

Data Structures & DSAD

→ organizing the data efficiently.

Efficiency ;
in terms of space

and time.

So that

⇒ one searching, processing and
sorting of data gets better.

features

→ Each data structure allows
data to be stored in specific manner.

→ DS allows efficient data search
and retrieval.

→ Certain DS are decided to work for
specific problems.

→ allows you to inserting an

Dictionary : —

Array :

int numbers [4] = {10, 20, 30, 40};

What is ADT?

→ two ways of looking at DS,

→ mathematical / logical.

↳ DS follow some rules or segmentations to store data, these rules can be termed as logical or nuclear.

→ Implementation is using all those rules to implement DS practically with the help of any programming language.

DATA

Data types.

→ Defines a certain domain of values.

→ Defines operations allowed on these values.

int type

→ will take only integer values.
→ operations allowed subtraction.

ADT

ADTs are entities that are definitions of data and operations but don't have implementation details.

10	20	30	40	50
Index	0	1	2	3

Abstract view

→ stare set of elements.

→ Read element by index

→ modify elements by index

Implementation view

int arr[5] = {1, 2, 3, 4, 5};

cout << arr[1];

arr[2] = 10;

Algorithm

finite number of steps to solve a particular problem.

S₁ → Read A

S₂ → Read B

S₃ → sum = A + B

S₄: Print (sum).

→ instructions should be finite time.

Analysis:

comparison of different algo's on a specific parameters.

i.e time, space.
no. of registers, throughput.

Two types of analysis -

1^o Prior

→ Analysis before execution.

↳ find out the iteration

of any function.

→ It gives approx value.

example:

S₁ : Read A

S₂ : Read B

S₃ : Sum = A + B

S₄ : Print(Sum)

Posterior:

→ giving better values.

→ This analysis is often redundant.

main ()

int a, b, sum;

scanf ("%d %d");

sum = a + b;

printf ("%d");

Code.

Primitive and reference data types -

Primitive types :

(~~pointer~~)

int, long, double, float, and char.

int i = 15;

char c = 'a';

i

15

1004

0

c

a

1016

data stores directly into memory.

Reference type :

types that hold complex values.

(objects)

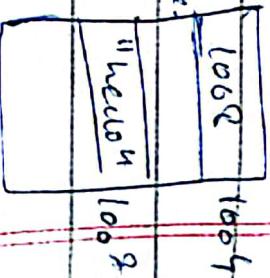
→ string is a reference type.

String s = "hello";

↳ This will contains the address of value.

→ variable references (me & value).

value.



s ↓ hello

it's act like just a pointer

int i^o = 5.

int i^o = 11;

i ^o	5	800
i ^o	1020	600
i ^o	1020	1020

String s^o = "Hello";

String s^o = s^o - s^o - "Hello" - 1020

Object and differences

Object is an instance of a class.

[Clocks]

white-board

person table

Creating
new
location

Person salary = new Person();

constructor

Class Object name

salary = 97

8	9	0
9	1	1
1	0	1

→ object saying will

actually refers to ^{new} location

To actual location person

of the class.

Common analogy:

Class → House → Blue point.

Object → actual house.

Memory reference + gives the

actual idea that where the
actual house is, street, country etc.

Person Adress = new person()

Person p_1 = new person();

Person p_2 = new person();



these actually are not persons
but are memory references

that refers to the memory locations.

Types of data structures:

D₃

Linear

non-linear

↳ when all the elements are arranged in a sequential order.

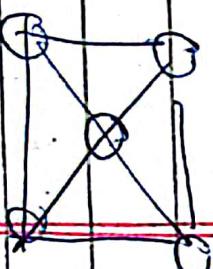


array graphs stacks linked list

We also say that when a ds have only one predecessor and one successor.

Non - Linear DS :

a ds when all the elements are not stored in a linear order.



0 0 0 → 1 predecessor

0 0 0 → 2 Successor graphs

~~REVIEW~~

Interview Questions:

Static and dynamic DS ?

Asymptotic Analysis :

A DS is the organization of the data in a way so that it can be used efficiently.

map.

→ In order to use efficiency, we have to store it efficiently.

Efficiency of DS is always measured in terms of time and space.

→ Ideal DS would be tract takes less time and consumes least memory location.

(Q) On what basis, we compare the time complexity of DS.

f) On the basis of operations performed on the.

Ex :-

arr [1 | 2 | 5 | 6 | 8 | 9 | 10 | 8]
 ↑
 ↑

assume our array can store upto
100 elements.

Add data ~~at~~ ^{at} the beginning,
100 elements.

[4]

⇒ we will shift all the elements
to the right, obviously this will
take more time.

take a linked list,

→ sequence of nodes connected
with each other.

head
pointer
[1 | 2000] → [2 | 3000] → [6 | null]
1000 2000 3000

head
pointer
[4 | null]
1000

Conclusion:

inserting an element at the
beginning of an ~~array~~ linked list
is way faster than inserting in
the array.

logarithmic time complexity.

$$\log_2(8) = 3$$

How many times to multiply 2 in order to get 8.

iter 1 initially $i=1$.

$$\text{for } (i=1; i \leq n; i++)$$

$$i = 1 \times 2;$$

$$i = 2$$

$$i = 4$$

$$i = 8$$

$$i = 16$$

$$i = 32$$

$$i = 64$$

$$i = 128$$

$$i = 256$$

$$i = 512$$

$$i = 1024$$

$$i = 2048$$

$$i = 4096$$

$$i = 8192$$

$$i = 16384$$

$$i = 32768$$

$$i = 65536$$

$$i = 131072$$

$$i = 262144$$

$$i = 524288$$

$$i = 1048576$$

$$i = 2097152$$

$$i = 4194304$$

$$i = 8388608$$

$$i = 16777216$$

$$i = 33554432$$

$$i = 67108864$$

$$i = 134217728$$

$$i = 268435456$$

$$i = 536870912$$

$$i = 1073741824$$

$$i = 2147483648$$

$$i = 4294967296$$

$$i = 8589934592$$

$$i = 17179869184$$

$$i = 34359738368$$

$$i = 68719476736$$

$$i = 137438953472$$

$$i = 274877906944$$

$$i = 549755813888$$

$$i = 1099511627776$$

$$i = 2199023255552$$

$$i = 4398046511104$$

$$i = 8796093022208$$

$$i = 17592186044416$$

$$i = 35184372088832$$

$$i = 70368744177664$$

$$i = 140737488355328$$

$$i = 281474976710656$$

$$i = 562949953421312$$

$$i = 1125899906842624$$

$$i = 2251799813685248$$

$$i = 4503599627370496$$

$$i = 9007199254740992$$

$$i = 18014398509481984$$

$$i = 36028797018963968$$

$$i = 72057594037927936$$

$$i = 14411518807585968$$

$$i = 28823037615171936$$

$$i = 57646075230343872$$

$$i = 115292150460687744$$

$$i = 230584300921375488$$

$$i = 461168601842750976$$

$$i = 922337203685501952$$

$$i = 1844674407371003904$$

$$i = 3689348814742007808$$

$$i = 7378697629484015616$$

$$i = 14757395258968031232$$

$$i = 29514790517936062464$$

$$i = 59029581035872124928$$

$$i = 118059162071744259856$$

$$i = 236118324143488519712$$

$$i = 472236648286977039424$$

$$i = 944473296573954078848$$

$$i = 1888946593147908157696$$

$$i = 3777893186295816315392$$

$$i = 7555786372591632630784$$

$$i = 15111572745833265261568$$

$$i = 30223145491666530523136$$

$$i = 60446290983333061046272$$

$$i = 120892581966665122092544$$

$$i = 241785163933330244185088$$

$$i = 483570327866660488370176$$

$$i = 967140655733320976740352$$

$$i = 1934281311466641953480704$$

$$i = 3868562622933283906961408$$

$$i = 7737125245866567813922816$$

$$i = 15474250491733135627845632$$

$$i = 30948500983466271255691264$$

$$i = 61897001966932542511382528$$

$$i = 123794003933865085022765568$$

$$i = 247588007867730170045531136$$

$$i = 495176015735460340090662312$$

$$i = 990352031470920680181324624$$

$$i = 1980704062941841360362649248$$

$$i = 3961408125883682720725298496$$

$$i = 7922816251767365441450596992$$

$$i = 15845632533534730882901193888$$

$$i = 31691265067069461765802387776$$

$$i = 63382530134138923531604775552$$

$$i = 126765060268277847063209551104$$

$$i = 253530120536555694126419102208$$

$$i = 507060241073111388252838204416$$

$$i = 1014120482146222776505676408832$$

$$i = 2028240964292445553011352817664$$

$$i = 4056481928584891106022705635328$$

$$i = 8112963857169782212045411270656$$

$$i = 16225927714339564424090822541312$$

$$i = 32451855428679128848181645082624$$

$$i = 64903710857358257696363290165248$$

$$i = 129807421714716515392726580320496$$

$$i = 259614843429433030785453160640992$$

$$i = 519229686858866061570906321281984$$

$$i = 1038459373717732123141812642563968$$

$$i = 2076918747435464246283625285127936$$

$$i = 4153837494870928492567250570255872$$

$$i = 8307674989741856985134501140511744$$

$$i = 16615349879483713970269022281023488$$

$$i = 33230699758967427940538044562046976$$

$$i = 66461399517934855881076089124093952$$

$$i = 132922799035869711762152178248187808$$

$$i = 265845598071739423524304356496375616$$

$$i = 531691196143478847048608712992751232$$

$$i = 1063382392286957694097215425985502464$$

$$i = 2126764784573915388194430851971004928$$

$$i = 4253529569147830776388861703942009856$$

$$i = 8507059138295661552777723407884019712$$

$$i = 17014118276591323105555446815768039424$$

$$i = 34028236553182646211110893631536078848$$

$$i = 68056473106365292422221787263072157696$$

$$i = 136112946212730584844443774526144315392$$

$$i = 272225892425461169688887549052288630784$$

$$i = 544451784850922339377775098104577261568$$

$$i = 1088903569701844678755550962089554523136$$

$$i = 2177807139403689357511101924179109046272$$

$$i = 4355614278807378715022203848358218092544$$

$$i = 8711228557614757430044407696716436185088$$

$$i = 17422457115229514660088815393432872370176$$

$$i = 34844914230459029320177630786865744740352$$

$$i = 69689828460918058640355261573731489480704$$

$$i = 139379656921836117280710523147462889764088$$

$$i = 278759313843672234561421046294925779528176$$

$$i = 557518627687344469122842092589851559056352$$

$$i = 111503725537468893824568418517903118112704$$

$$i = 223007451074937787649136837035806236224408$$

$$i = 446014902149875575298273674071612472488176$$

$$i = 892029804299751150596547348143224944976352$$

$$i = 178405960859950230119309469628644988953204$$

$$i = 356811921719850460238618939257289777906408$$

$$i = 713623843439700920477237878514579555812816$$

$$i = 142724768687850180855447775702959111625632$$

$$i = 285449537375700361710895551405819223251264$$

$$i = 570899074751400723421791102811638446502528$$

$$i = 114179814950280144684358220562327689305056$$

$$i = 228359629900560289368716441124655378610112$$

$$i = 456719259801120578737432882249310757220224$$

$$i = 913438519602241157474865764498621514440448$$

$$i = 182687703920448231494973152899724302880896$$

$$i = 365375407840896462989946305799448605761792$$

$$i = 730750815681792925979892611598897211523584$$

$$i = 146150163136358585195978522319779422306768$$

$$i = 292300326272717170391957044639558844613536$$

$$i = 584600652545434340783914089279117689227072$$

$$i = 1169201305090868681567828178558235378444144$$

$$i = 2338402610181737363135656357116471556888288$$

$$i = 46768052$$

If - else construct.

```
scanf("%d", &n);  
if (n == 0)
```

```
{
```

```
    // scanner,  
}
```

```
else
```

```
{
```

```
    for (i = 1, sum = n, i++)
```

```
        // scan,
```

```
}
```

\Rightarrow worst case running time =

test + if part or else part.
(whichever is longer)

for "if" part :

$n = 0$ will take a

constant amount of time.

assumption, ~~constant~~ inside
statement will also take constant
time.

total time = $1 + 1 = \Theta(1)$

for "else" part :

$n = 0$, will take constant time

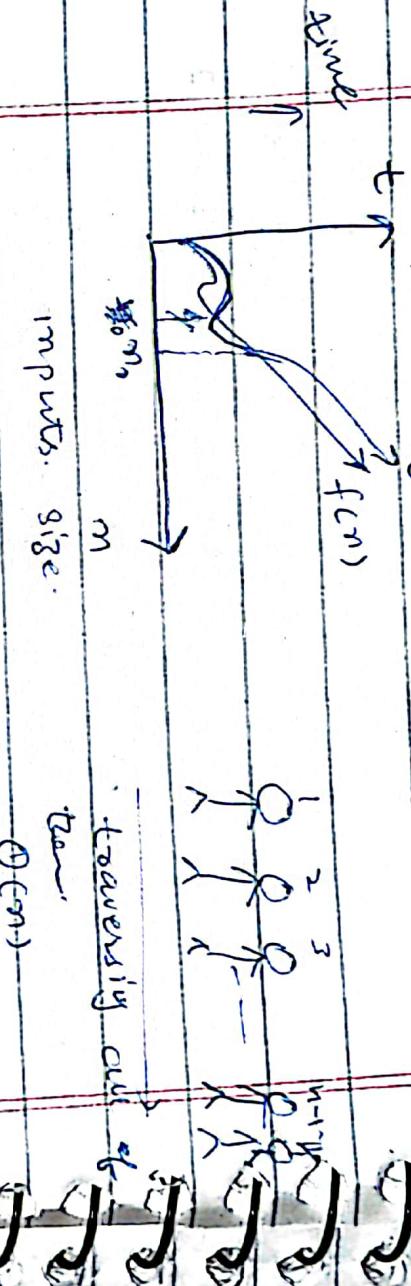
time $\in 1 + n \geq \Omega(n)$

Show upper bound: f is a function.

Big(O) \rightarrow Asymptotic Notations:

It's a mathematical way
of representing one time
worst case complexity of the program
or algorithm without being
executed.

$$c \cdot g(n) \rightarrow \text{upper bound}$$



Inputs size: $n^{1.5}$

$$f(n) \leq c \cdot g(n)$$

where $n \geq n_0$.

$$c > 0, n_0 \geq 0$$

Let's see

$$f(n) = 2^n + 3$$

$$2^n + 3 \leq \frac{5}{3} \cdot 2^n \quad \therefore f(n) = O(2^n)$$

$$\downarrow \quad \quad \quad \downarrow$$

$$f(n) \quad c$$

unction.

$$1 < \log n < \sqrt{n} < n < \log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots$$

lower bound range
upper bound.

$$f(n) = 2n + n \Rightarrow f(n) = O(n)$$

$$\text{Dominant value } 2n^2 + n \leq C \cdot g(n) \xrightarrow{\substack{2n+3 \leq 2n^2 \\ n \in O(n)}}$$

$$2n^2 + n \leq 3n^2 \rightarrow$$

$$n \leq n^2$$

$$2n+3 \leq 2n^2 \xrightarrow{n \geq 0} O(n)$$

~~O(n^2)~~

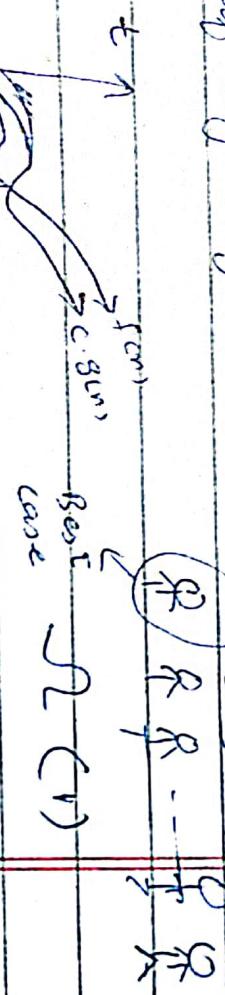
$$f(n) = O(g(n))$$

$$1 \leq n / n \geq 1$$

averaging out

Big-Omega (Ω) / best case

Ω is used to find best case of any algorithm.



$$f(n) = O(n)$$

$$f(n) \geq c \cdot g(n)$$

$$n_0, n_0$$

$$c > 0, n_0 > 1$$

$$f(n) = \Omega(g(n))$$

Q

$f(n) \geq 0.8(n)$

$$2^n + 3 \geq 1 \cdot n$$

\downarrow
 \downarrow
 \downarrow

$f(n) = \Theta(n)$

$f(n)$

$$2^n + 3 \geq 1 \cdot \log(n)$$

$\Sigma(\log n)$

