Design pattern

(*) **Design Pattern:** এটা হলো বারবার use করা যায় এমন একটা solution যা software development এর common problem গুলোকে solve করে,

(*) **Advantage**

→ reusable and can be used in multiple project
→ provide template solution
→ well tested and proven means of developing robust solution effortlessly

(*) **3 types**

(1) **Creational pattern:** during software development, তখন object কে কিভাবে create করবো, তাদের requirement এর ভিত্তিতে " " " " ",
creation logic hidden থাকবে, user এর কাছে এটা কি করে,

(2) **structural:** যেসব object আগের create করা, আমরা কিভাবে এদের combine করতে পারি। এবং obj এর feature অন্য obj use করতে চায়, তাদেরকে combine করে একটা single obj এর ভিতরে composition করতে পারি তার idea. inheritance use করে

(3) **Behavioral:** assemble না করে objects এর মধ্যে communication করানো

Creational

i) Singleton:
ii) Factory
iii) Abstract Factory
iv) Prototype
v) Builder

Structural

i) Adapter ii) composite iii) Proxy iv) Fly weight
v) Facade vi) Bridge vii) Decorator

Behavioral

i) Template method ii) Medicator iii) Chain of responsibility
iv) observer v) Strategy vi) command vii) State
viii) Visitor ix) iterator x) interpreter xi) Memento

* **Factory method**: user কে interface provide করা হবে। user superclass এর কোন obj create করতে পারবে। কিন্তু superclass এর under এ যে subclass আছে তারা এই obj creation alter করতে পারবে।

* **Abstract**: user এর যদি obj পরিবর্তন হয়, abstract factory obj create করে দেয় কিন্তু class এর inside এ কি ধরনের logic কাজ করতেছে বা obj কিভাবে create হয়েছে সেটা define করে দিবেনা।

* **Builder**: যখন কোনো complex obj create করা দরকার হয় তখন এই complx obj কে ছোট ছোট part এ divide করে দিবে। এবং প্রতিটি part কে integrate (create করে) করে user কে দিবে।

* **Prototype**: similar kind of obj creation এর জন্য

* **Object pool**: যখন performance boost করা লাগে বা obj creation অনেক costly

✱ <u>Singleton:</u> যা আমাদের ensure করে যে, আমরা যে class create করবো (singleton class), ঐ class এর only একটা instance create করা সম্ভব অর্থাৎ চাইলেও ঐ class এর multiple obj create করা যাবেনা। আর এটা access করার জন্য একটা global access point দেয়া থাকে, যাতে যেকোনো client এটা access করতে পারে।

✱ why we need singletone?

=> ধর, single DB connection ~~share করা আছে~~ এর multiple obj create করা আছে। ঐ ~~obj~~ connection obj গুলোর সাথে share করা আছে, তো যত obj create করবো হবে তত seperate DB connection create হবে, যা অনেক costly। এ class টা create হবে, DB এর connection share করে দিলাম, কিন্তু multiple obj create হতে দিবোনা। আমি একটা single obj create করে রাখবো। এটা সবাই access করবে ~~এটা সবাই~~ কেউ চাইলেও multiple obj create করতে পারবেনা, যাতে multiple database create না হয়।

<u>কিভাবে create করা যায়?</u>

① default যে constructor আছে তাকে private রাখতে হবে। কারণ - যখন আমরা new দিয়ে obj create করবো তখন default constructor automatically call হয়ে যায় এবং def " " এর কারণে নতুন obj ও create হয়ে যায়। এখন আমি যদি constructor কে private করে রাখি, তাহলে চাইলেও constructor কে call করা যাবেনা, new দিয়ে obj create করা যাবেনা। So private constructor করা হয়।

(11) singletone class এর মধ্যে static obj create করে রাখবো' এবং ঐ static obj কে access করতে পারবে শুধুমাত্র static method

~~stati~~ We know, static obj একের অধিক ~~use~~ access করতে পারিনা। অর্থাৎ static use করলে একের অধিক copy ঐ class এ create করা যায় না। তাই কোনো static obj কে access করার জন্য static method এর প্রয়োজন। তাই এটাই global access point

5 method to design singleton

- Classic implementation
- make getInstance() synchronized
- Eagen instantiation
- Use "Double checked Locking"
- By ENUM

(*) SOLID

S → Single Responsibility → প্রত্যেক class ~~...~~ একটি মাত্র functionality এর জন্য responsible. শুধু এবং শুধুমাত্র একটি কারণ থাকবে ঐ class কে change করার জন্য

O → Open closed principle →

L → LISKOV substitution

I → Interface segregation

D → Dependency Inversion

কেন use করবো?

=)

5→

O → Open for extension and closed for modification

project এ নতুন feature add করতে হচ্ছে। আমরা নতুন feature add এর জন্য নতুন code add করা লাগবে সেটাকে আমরা welcome জানাচ্ছি।

Existing code কে modify করা যাবেনা।

মানে আমরা entity র behaviour extend করতে পারবো without modifying anything in existing source code.

L ⇒ Child class, parent class এর definition & behaviour কে same ~~থাকতে~~ হবে।
যদি কোনো program কোনো base class কে use করে তাহলে derived class এর base class কে orginal implementation alter করা ছাড়া extend করতে পারবে।

I → আমাদের যে method প্রয়োজন নাই সে কোড implement করতে যাতে বাধ্য না করা হয়
client এর requirement অনুযায়ী আমরা যত ছোটখাট interface use করতে পারি কিন্তু একটা common interface use করার চেষ্টা করবো।

D → higher level module কোনো lower level module এর উপর নির্ভর করে। এই class অন্য একটি class এর উপর direct depend করে থাকবেনা। dependent হতে হলে abstraction এর মাধ্যমে হতে হবে।

SDLC → used by software industry to design, develop and test high quality software.

→ Planning -
→ Defining/Analysis → SRS (Software Requirement Specification)
→ Designing
→ coding/implementation
→ Testing
→ Deployment and maintainance

SRS: agreement between developer and customer যেখানে যে software develop করা হবে তার সমস্ত functional ও non-functional requirement কে agreement এ ~~করতে~~ উল্লেখ থাকবে

Planning: unique idea আসবে, এটা market এ implement করবে service provider এর সাথে formal/informal communication

Defining/Analysis: service provider যদি customer এর idea/requirement ভুল বোঝে, তাহলে ই requirement team এর মাঝে share করবে। এরপর এরা estimate করবে যে আমার এত cost লাগবে, এত stuff কত লাগানো লাগবে, এত টাইম লাগবে, এরপর customer কিভাবে cost, time এর কমবেশি করবে। এভাবে negotiation চলবে। এই negotiation এর পরে SRS document তৈরি হয় যা provider কে দেওয়া হয়।

Designing: Design related সব কাজ এখানে। SRS এর উপর depend করে software development এর ওই basic structure design করা হয়। Design and other documents, DDS (Design Document Specification) এ documented থাকে।

coding / implementation: কোন platform, কোন stack, backend, frontend এ কোন framework / language use করে DDS অনুযায়ী program code করা হয়।

Testing: SRS এর requirement meet না করা পর্যন্ত testing চলবে।

Unit testing: focuses on smallest unit of software design. done by programmer using sample input and observing it's corresponding output

Integration testing: output produce করার জন্য একাধিক group of components কে combine করা হয়।
top down, bottom up, sandwich, big bang

Black box: used for validation. internal mechanism কে ignore করে, output কি আসছে শুধু সেদিকে focus করে

white box: used for verification. internal mechanism এ focus করে, কিভাবে output টা আসছে তাতে focus করে।

Alpha: product customer এর কাছে release দেওয়ার আগে validation testing করা, QA people করে।

Beta: Limited number of user এর কাছে product released হয়, যারা করে তারা real-time environment এ test করতে পারে।

System: different OS এ যাতে কোনোভাবে কোড করে তার test. এটা black box testing এর under।

Deploy and maintainance : যখন product test করা হয়ে যায় তখন তা deploy করা হয় । এবং market এ release হওয়ার পর existing customer এর জন্য maintainance করা হয় ।

Model

1) Waterfall
ii) Spiral
iii) Rapid Application development
iv) Iterative
v) V-shaped
vi) Big bang
vii) Agile

Database

(*) DBMS : collection of program যা user কে DB create, maintain করতে সাহায্য করে

(*) Database: interrelated বা একাধিক data table/file এর সমষ্টি

2 types: 1) General → only 1 file on uninterrelated একাধিক file নিয়ে গঠিত DB
2) Relational → interrelated একাধিক file এর সাহায্যে যে DB

(*) elements in DB —
- → Field = column
- → record = row
- → value

(*) key field: যে field এর উপর ভিত্তি করে DB এর record খোঁজা, অনুসন্ধান, সম্পর্ক তৈরি করা যায়

3 type → primary, composite, foreign

(*) primary: যে field এর সাহায্যে প্রতিটি record কে সুনির্দিষ্টভাবে আলাদা করা যায়।

(*) composite: অনেক সময় একটি field এর সাহায্যে প্রতিটি record কে আলাদা করা যায়না। তখন 2 বা ততোধিক key field এর সাহায্যে composite key তৈরি করে প্রতিটি রেকর্ড আলাদা করা হয়

(*) foreign: এটি table এর primary key যদি অন্য table এর সাধারণ key হিসেবে used হয়,

DBMS এর element :

- → Data
- → Hardware & software
- → user
- → methodology

RDMS :

DBMS	RDMS
i) store data as file	i) as tabular form
ii) table গুলোর মধ্যে কোনো সম্পর্ক থাকবেনা	ii) সম্পর্ক থাকবে
iii) single user support করে	iii) multiple user support করে
iv) কম ডাটা নিয়ে কাজ করা হয়	iv) Large amount of data নিয়ে কাজ হয়
v) Microsoft access	v) MySQL, oracle, SQLite

Database relation:

দুই বা ততোধিক table এর মধ্যে যে পদ্ধতিতে সম্পর্ক স্থাপন করা হয় তার উপর query চালানো হয়।

One to one : কোনো একটি table এর একটি record এর সাথে অন্য table এর only একটি record এর সম্পর্ক স্থাপিত হয়।

One to many : কোনো একটি table এর একটি রেকর্ডের সাথে অন্য table এর একাধিক record এর সম্পর্ক স্থাপন করা হয়

Many to many : দুটি table এর মধ্যে একাধিক record এর matching থাকবে। এক্ষেত্রে extra আরেকটা table লাগবে। যাকে জংশন table বলে।

sorting: Data table এর রেকর্ডগুলোকে কোনো specific field অনুসারে সাজানো। sorted file এর ফলে extra space ~~table~~ create হয়, so extra space নেয়। কোনো রেকর্ড add করলে sort করা file update হয়না। তাকির সময় লাগে

indexing: পুরো table অপরিবর্তিত রেখে data table এর record গুলোকে specific field অনুসারে সাজানো। new data add করলে index file auto update হয়, তাই সময় 3 space লাগে।

Query: Database এ সংরক্ষিত কিছুল পরিমাণ তথ্য থেকে প্রয়োজনীয় তথ্য খুঁজে বের করাকে query বলে।

1) select query – Data table থেকে field বা col নির্বাচন করে যে query
2) parameter
3) crosstab – query করা data কে group আকারে উপস্থাপনের জন্য যে query
4) unmatched – 2টি টেবিলের unmatched data খুঁজে বের করা জন্য যে query
5) action – কোনো query র সাহায্যে দিয়ে table এর data যদি change করা হয়।

↓

4 types –

1) Make table query
2) Append
3) Delete
4) Update

6) 3টি query ভাষা

1) QUEL (Query Lang)
2) QBE (query By Ex)
3) SQL (Structured Query Language)

SQL statement

i) DML (Data Manipulation Lang)

- select
- insert
- update → update existing record
- delete → removes only rows in the table and struct remain same

ii) DDL (Data Definition Lang)

- create — Truncate (remove all row), rename
- Alter → used to add delet modify attribute of the relation in dB
- drop → delet all the data in the table and table struct

iii) DCL (Data control Language)

- Grant → user কে তোমার access সুবিধা দিতে চাইলে
- Revoke → " " ফিরিয়ে নেওয়া

iv) DTL (Data Transaction Lang)

- commit — তোমার data change করে স্থায়ীভাবে সংরক্ষণ করে
- Rollback — data র পরিবর্তন বাতিল করে আগের অবস্থায় ফিরে যায়
- savepoint — data র " কে নির্দিষ্ট point এ save করতে চাইলে

centralized: that is located, stored and maintained in single location

single player games. nfs, vice city (entire game in one system)

Decentralized: There is no central storage. কিছু server client দের information provide করে। serverগুলো একে অপরের সাথে connected. cryptocurrency, blockchain

Ex: Bitcoin. কোনো single organization bitcoin network কে own করে না। এই কারণে এটার সমস্ত লোকের মধ্যে, এখানে সবাই প্রত্যেক account নিজে নিজেদের bitcoin এর amount maintain করার জন্য

Distributed: No data storage. All nodes contain same information. The clients are equal and have equal rights

⇒ Google search system. Multiple computer working together
for user to complete a single task.

Design pattern

(*) **Design Pattern:** এর অনেক ধারণার use করে যায় এমন অনেক solution যা software development এর common problem গুলোকে solve করে,

(*) **Advantage**

→ reusable and can be used in multiple project
→ provide template solution
→ well tested and proven means of developing robust solution effortlessly

(*) **3 types**

(1) **Creational pattern:** during software development, আমরা object কে কিভাবে create করবো, আমাদের requirement এর ভিত্তিতে " " " " ",
creation logic hidden থাকবে, user এর কাছে এটা কি করে,

(2) **structural:** যেসব object আমরা create করি, আমরা কিভাবে এদের combine করতে পারি। এই obj এর feature অন্য obj use করতে চায়, তাদেরকে combine করে, একটা single obj এর ভিতরে composition করতে পারি এর idea. inheritance use করে

(3) **Behavioral:** assemble বা বড় objects এর মধ্যে communication করানো

Creational

i) Singleton :
ii) Factory
iii) Abstract Factory
iv) Prototype
v) Builder

Structural

i) Adapter ii) composite iii) Proxy iv) Fly weight
v) Fecade vi) Bridge vii) Decorator

Behavioral

i) Template method ii) Mediator iii) Chain of responsibility
iv) observer v) Strategy vi) command vii) State
viii) Visitor ix) iterator x) interpreter xi) Memento

* Factory method : user কে interface provide করা হবে। user superclass এর obj create করতে পারবে। কিন্তু superclass এর under এ যে subclass আছে তারা এই obj creation alter করতে পারবে।

* Abstract : user এর যদি obj দরকার হয়, abstract factory obj create করে দেয় কিন্তু class এর inside এ কি ধরনের logic কাজ করতেছে বা obj কিভাবে create হয়েছে সেটা define করে দিবেনা।

* Builder : যখন কোনো complex obj create করা দরকার হয় তখন এই complx obj কে ছোট ছোট part এ divide করে দিবে। এবং প্রতিটি part কে create করে integrate করে user কে দিবে।

* Prototype : similar kind of obj creation এর জন্য

* Object pool : যখন performance boost করা লাগে তখন obj creation অনেক costly

(*) singleton: যা আমাদের ensure করে যে, আমরা যে class create করবো (singleton class), ঐ class এর only একটা instance create করা সম্ভব অর্থাৎ চাইলেও ঐ class এর multiple obj create করা যাবেনা। আর এটা access করার জন্য একটা global access point দেয়া থাকে, যাতে যেকোনো client এটা access করতে পারে।

(*) why we need singletone?

=> ১ টা single DB connection ~~share করা আছে~~ এর: multiple obj create করা আছে। ঐ ~~obj~~ connection obj গুলোর সাথে share করা আছে, তো যত obj create করবো হবে তত seperate DB connection create হবে, যা অনেক costly। এ class টা create হলে, DB এর connection share করে দিলাম, কিন্তু multiple obj create হইতে দিবোনা। আমি একটা single obj create করে রাখবো। এটা সবাই access করবে ~~এটা সবাই~~ কেউ চাইলেও multiple obj create করতে পারবেনা, যাতে multiple database create না হয়।

কিভাবে create করা যায়?

(1) default যে constructor আছে তাকে private রাখতে হবে। কারণ - যখন আমরা new দিয়ে obj create করবো তখন default constructor automatically call হয়ে যায় এবং def " " এর কারণে নতুন obj ও create হয়ে যায়। এখন আমি যদি constructor কে private করে রাখি, তাহলে চাইলেও constructor কে call করা যাবেনা, new দিয়ে obj create করা যাবেনা। So private constructor করা হয়।

(11) singletone class এর মধ্যে static obj create করে রাখবো' এবং ঐ static obj কে access করতে পারবে শুধুমাত্র static method ~~stati~~ we know, static obj একের অধিক ~~use~~ access করতে পারিনা। অর্থাৎ static use করলে একের অধিক copy ঐ class এ create করা যাবেনা। তার যেকোনো static obj কে access করার জন্য static method এর প্রয়োজন। তার এটাই global access point

5 method to design singleton

- Classic implementation
- make getInstance() synchronized
- Eager instantiation
- Use "Double checked" Locking"
- By ENUM

(*) SOLID

S → Single Responsibility → প্রত্যেক class ~~কে~~ ১টি মাত্র functionality এর জন্য responsible; শুধু এবং শুধুমাত্র ১টি কারণ থাকবে ১টা class কে change করার জন্য

O → Open closed principle →

L → LISKOV substitution

I → Interface segregation

D → Dependency Inversion

কেন use করবো?

=>

S →

O → Open for extension and closed for modification

project এ নতুন feature add করতে হচ্ছে। আমরা নতুন feature add এর জন্য নতুন code add করা সাপোর্ট করে আমরা welcome জানাচ্ছি।

Existing code কে modify করা যাবেনা।

মানে আমরা entity র behaviour extend করতে পারবো without modifying anything in existing source code.

L ⇒ child class, parent class এর definition & behaviour কে same রাখবে।
যদি কোনো program কোনো base class কে use করে তাহলে derived class এর base class কে orginal implementation alter করা ছাড়া extend করতে পারবে।

I → আমাদের যে method প্রয়োজন নাই, সে কেনো implement করতে যাবে বাধ্য না করা হয়
Client এর requirement অনুযায়ী আমরা যত ধরনের তত interface use করতে পারি কিন্তু একটার common interface use করার যাবে।

D → higher level module গুলো lower level module এর উপর নির্ভর করে। এর class অন্য একটি class এর উপর direct depend হতে পারবেনা। dependent হতে হলে abstraction এর মাধ্যমে হতে হবে।

SDLC → used by software industry to design, develop and test high quality software.

→ Planning -
→ Defining/Analysis → SRS (Software Requirement Specification)
→ Designing
→ coding/implementation
→ Testing
→ Deployment and maintainance

planning & analysis
Designing
coding
testing
Deployment & maintanence

SRS: agreement between developer and customer যেখানে যে software develop করা হবে তার সমস্ত functional ও non-functional requirement যে agreement এ ~~করতে~~ উল্লেখ থাকবে

Planning: unique idea আসবে, এটা market এ implement হবে service provider এর সাথে formal/informal communication

Defining/Analysis: service provider যদি customer এর idea/requirement বুঝে নেবে, তারপর ই requirement team এর মাঝে share করবে। এরপর এরা estimate করবে যে আমার এত cost দরকার, এত stuff কত লাগানো লাগবে, এত টাইম লাগবে, এরপর customer নিজের cost, time এর কথা বলবে। এভাবে negotiation চলবে। এই negotiation এর পরে SRS document আসে যা provider কে দেওয়া হয়।

Designing: Design related যত কাজ হবে। SRS এর উপর depend করে software development এর ওই basic structure design করা হয়। Design and other documents, DDS (Design Document Specification) এ documented থাকে।

coding / implementation: কোন platform, কোন stack, backend, frontend এ কোন framework / language use করে DDS অনুযায়ী program code করা হয়।

Testing: SRS এর requirement meet না করা পর্যন্ত testing চলবে থাকে।

Unit testing: focuses on smallest unit of software design. প্রত্যেকটা individual unit test হয়। প্রত্যেক unit expected output দিয়ে ঠিক check করা, done by programmer using sample input and observing it's corresponding output

Integration testing: individual software module combined হয়ে output produce করার জন্য এরকম group এলাকার test করে of components কে combine করা হয়। group top down, bottom up, sandwitch, big bang

Black box: used for validation, internal mechanism কে ignore করে, output টি আসছে সেদিকে focus করে

white box: used for verification. internal mechanism এ focus করে, from a output পর্যন্ত কোথায় focus করে।

Alpha: product customer এর কাছে release হওয়ার আগে validation testing করা, QA people করে এগুলো

Beta: Limited number of user এর কাছে product released হয়, যারা কাজ করা real-time environment এ test করতে পারে.

System: different OS এ যাতে কোনোভাবে কোনো কাজ করে তার test. এটা black box testing এর under এ

Deploy and maintainance: যখন product test করা হয়ে যায় তখন তা deploy করা হয়। এবং market এ release হওয়ার পর existing customer এর জন্য maintainance করা হয়।

Model

i) Waterfall
ii) Spiral
iii) Rapid Application development
iv) Iterative
v) V-shaped
vi) Big bang
vii) Agile

ex) ecommerce

Planning & analysis: এ টিমে business analyst, product manager business stakeholder দের requirement gather করে যে কিভাবে ecommerce কাজ করা উচিত এবং কি কিভাবে দেখতে উচিত। এর পর তারা SRS document তৈরি হবে। Business analyst রা এ SRS review করে দিবে

Design: SRS document টা developer দের পাঠানো হবে। Developer গুলো SRS doc দিয়ে এবং requirement গুলো বুঝার try করবে। Designer রা web design করবে। Developer রা system architecture টা design করবে

Development: Develop web page, API's required to implement the functionality

Testing: ecommerce এ bug খুঁজে বের করা হয়েছে, ঠিক করো

Deploy & main: code deploy করা হবে এবং customer এর কাছে available করা হবে

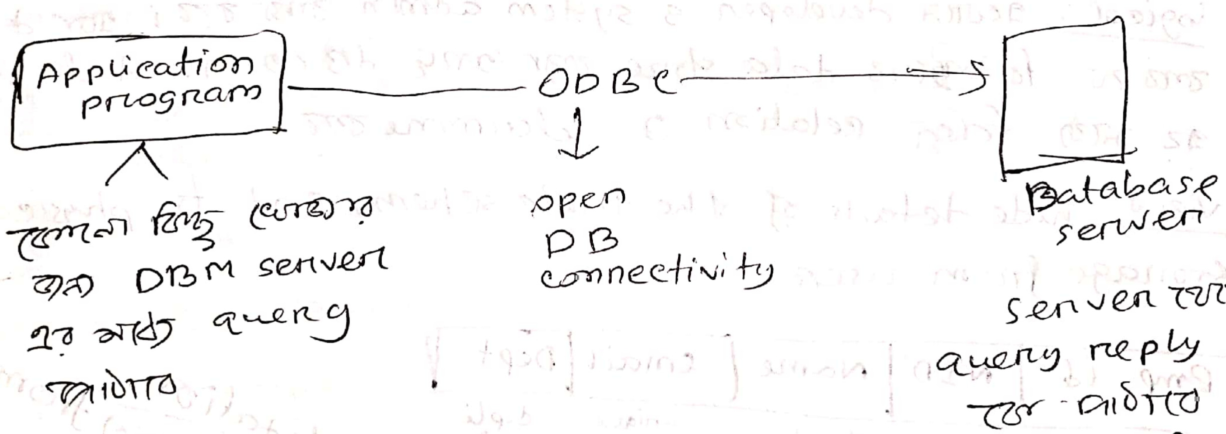
DBMS

DBMS architecture →

3 types → i) 1-tier
ii) 2- "
iii) 3- "

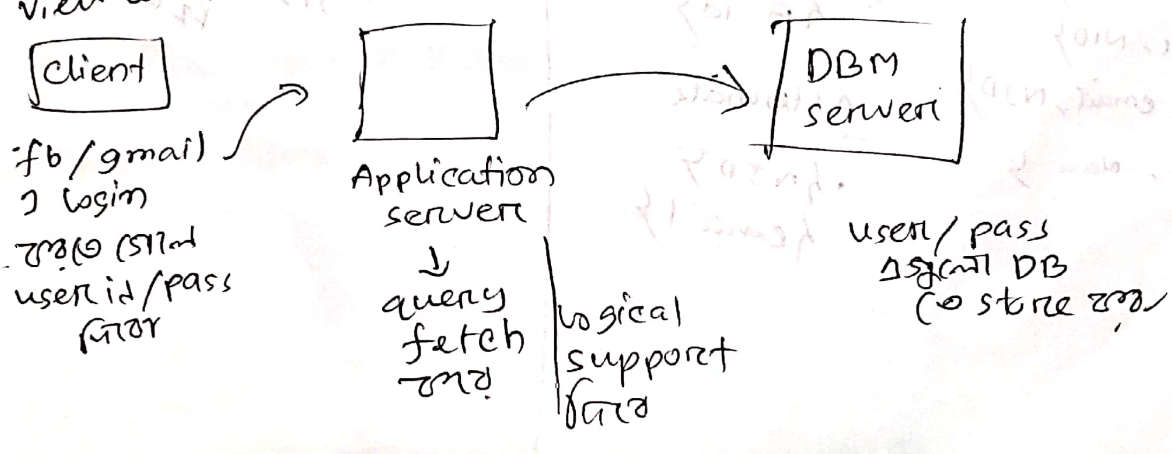
1 tier : client, server and DB all reside in the same machine.
PC এ microsoft word এ ছবি insert করি। এটা তো DB এ store হয়। i.e. DB টা আমার PC তেই আছে, এবং আমার store আমার গুলো সব PC তেই রাখতেছি। So এখানে layer একটাই আছে এটা 1-tier

2-tier : 2 টা layer. client layer, DBM server layer



Ex: Bank - এর program এ আমরা আমার acc no দিলাম, তো আমার সব info এখানে DB server এ আছে। এটা client side এ run হয়। সরাসরি DB এর সাথে interact করতে।

3-tier :



ACID

i) Atomicity – হয় সবগুলো change হবে, otherwise, successfully কোনোটাই change হবে না
ii) Consistent – before and after transaction. data কে consistent state এ থাকতে হবে
iii) Isolated – transaction যখন running থাকবে তখন অন্য কেউ data কে change করতে পারবে না, প্রভাবিত
iv) Durable – একবার হয়ে গেলে কোনো change আসবে না or স্থায়ী হবে

Data abstraction in DBMS

Physical : lowest level, managed by dbms. consist of data storage description and details of this level are hidden from system admin, developer and user

Logical : এখানে developer ও system admin কাজ করে। এটা determine করে যে কি ধরনের data store হবে এবং DB তে যে data point গুলোর মধ্যে কি ধরনের relation তা determine করে

View : hide details of the table schema and its physical storage from user

Emp_id	NID	Name	email	Dept
unique	unique	dupli	unique	dupli

Super key
{Emp id}
{NID}
{email}
{E_id, NID}
{E_id, email}
{email, NID}
{E_id, email, NID}
{E_id, Nam}

Candidate
{E_id}
{NID}
{email}

Primary
{E_id}

Alternate
{NID}
{email}

Normalization
reduce redundancy from table
row level
col level
↓
Insertion anomaly
Deletion "
Updation "

SID Sname CID Cname
FID Fname Salary

⊛ Super key: combination of all possible attributes যা uniquely তোমার row কে identify করতে পারে।

⊛ Candidate: তোমার এই attribute or set of attribute দিয়ে তোমার row কে uniquely identify করতে
→ minimal super key / যা Super key with no redundant attributes

⊛ Alternate := Candidate key - primary key

⊛ Unique key: is primary key with NULL value

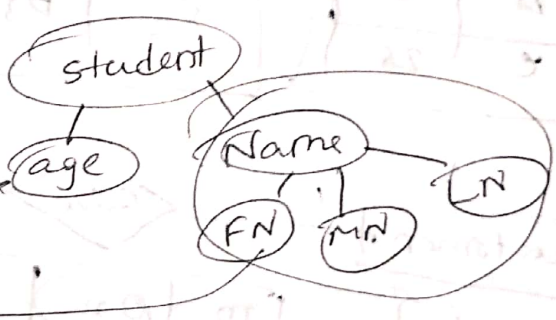
ER model.

Entity: Any ~~entity~~ object that has physical existance
student

Type of attribute

i) Single vs Multivalued

ii) Simple vs composite



iii) stored vs derived
↓ ↓
DoB age from DoB

iv) key vs Non key
(unique)

v) Required vs optional attribute

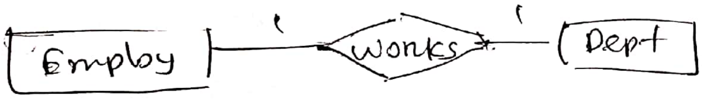
vi) Complex = composit + multivalued

One to one: দুটি entity র প্রত্যেকটির single instance relationship এ যুক্ত থাকে associated থাকে

One to many: more than one instance of an entity যদি relationship এ linked থাকে

Many one one:

Many to many: দুই দিক থেকেই instance on left and right যদি relationship এ linked থাকে



Eid	Ename	age
E1	A	20
E2	B	25
E3	A	24
E4	C	28

Eid	DID
E1	D1
E2	D2
E3	D3

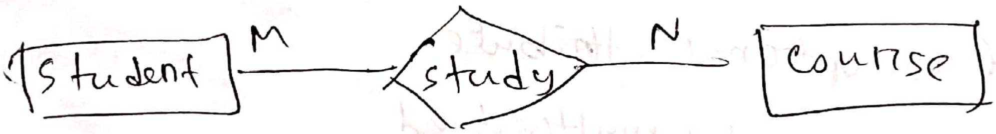
Did	Dname	Loc
D1	IT	Dhk
D2	Prod	Ctg
D3	HR	Raj

→ ১টি customer multiple order দিতে পারে। so, customer

ID	N	city
C1		
C2		
C3		

ID	O no	Date
C1	O1	
C1	O2	
C2	O3	
C3	O4	

O no	item	cost
O1		
O2		
O3		



complexities

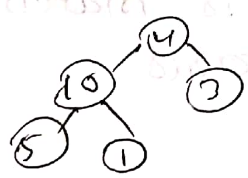
	WC	AC	BC	SC
insertion –	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
selection –	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Bubble –	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
Merge –	$O(n * \log(n))$	$O(n * \log(n))$	$O(n * \log(n))$	$O(n)$
Heap –	$O(n * \log(n))$	"	"	$O(1)$
Linear search –	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Binare –	$O(\log(n))$	$O(\log(n))$	$O(1)$	$O(1)$

Heap sort

→ given input থেকে heap build করা লাগবে

4 10 3 5 1

Build Heap



After building heap কে আবার maximum heap এ transform করা লাগবে।

max heap এ parent node ≥ child node

swap first and last node and delete the last node from heap

Merge : Divide and conquer algo

SQL

Select distinct → return distinct value

ORder by – sort result-set

Insert into – insert new record in table

Update – modify existing records in table

Delete – delete existing record in

Aliases – give a table or col. in a table a temporary name

Join – used to combine row from two or more table

innerjoin : ~~return records~~ both table এর matching values আছে এমন record return করে

left join : left table এর সব record return করে এবং right table এর matching record গুলো return করে

right :

Full outer : return all ~~এর~~ record যখনই কোনো match থাকে left or right table এ –

Union = 2 or more select statement এর result set কে combine করে

OOP

Class: It is a blueprint, that describes the details of object. Class এ যদি কোনো data বা function define করি, তখন ~~obj~~ ই class এর obj create করলে, ই obj automatically data/function গুলো নিয়ে আসে।

Object: instance of class. It has own state, behaviour & identity.

Encapsulation: data এবং code কে একটা single unit এ bind করা। মানে prog এর data member & method কে একসাথে bind করে, unnecessary details reveal করা ছাড়া কোনো specific কাজ করানোর প্রক্রিয়া হলো encapsulation.

```
Public class Person {
    private ~~string name;~~
    private: int age // data hidden
    void ~~get(int)~~ set(int a)
        age = a
    }
    void ~~talk()~~ get()
        return age
    }
}
```

2 ways
→ Data hiding
→ Data binding

```
Person P;
P.set(5)
cout<< P.get() <<endl
```

যখন আমি login করে email এ ঢুকতে যাব অনেক internal process backend এ কাজ করে যার কোনো প্রকার ধারণা আমাদের থাকেনা। যখন login করার জন্য আমরা পাসওয়ার্ড দিই, তখন তা বিভিন্ন উপায়ে encrypted হয়ে verify করে যে authentic কিনা। এরপর আমাকে account এর access দেয়া হয়।

Polymorphism: Already declare করা আছে এমন value or behaviour যদি subclass এও assign করা থাকে, তখন তা ---

many + form

২ প্রকার –

1) compile time / static / early binding / method overloading

এ সমস্ত event compile time এ occur হয়। যখন কোনো function কে call করা লাগবে এবং এর জন্য প্রয়োজনীয় information compile time এ জানা থাকলে তখন early binding occur হয়। compiler knows which function should call

```
class Overload
{
  public:
    void add(int a, int b)
    {
    }
    void add(int a, int b, int c)
    {
    }
    void add()
    {
    }
};

main()
{
   Overload obj;
   Obj.add();
   Obj.add(10,20)
   Obj.add(10,20,30)
}
```

method overloading

* parameter different
* occurs in same class
* inheritance এর প্রয়োজন নাই
* return type same or same না হতে পারে
* দুইটা method এর method কে hide করে

1) run time/ dynamic/ late binding (method overriding)

যে সমস্ত event runtime আসা পর্যন্ত resolve হয়না তাদের কে late Binding refer করে। late binding achieve করার জন্য virtual func use করা হয়।

কোন func call করা আসাবে বা কোন value নেয়া আসাবে তা execution এর উপর নির্ভর করবে।

```
class person
{ public:
   void display()
   {
   }
};
class student : public person
{ public:
   void display()
   {
   }
};
class teacher : public person
{ public:
   void display()
   {
   }
};
main(){
   Teacher t; Student s; Person P;
   P.display() s.(), t.()
```

overriding

* parameter same
* occur in sub class super class
* inheritance আছে
* return type same
* child method hides parent method

other way

```
Person * P
P = &S
P → display
```

→ এখানে person কে virtual করে

Inheritance : একটা class এর structure, behaviour সুবো যদি অন্য একটি class ও share করে তাহলে এরকম child class parent class কে properly acquire করে

```
class Person
{ public:
    string name;
    int age;
    void display1()
    { cout (Name)
      cout (age)
    }
}
class student: public person
{ public:
    int id;
    void display2()
    { cout (id)
      display1();
    }
}
```

- code reuseability
- less memory
- execution time কম

Type:

① single : এর child class এর parent class কে extend করে

```
Class A {
    }
class B : public A
{
}
```

(ii) Multilevel: class A extend class B, class B extend class C

```
class A {
  };
class B: public A{
};
class C: public B{
};
```

(iii) Hierarchical: Many child class extends one parent class

```
class A{
  };
class B : public A {
  };
class C : public A {
}
```

(iv) Multiple: এটি class যেটি more than one base class থেকে inherit করে। এটি child এর একাধিক parent.

(v)

Abstraction: Hiding the implemention details, user কে only functionality দেখাবে।

Ex: ATM, sending msg, dialer

```
class MobileUser () {                          ——→ Abstract class
  public
    virtual void sendMsg () = 0;               → pure virtual func
}

class Akib : public MobileUser {
    public :
        void sendmsg ()
        { cout ( Hi)
        }
};

class Rakib : public MobileUser {
    public :
        void sendmsg ()
        { cout << Hello)
        }
}
```

```
class User {
public
  virtual void sendmsg()=0;
}
class Akib : public user {
public:
  void send msg() {
    cout << hi <<
  }
```

```
main() {
    MobileUser *m    //abstract class এর pointer or reference
                     use করা যায়। object use করা যায় না

    Akib a;
    Rakib r;

    m = &r
    m → sendmsg(); ——→ (Hi)
```

~~AB~~ Abstraction	Encapsulation
① problem solved at interface level	① implementation level
② we can implement with abstract class and interface	② using access modifier

Errors ⇒ 2 types –

i) Compile time error : Run করার পূর্বে যে error দেখা যায়।
উদাহরণ: semicolon / {} missing

ii) Run time error : Run করার সময় যে error
উদাহরণ: int এর জায়গায় double/char

Exception handling: এটা একটা mechanism যা exception handle করতে
ব্যবহৃত হয় try, catch, throw

try: prog এর যে statement এ error আসার সম্ভাবনা থাকে তা try block এ রাখতে হয়

throw: যে problem ধরা পড়বে সে প্রব্লেমটি যে আমরা throw করবো.

catch: যে exception throw করা হবে তা catch করবো

```
main()
{ try {
      int n1, n2;
      cin >> n1, n2;
      if (n2 == 0)
          throw -1
      }
      double res = (double) n1 / n2;
      cout << res << endl;
  } if n2 == 0
}
catch (int x) {
      cout << "is not possible" << endl;
}
```

Friend class : allowed to access public, private, protected data, in same class

```
class A {
    private :
        int id = 101
        string name = "Akib";
    public :
        friend class B;
};
class B {
    public :
        void display (A ob) {
            cout << ob.id << endl;
            cout << ob.name << endl;
        }
};
main () {
    A ob1, B ob2
    ob2.display(ob1);
}
```

Virtual function

```
Class A {
    public :
        void display() {
            cout("Hi from class A")
        }
};
class B : class A {
    public :
        void display () {
            cout ("Hi from class B)
        }
};
```

```
main () {
    A *a
    B b
    a = &b
    a → display ()
}
```

এখানে a→display (class A) return করবে। কারণ এখানে early binding হচ্ছে। যখন compile করি, compiler সবকিছু ready করে, পরে কোন লাইনে jump করতে হবে সে জানে। early binding হলে compiler doesn't care about address pointer.

so, a→ (a) = (&b) → যার মান

class A show করবে।

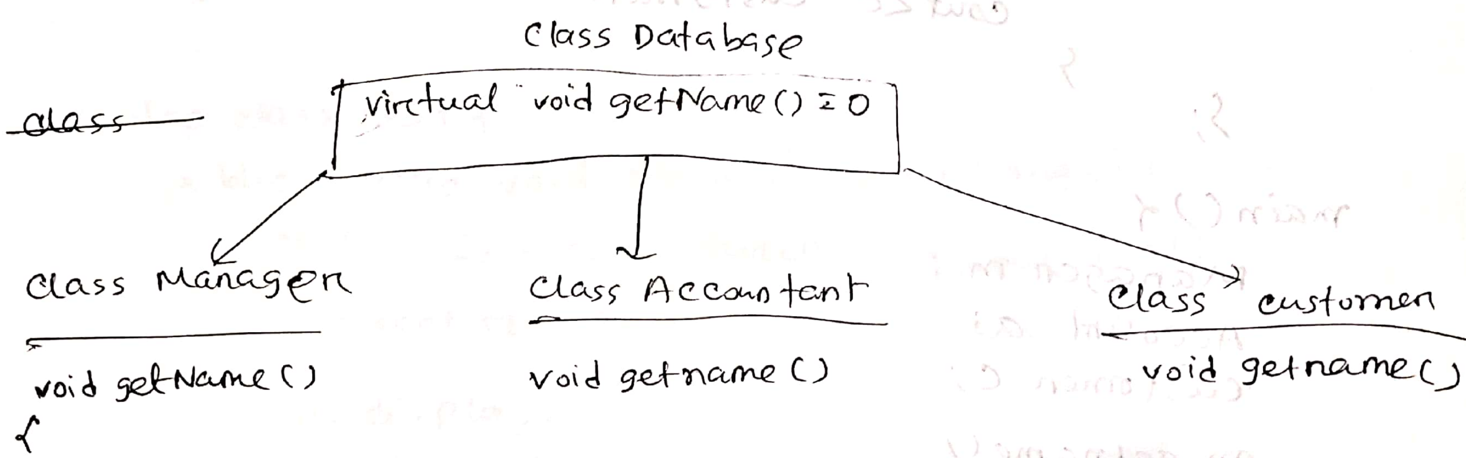
so, আমরা যদি class B return করাতে চাই, late binding করা লাগবে। so virtual add করে আমরা তা করতে পারবো।

abstract class: A class that contain pure virtual func. is called abstract class. can't create object. used as base class for other derived class.

pure virtual:

```
public:
    virtual void function () = 0; // No definition
```

আমি এ func এ class এ use করতে পারবো না। আমাকে এ class সকল derived class এ use করতে হবে।



```
class Database {
  public:
    virtual void getname() = 0;
};
class Accountant : public Database {
  public:
    void getname() {
        cout << "accountant"
    }
};
class Managen : public Database {
  public:
    void getname() {
        cout << "managen"
    }
};
class customer : public Database {
  public:
    void getname() {
        cout << "customer"
    }
};
main() {
    Manager m;
    Account a;
    Customer c;
    m.getname()
    a.   "   ()
    c.   "   ()
}
```

* Static keyword use করে memory management করা যায়।
static variable তৈরি করলে obj ছাড়া সরাসরি নাম দিয়ে class এ access করা যায়।

Package static_keyword

```
public class Student {
    String name;
    int id
    static string uniName = "LU";

    Student(string n, int i)
    {
        name = n;
        id = i;
    }
    void display(){
        System.out.println("Name : "+name)
                          ("Id : "+id);
                          ("Uni : "+uniName);
    }

public class test{
    public static void main (string[] args){
        Student s1 = new student("Anisul",101);
        student s2 = new      ("Rakib", 102);

        s1.display()
        s2.display()
}
```

student এর uniname সবার university একসাথে

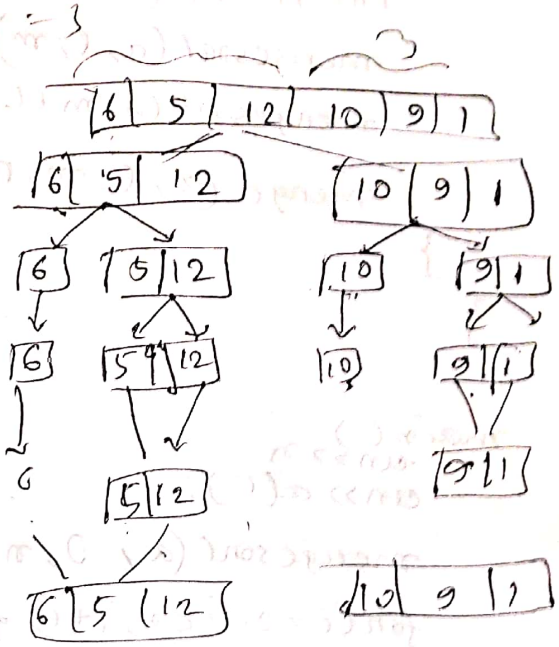
```
void merge(int a[], int l, int mid, int r)
{
    int n1 = mid - l + 1
    int n2 = r - mid;
    int left[n1], right[n2]
    for(i=0; i<n1; i++)
        left[i] = a[l+i]
    for (j=0; j<n2; j++)
        right[j] = a[mid+1+j]

    int i, j, k;
      i=0 = j=0, k = l;

    while (i<n1 and j<n2){
        if(left[i] <= right[j]){
            a[k] = left[i]
            i++;
        }
        else{
            a[k] = right[j]
            j++;
        }
        k++;
    }
    while (i<n1){
        a[k] = left[i]
        i++;
        k++;
    }
    while (j<n2){
        a[k] = right[j]
        j++
        k++;
    }
}
```

p = l

q = mid

r2 = 0 + 1 = 3 r = r

5 - 2 = 3



```
void mergesort (int a[], int l, int r)
{
    if (l<r){
        int m = l+ (r-l)/2;
        mergesort (a, l, m);
        mergesort (a, m+1, r);
        merge (a, l, m, r);
    }
}

main()
{
    cin>>n
    cin>> a[i];
    mergesort (a, 0, n-1);
    for(i=0; i<n; i++) {
        cout<< a[i];
    }
```

Heap sort

```
void heapify (int a[
{
    int largest = i
    int left = 2*
    int right = 2*

    if (left<n and
        largest =

    if (right<n
        larges

    if (largest !

        int temp =
        a[i] = a
        a[largest

        heapify
    }
}

void heap sort (i
{
    for (int =n
    {
        heapify
        for (i=n

            int
            a[
            a[

            hea
        }
    }
```

Heap sort

```
void heapify (int a[], int n; int i)
{ int largest = i  // initialize largest as root
  int left = 2*i+1 // left child
  int right = 2*i+2
  if (left<n and a[left] > a[largest])
      largest = left
  if (right<n and a[right] > a[largest])
      largest = right
  if (largest != i) {
      int temp = a[i];
      a[i] = a[largest]
      a[largest] = temp;
      heapify (a, n, largest)
  }
}
void heap sort (int a[], in n)
{ for (int = n/2 - 1; i>=0; i--)
  {  heapify(a, n, i);
     for (i=n-i; i>=0; i--) {
         int temp = a[0];
         a[0] = a[i]
         a[i] = temp
         heapify (a, i, 0)
     }
  }
```

Insertion

7 8 (5) 2 4 6 3

5 7 8 (2) 4 6 3

2 5 7 8 (4) 6 3

2 4 5 7 8 (6) 3

2 4 5 6 7 8 (3)

2 3 4 5 6 7 8

$O(n^2)$

$O(1)$

(1) select the first unsorted ...

Selection

64 25 12 22 (11)

i থেকে traverse করবো কোথায় minimum হয় দেখ, i কে select করে jth minimum এর সাথে swap করবো

11 25 (12) 22 64

11 12 (25) (22) 64

11 12 22 25 64

$O(n^2)$

Bubble

(5) 1 4 2 8 9

1 5 4 2 8 9

1 4 5 2 8 9

1 (4) 2 5 8 9

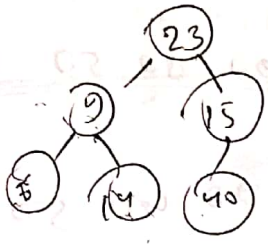
1 2 (4) (5) (8) (9)

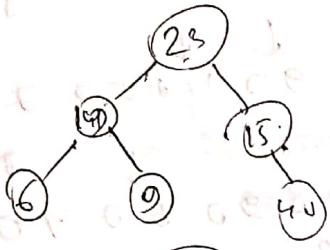
$O(n^2)$

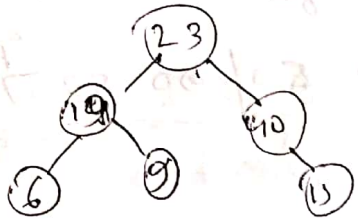
Heap sort

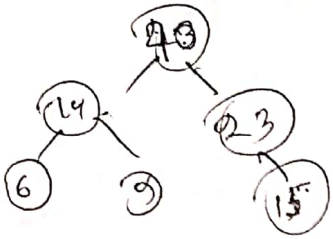
23 9 15 6 14 40

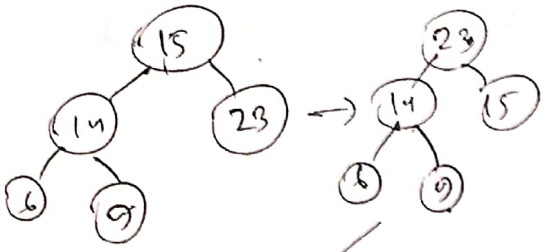
6 9 14 15 23 40

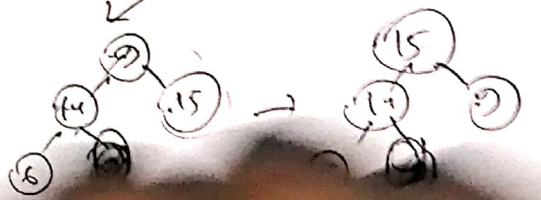








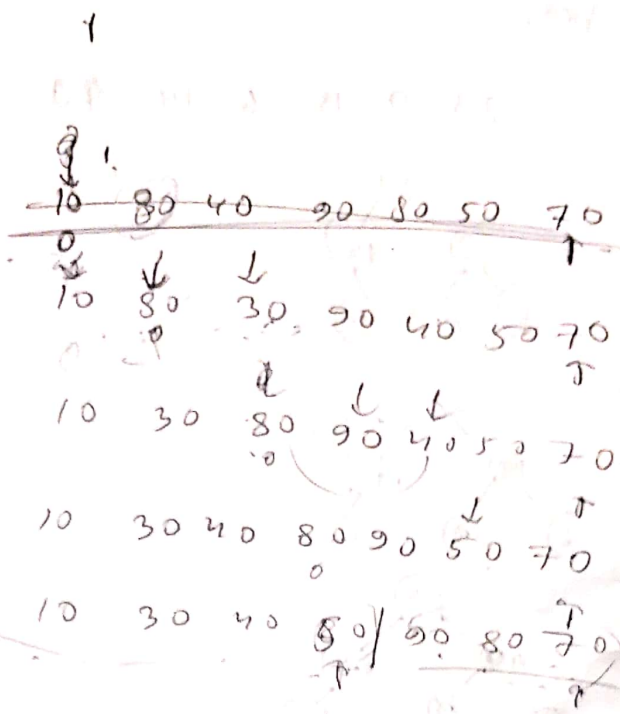




Quick sort

i = smaller elem
j = loop

```
i = -1
j = 0
for(j=0, i<j
if (a[j] <= pivot)
    i++
    swap (a[i], a[j])
```




```
int stack[size], top = -1
void push (int x)
    if (top >= size-1) overflow
    else stak[++top] = x;
pop ( ){
    if (top < 0) underflow
    else top--;
peek ( ){
    if (top<0) en
    else stack[top]
```

Queue

```
void que
}
  if (
  els
}
pop( )
cout<<a[
  en
```

web 1.0

read o
AJ
static

web 2.0

connect

90 80 50 70

90 40 50 70

90 40 50 70

80 90 50 70

80 90 80 70

Queue

front = 0; rear = -1

```
void que (int 'x')
 }
   if (rear = size) queue full
   else {
        que [++rear] = x
        cnt++;
      }

pop( )
 cout<<a [ifront++])
     cnt--
```

web 1.0

read only, কারও information share করা যেতো
এবং শুধু information connect করা যেতো
static content থাকতো।

web 2.0

read and write, interaction করা যেত, People
connect করতে, two way communication, face--.