Binary Search Tree (BST)

- A BST is a binary tree where:
    - The left subtree contains nodes with values less than the parent.
    - The right subtree contains nodes with values greater than the parent.
- The order in which values are inserted affects the shape of the tree.

AVL Tree (Adelson-Velsky and Landis Tree)

- A self-balancing BST that maintains the AVL balance property:
    - The balance factor of a node is calculated as:
        - Balance Factor = height(left subtree) - height(right subtree)
    - A node is balanced if the absolute value of its balance factor is ≤ 1.
- AVL trees ensure a balanced height, keeping operations efficient.

AVL Tree Rotations (Rebalancing)

An insertion can cause an imbalance in one of the following four ways. In each case, alpha (α) is the first unbalanced node.

Left-Left (LL) Case

- Occurs when a node is inserted into the left subtree of the left child of α.
- Structure before imbalance:

```
    α
   /
  y
 /
z
```

- Fix: Single Right Rotation around α.
    - Make y the new root.
    - Move α to the right of y.

Left-Right (LR) Case

- Occurs when a node is inserted into the right subtree of the left child of α.
- Structure before imbalance:

```
   α
  /
 y
  \
   z
```

- Fix: Double Rotation (Left Rotation + Right Rotation)
    - First, Left Rotate around y, converting it into an LL case.
    - Then, Right Rotate around α, making z the new root of the affected subtree.

Right-Left (RL) Case

- Occurs when a node is inserted into the left subtree of the right child of α.
- Structure before imbalance:

```
 α
  \
   y
  /
 z
```

- Fix: Double Rotation (Right Rotation + Left Rotation)
    - First, Right Rotate around y, converting it into an RR case.
    - Then, Left Rotate around α, making z the new root of the affected subtree.

Right-Right (RR) Case

- Occurs when a node is inserted into the right subtree of the right child of α.
- Structure before imbalance:

```
α
 \
  y
   \
    z
```

- Fix: **Single Left Rotation** around α.
    - Make **y** the new root.
    - Move **α** to the left of **y**.

Key Notes on AVL Rotations

- LL and RR cases require a single rotation.
- LR and RL cases require a double rotation.
- The case type refers to the position of the inserted node relative to the unbalanced node (α).
- AVL trees do not stand for anything specific – they are named after the inventors Adelson-Velsky and Landis.

Time Complexity of AVL Tree Operations

- Search: O(log n)
- Insertion: O(log n) (due to rebalancing)
- Deletion: O(log n) (may require rotations)

Hash Tables

Definition

- A hash table (also called a hash map) is a data structure that stores key-value pairs.
- It uses a hash function to map keys to indices in an array.
- Provides fast insert, delete, and lookup operations.
- Dispersion: spread of hash values (we want good dispersion)

Key Operations & Time Complexity

- Insertion: O(1) on average, O(n) in the worst case (due to collisions).
- Search: O(1) on average, O(n) in the worst case.
- Deletion: O(1) on average, O(n) in the worst case.

Hash Function

- A hash function converts a key into an index.
- It should:
    - Be fast to compute.
    - Distribute keys uniformly across the table (create dispersion).
    - Minimize collisions (different keys mapping to the same index).
    - Should include modulo division with the table size to ensure it fits within the bounds

Collisions

- A collision occurs when two keys produce the same index.
- Collision Resolution Strategies:

1. Chaining (Separate Chaining)

- Each index in the array stores a linked list (or another data structure) of key-value pairs.
- When a collision occurs, the new key-value pair is added to the list at that index.
- Pros: Simple, handles many collisions well.
- Cons: Increased memory usage due to linked lists.

2. Open Addressing

- Instead of using a linked list, all elements stay within the array itself.
- If a collision occurs, the algorithm searches for an open slot.
- Types of Open Addressing:
  - Linear Probing: If a slot is occupied, check the next slot (index + 1, wrap around if necessary).
  - Quadratic Probing: Check slots in a quadratic sequence (index + 1^2, index + 2^2, etc.).
  - Double Hashing: Use a second hash function to determine the step size for probing.
- Pros: More cache-efficient, no extra memory for linked lists.
- Cons: Can lead to clustering (many elements in the same region), reducing efficiency.

Load Factor (α)

- Load Factor = (number of elements) / (size of hash table).
- Determines when to resize the table (typically when $\alpha > 0.7$).
- Higher load factor → More collisions.
- Lower load factor → More wasted space.

Resizing & Rehashing

- When the load factor exceeds a threshold (e.g., 0.7), the table is resized:
  1. A new array (typically twice as large) is allocated.
  2. Each key is rehashed into the new array.
  3. The old array is discarded.
- Rehashing is costly (O(n)), but reduces future collisions.

Advantages of Hash Tables

- Fast lookups, insertions, and deletions (O(1) average case).
- Efficient memory usage for large datasets.
- Used in many applications (caching, databases, sets, dictionaries).

Disadvantages of Hash Tables

- Worst-case O(n) time complexity if too many collisions occur.
- Requires a good hash function to avoid inefficiency.
- Rehashing is expensive when resizing.

B+ Trees

- Optimized for disk-based indexing, reducing disk access operations.
- A B+ Tree is an m-way tree of order m, meaning:
  - Each node can have at most m keys.
  - Each node (except leaves) can have at most m+1 children.
- Internal nodes only store keys and pointers to children (no data).
- Leaf nodes store all data records and are linked as a doubly linked list for fast range queries.

B+ Tree Properties

- All nodes (except root) must be at least half full (at least $\lceil m/2 \rceil$ keys).

- Root node does not need to be half full (can have fewer keys initially).
- Insertion always happens at the leaf level.
- Leaf nodes are connected as a doubly linked list, enabling efficient range queries.
- Keys are always kept sorted within nodes.
- Splitting nodes:
    - If a leaf node splits, the smallest key in the right half is copied up to the parent.
    - If an internal node splits, the middle key is moved up to the parent.

B+ Tree Insertions

Case 1: Inserting Without Splitting

If the target leaf has space, insert the new key while keeping the order.

Before inserting 25:

```
Leaf Level:   | 10  20  30  40 |
```

After inserting 25:

```
Leaf Level:   | 10  20  25  30  40 |
```

Case 2: Leaf Node Split

If a leaf node is full, it must split into two.

- The smallest key in the right half is copied to the parent.
- The parent may also need to split if it becomes full.

Before inserting 35 (Leaf Node Full):

```
    [ 20  40 ]
   /    |    \
[10]  [20 30]  [40 50]
```

After inserting 35 (Leaf Split):

```
    [ 20 30 40 ]
   /     |     \
[10]  [20 25]  [30 35]  [40 50]
```

Case 3: Internal Node Split

If an internal node is full after an insertion, it splits, and the middle key moves up.

Before inserting 45 (Internal Node Full):

```
      [ 30 ]
     /     \
 [10 20]   [30 40 50]
```

After inserting 45 (Internal Split):

```
      [ 30 40 ]
     /    |    \
 [10 20] [30 35] [40 45 50]
```

B+ Tree Deletions

- Deletion happens at the leaf level.
- If a node has too few keys, it may borrow from a sibling.
- If borrowing is not possible, the node merges with a sibling, and the parent key is deleted.
- The root is the only node allowed to have fewer than [m/2] keys.

Advantages of B+ Trees

- Better disk efficiency: Nodes are designed to fit in memory pages.
- Fast range queries: Leaves are linked, enabling fast sequential access.
- More keys per node: Fewer levels compared to a binary search tree.
- Efficient insertions & deletions: Keeps balance with minimal restructuring.

AWS (Amazon Web Services)

- Leading cloud platform with 200+ services for computing, storage, databases, networking, AI, and more.
- Globally available through a network of regions (geographic areas) and availability zones (isolated data centers within regions).
- Uses a pay-as-you-use model, meaning you only pay for the resources you consume.
    - Cost-effective compared to traditional on-premise data centers, but costs can scale quickly if not managed properly.

AWS History & Growth

- Launched in 2006 with only S3 (Simple Storage Service) and EC2 (Elastic Compute Cloud).
- By 2010, AWS expanded with services like SimpleDB, EBS, RDS, DynamoDB, CloudWatch, CloudFront, and more.
- Early adoption incentives: Amazon ran competitions with large prizes to encourage developers to use AWS.
- Continuous innovation has led to 200+ services spanning operations, development, analytics, security, and AI.

AWS Global Infrastructure

- Regions: Geographic areas containing multiple data centers.
- Availability Zones (AZs): Isolated data centers within a region, offering redundancy and fault tolerance.
- POP (Points of Presence): Locations for Content Delivery Networks (CDN) to serve users faster (CloudFront uses these).

Cloud Computing Models

AWS supports the three main cloud service models:

1. Infrastructure as a Service (IaaS)
    - Provides virtualized computing resources (e.g., EC2, VPC, EBS).
    - User has the most control over configurations, networking, and OS.
2. Platform as a Service (PaaS)
    - Provides a managed platform for deploying applications (e.g., AWS Elastic Beanstalk, RDS).
    - AWS handles infrastructure, users focus on app development.
3. Software as a Service (SaaS)
    - Fully managed services where AWS handles everything (e.g., AWS Lambda, Amazon RDS, AI Services).
    - Users just interact with the software without worrying about infrastructure.

AWS Shared Responsibility Model

AWS operates under a shared security model, meaning security responsibilities are divided:

AWS Responsibilities (Security OF the Cloud)

- Protects physical infrastructure (data centers, power, networking).
- Manages host operating systems & hypervisors.
- Secures AWS-managed services (e.g., S3, RDS).

Customer Responsibilities (Security IN the Cloud)

- Controls data, encryption, and access management.
- Manages Identity & Access Management (IAM) roles and policies.
- Configures network security (e.g., security groups, VPN, firewalls).
- Ensures compliance with governance policies.

Core AWS Services

Compute Services

AWS provides multiple compute options:

1. VM-Based (Virtual Machines)
    - EC2 (Elastic Compute Cloud) – Virtual machines (VMs) with various instance types for different workloads.
2. Container-Based (Managed Containers)
    - ECS (Elastic Container Service) – Managed container orchestration for Docker.
    - EKS (Elastic Kubernetes Service) – Managed Kubernetes.
    - ECR (Elastic Container Registry) – Container image storage.
    - Fargate – Serverless container execution (no VM management).
3. Serverless Compute
    - AWS Lambda – Event-driven, serverless functions that scale automatically.

Storage Services

AWS offers various storage solutions based on needs:

1. Object Storage
    - Amazon S3 (Simple Storage Service) – Scalable, high-durability storage.
        - Supports time-limited upload links for secure file sharing.

        - Maximum file size: 5 TB.
    - Amazon Glacier – Low-cost archival storage.
2. Block Storage
    - Amazon EBS (Elastic Block Store) – Attachable disk storage for EC2 instances.
3. File Storage
    - Amazon EFS (Elastic File System) – Scalable, managed file storage for multiple EC2 instances.
    - Amazon File Cache – Cache storage for high-speed data access.
4. Backup & Recovery
    - AWS Backup – Centralized backup management.

Database Services
AWS provides both SQL and NoSQL database services:

1. Relational Databases (SQL-based)
    - Amazon RDS (Relational Database Service) – Managed SQL databases (MySQL, PostgreSQL, MariaDB, SQL Server, Oracle).
    - Amazon Aurora – MySQL/PostgreSQL-compatible, high-performance relational database.
2. NoSQL Databases
    - Amazon DynamoDB – Managed key-value database (similar to Redis).
    - Amazon DocumentDB – Managed NoSQL document database (MongoDB-compatible).
    - Amazon Neptune – Graph database service for complex relationships.
3. In-Memory Databases
    - Amazon ElastiCache – Managed Redis/Memcached caching for fast data retrieval.
    - Amazon MemoryDB – Fully managed Redis-compatible database.

Networking & CDN

- Amazon VPC (Virtual Private Cloud) – Isolated network for AWS resources.
- AWS CloudFront – Amazon's Content Delivery Network (CDN) (competes with Cloudflare).

Analytics Services
AWS provides data analytics tools for big data processing, real-time streaming, and business intelligence:

- Amazon Athena – SQL-based queries on S3 data.
- Amazon EMR (Elastic MapReduce) – Managed Hadoop & Spark for big data.
- AWS Glue – Serverless ETL (Extract, Transform, Load) service.
- Amazon Redshift – Cloud-based data warehouse.
- Amazon Kinesis – Real-time data streaming service.
- Amazon QuickSight – Business intelligence & visualization tool.

Machine Learning (ML) & AI Services

AWS provides managed AI/ML services:

1. Amazon SageMaker – End-to-end machine learning platform for building, training, and deploying ML models.
2. AWS AI Services – Pre-built AI models:
    - Amazon Comprehend – Natural language processing (NLP).
    - Amazon Rekognition – Image and facial recognition.
    - Amazon Textract – Extracts text from scanned documents.
    - Amazon Translate – Language translation service.

Why Use AWS?

- Scalability – Auto-scaling and load balancing allow seamless growth.
- Cost-Effectiveness – Pay only for what you use.
- Security – AWS follows the highest security standards.
- Flexibility – Supports multiple architectures (VMs, containers, serverless).
- Global Reach – Available across the world with high availability.

Final Thoughts

AWS is the most widely adopted cloud platform due to its scalability, flexibility, and rich set of services. Understanding core AWS services, pricing, security, and deployment models is essential for cloud computing professionals.