by one or more preconditions and postconditions
Precondition → کسی چیز سے پہلے initialize کر دیے
global variables ke initial values set krna
Post condition → جب وہ ختم ہوگی تو ہمیں required output / answer ملے۔
input has a relation with output
Exception occurs at output ← sometime

Search

pre condition => integer input
post condition => k woh integer kahan mila

$$\underbrace{0 \text{ ---- } i}_{j} \text{ ---- } n$$

~~Bubble Sort~~ Array => Minimum NO

- n : positive integer
- Arr : integer array of length n, with entries.
  Arr[0], Arr[1] .... Arr[n-1]
- Initialize int min = arr[0]
- min : An integer found in array (i.e min = arr[i] —— 0 — n-1)

```
Arr = {1, 2, 3, 4, 0}
min = arr[0]
for(i=0; i<5; i++)
{ if[arr[i] < min]
  { min = arr[i]; }
}
```

Post Condition
. Output is the integer i such that $0 \le i < n$ for every integer ie $0 \le j < i$ Arr[j] ≠ min

if not value found in search
↳ so output
↳ Exception.

Finding maximum — Integers => index

- list finite integers.
- find max integer from array/list
-

Write algorithms => in a proper format

Correctness
- → Partial (← output میں situation کچھ کچھ)
- → Total (← output میں situation ہر)

Partial — max => -1 → some cases
Total — max => arr[0]. → all cases

DAA

Loop Invariant

variable (variant) => kyu change ho.

invariant => constant

```
while () ---> ①
{
    stmnts ---> ②
}  ---> ③
```

- true when control enters a loop
- remains true each time the program executes the body
- true when control exits the loop

initialization, darmayaan (mantainence) mai, terminate hone k baad bhi true rahe called => loop invariant

Loop Invariant eg

i helping variable, a bhi fixed hai

invariant => s => kyu k sum kr raha

① s ko hum nai traverse nahi liye s=0

② s ander add hota jata

③ s bahir nikle raha

s is the sum of i array element

s joh variable hai us k 1st array element ka sum hai.

length = 0 => s = 0

length = 1 => s = 0 + arr[i]

=> s her length k liye thee k kaam kr rah.

guard => loop k ander condition while(guard)

variant => joh change horaha (i=i++)

i = 0 => s = a[0]

i = 1 => s = a[0] + a[1]

sum of i array element

Correctness of a loop

- Initialization
- Invariance
- Progress — end کی طرف move کرے
- Boundedness — true ہے تو چلے false پر باہر آئے، loop bound میں
- Exit — loop کی condition false ہو تو باہر آجائے

Purpose of loop Invariant

- Comment add krna (loop invariant ko comment mai likhen taken koi or programmer ko baad mai asaan ho)
- Debugging k liye easy hota (agar objective achieve hua toh debugging nahi krte, agar achieve nahi hua toh section wise - module wise debugging krte)
- Correctness of computer program

Task

```
power (int x, int n)
{ int p = 1, i = 0
  while (i < n)
  {
    p = p * x;
    i = i + 1;
  }
return p
}
```

number ↓ power ↓

The loop invarant is $p = x^i$

① Initialization => $p = 1$

② Invariance

$i = 0, p = x^0 = 1$

$i = 1, p = x^1 = 1$

③ Progress

$i = i + 1$ towards n

④ Boundedness

$i < n$ becomes false

⑤ Exit

$p = x^n$ checks the goal.

Recursion repetion, recursive call

Stack ⇒ LIFO m → a → b → c

Stack Overflow ⇒ function se jab hum dusre function ko call kte toh pehla function active nahi hota pichle ka sarah data stack mai dal date

1 → num tak number print krn using recursion

```
void func(int n)
{
    if (n>1)
    {break;}
    func(n-1);
    cout<<n;
}
```

```
void disp(int n)
{
    if (n==0)
    {return;}
    disp(n-1);
    cout<<n;
}
```

1
2
3
4
5

no need to improve another variable → performance slow

DAA

Recursive → depth approach اندر اندر جاتی ہے

Divide and conquer {Bigger problem ko smaller problem/pieces mai divide krte}

Steps

1. Base Case
2. Recursive case
3. Termination
4. Combine the solution

Factorial

```
int f(int n)
{
    if (n==1)
      { return 1; }
    else
    {
        return n * f(n-1);
    }
```

Stack overflow

calls infinite hojayen

```
int foo()
{
    return foo();
}
```

loop => a=1; while (true) { a = a + 1 }

infinite loop => no stack overflow because loop does not use stack

4 pillars of programming

1. Code segment
2. Data segments
3. Stack overflow
4.

How can we avoid stack overflow error?

جس کا بڑا data ہے اس کے لیے recursion کے use نہ کریں کیونکہ یہ سارا data stack میں store ہوتا اور اس سے error آتا ہے

↳ Use Iterative approach.

limit fix: depth کی limit لگا دیں کہ اس سے زیادہ وہ call ہی نہ ہو سکے۔

return multiple جگہ make sure کریں کہ وہ stop ہو (ایک یا لگائیں)

```
int fibonacci (int n)
{
    if (n==0 || n==1)
    {   return n; }
    else
    {   return fibonacci(n-1) + fibonnaci(n-2); }
}

int power (int base, int exp)
{
    if (exp==0)
    {   return 1; }
    else
    {   return base * power(base, exp-1); }
}
```

Time Complexity (TC)

Imp!!

- Finiteness property
- Kisi code ke rating
- Algorithm ke efficiency measure krna

execute جو instruction ہو رہی ہیں وہ کتنی دفعہ execute ہو رہی ہیں وہ number = time complexity ہے۔

Time complexity ~~Execution~~ does not depend on machine

It depends on number of instructions

time ⇒ number hai {second, hours نہیں ہوتے}

The time required by algorithm to solve given problem is called TC

```
int add (int a, int b)
{   return a+b;  ⟶ only 1 statement
}
```

(1 instruction)

$T(n) = O(1)$

```
int n   →
t = a   →
a = b   →   } 4 instructions  O(4)
b = t   →
```

جب statement کی execution کا data پر fixed نہیں depend کرتی تو ہم 'C' لکھتے ہیں

```
cin >> n  ⟶ 1
for (i = 0, (i < n), (i++))
{ cout << i << endl; } → n
```

Running time input size per depend kr raha yeh sab 'n' input per depend کر رہا ہے

1 $\quad 1 + 1 + n + n + n$

$3n + 2$

$\Rightarrow 3n \quad \Rightarrow n$

jitnai bhi constants hain un ko drop krden (+, −), jitnai bhi constant (×, −)

$\Rightarrow$

```
int main()
{ int i, n = 8;  ⟶ 1
  for (i = 1, i <= n, i = i * 2)  → n/2
  { cout << Hello; } ⟶ n/2
}
```

multiply کر رہی ہے calls

$\Rightarrow$ iteration half horahi

1

$1 + 1 + n + \frac{n}{2} + \frac{n}{2}$

$2 + n + n$

$2 + 2n$

$O(n)$

Sir Solution

constant na rakha only n.

i
1	$= 2^0$
2	$= 2^1$
4	$= 2^2$
8	$= 2^3$

$8 = 2^3$

$n = 2^{k}$ → find karna

$\log_2 n = k \log_2 2$

$\log_2$ کیونکہ 2 power میں ہے

$\boxed{\log_2 n = k}$

k => number -> 0,1,2,3 جو power میں ہے

$O(\log_2 n)$

$n = 2$
$j = 4$

```
for (i = 0; i < n; i++)
{   for(int j = 0; j < m; j++)
    {
        sum + = arr[i][j]
    }
}
```

$n = 4, m = 2$

n	m	
0	0→1	(2 times)
1	0→1	(2 times)
2	0→1	(2 times)
3	0→1	(2 times)

fix iterations

$n \times m$

matrix size => $n \times m$

(variables hain => variable)

=> $O(nm)$

square matrix donoh loops n tak chalte

1 → 4 $\quad 4 \times 4 = 16 = 4^2 = n^2$ | $4 \times 4 = n \times n$

2 → 4 $\quad n^2$

3 → 4

4 → 4

```
for(i=1; i<=n; i++)
{ for(j=0; j<i; j++)
  { cout << "*" << endl;
  }
}
```

$n = 4$

outer	inner
1	1
2	2
3	3
4	4
n	n

$1+2+3+\dots n$

$\frac{n(n+1)}{2}$

$= \frac{n^2+n}{2}$

Same iterations

→ $n^2 + n$

Rule => variable same, higher order => keep

=> $O(n^2)$