A group of cubes connected together

Description automatically generated

Task:03

Domain: Algorithm and Data-structure

Mehjabin Mostafa | Real-Time Processing with Divide and Conquer Algorithms in Big Data Applications

**Divide and Conquer:**

The idea of divide and conquer is to break the problem in to many subproblem until the sub problems are smaller enough to solve directly. It uses recursion to divide the problems again and again. After finding the answers to all the subproblems, they are combined or merge to produce the overall solution.

Dividing into sub problems and solving them at the same time allows parallel processing which is more efficient.

Examples:

* Merge Sort
* Quick Sort
* Karatsuba Algorithm for fast multiplication
* Strassen’s matrix multiplication
* Convex hull
* Quick hull

**Divide and conquer in Big Data:**

* Scalability
* Benefit of parallel processing
* Improved efficiency
* Fault tolerance
* Optimization of resources
* Application in big data algorithm(MapReduce)

Example:

1.merger sort with big data:

*#include <iostream>*

*#include <vector>*

*using namespace std;*

*void merge(vector<int>& data, int left, int mid, int right) {*

*int n1 = mid - left + 1; // Size of left subarray*

*int n2 = right - mid; // Size of right subarray*

*vector<int> leftArray(n1);*

*vector<int> rightArray(n2);*

*for (int i = 0; i < n1; i++)*

*leftArray[i] = data[left + i];*

*for (int j = 0; j < n2; j++)*

*rightArray[j] = data[mid + 1 + j];*

*int i = 0; // Initial index of left subarray*

*int j = 0; // Initial index of right subarray*

*int k = left; // Initial index of the merged array*

*while (i < n1 && j < n2) {*

*if (leftArray[i] <= rightArray[j]) {*

*data[k] = leftArray[i];*

*i++;*

*} else {*

*data[k] = rightArray[j];*

*j++;*

*}*

*k++;*

*}*

*while (i < n1) {*

*data[k] = leftArray[i];*

*i++;*

*k++;*

*}*

*while (j < n2) {*

*data[k] = rightArray[j];*

*j++;*

*k++;*

*}*

*}*

*void mergeSort(vector<int>& data, int left, int right) {*

*if (left >= right) {*

*return; // Base case: a single element or empty subarray*

*}*

*int mid = left + (right - left) / 2; // Find the midpoint*

*// Recursively sort each half*

*mergeSort(data, left, mid);*

*mergeSort(data, mid + 1, right);*

*merge(data, left, mid, right);*

*}*

*int main() {*

*// Example dataset*

*vector<int> data = {38, 27, 43, 3, 9, 82, 10};*

*cout << "Original Data: ";*

*for (int num : data)*

*cout << num << " ";*

*cout << endl;*

*mergeSort(data, 0, data.size() - 1);*

*cout << "Sorted Data: ";*

*for (int num : data)*

*cout << num << " ";*

*cout << endl;*

*return 0;}*

example: MapReduce (pseudo code):

*INPUT: List of text lines (text\_data)*

*OUTPUT: Word counts (global\_word\_count)*

*START*

*1. Load input text data (text\_data).*

*2. Divide text\_data into CHUNKS based on the number of threads (num\_threads):*

*- chunk\_size = ceil(size of text\_data / num\_threads)*

*- For each thread i:*

*Assign chunk[i] = text\_data[i \* chunk\_size to (i+1) \* chunk\_size]*

*3. Initialize local\_word\_counts[num\_threads] as an empty list of maps.*

*- Each map will store key-value pairs for words and their counts.*

*4. Mapper Phase:*

*For each thread i in [0, num\_threads):*

*Run MAPPER function on chunk[i] in parallel:*

*a. For each line in chunk[i]:*

*Split the line into words.*

*For each word:*

*Add word to local\_word\_counts[i] with count = count + 1.*

*5. Initialize global\_word\_count as an empty map.*

*6. Reducer Phase:*

*For each thread i in [0, num\_threads):*

*Run REDUCER function in parallel:*

*a. Lock access to global\_word\_count.*

*b. For each word in local\_word\_counts[i]:*

*Add word count to global\_word\_count.*

*c. Release lock.*

*7. Output Results:*

*For each word in global\_word\_count:*

*Print (word, count).*

*END*

***Mapper function:***

*MAPPER(chunk, local\_word\_count):*

*For each line in chunk:*

*Split the line into words.*

*For each word:*

*Clean word (remove punctuation, convert to lowercase).*

*Increment local\_word\_count[word].*

*Return local\_word\_count.*

***Reducer Function:***

*REDUCER(local\_word\_count, global\_word\_count):*

*Lock global\_word\_count for thread-safe access.*

*For each word in local\_word\_count:*

*Increment global\_word\_count[word] by local\_word\_count[word].*

*Unlock global\_word\_count.*

***Sample Input:***

*text\_data = [*

*"Hello world! This is a simple example of MapReduce.",*

*"MapReduce is designed for processing massive data.",*

*"Divide and conquer is the core of MapReduce.",*

*"Hello again, MapReduce!"*

*]*

***Output:***

*Word Counts:*

*again: 1*

*and: 1*

*conquer: 1*

*core: 1*

*data: 1*

*designed: 1*

*divide: 1*

*example: 1*

*hello: 2*

*is: 2*

*mapreduce: 4*

*massive: 1*

*of: 3*

*processing: 1*

*simple: 1*

*the: 1*

*this: 1*

*world: 1*

**Decision Tree:**

It’s a Machine learning algorithm that is used for classification and regression.it shows the decisions and their possible consequences in a tree-like or flowchart-like structure. Decision Tree can also be used in Data Mining, Statistics etc.

**Root node:**

Top node of the tree which represents the basic if whole data set and it splits based on information gain and impurity.

There are many more things in the decision tree. Internal node, branches and leaf node (final outcome of the DT) which holds a important part in a Decision Tree.

A diagram of a tree

Description automatically generated Fig: Structure of DT

**Advantages of DT:**

* Simplicity and Interpretability
* Versatility
* No Need for Feature Scaling
* Handles Non-linear Relationships

**Steps of finding root in DT:**

**Calculate Total Entropy**: Measure the dataset's uncertainty based on the target variable.

**Compute Information Gain**: For each attribute, calculate the reduction in entropy after splitting the dataset.

**Select Root Attribute**: Choose the attribute with the highest Information Gain as the root node.

A close up of text

Description automatically generated



**A math equations on a white background

Description automatically generated**



**Disadvantages of Decision Trees**

* Overfitting
* Instability
* Bias towards Features with More Levels

**Pruning**

To overcome overfitting, pruning techniques are used. Pruning reduces the size of the tree by removing nodes that provide little power in classifying instances. There are two main types of pruning:

* Pre-pruning (Early Stopping): Stops the tree from growing once it meets certain criteria
* Post-pruning: Removes branches from a fully grown tree that do not provide significant power.

**Applications of Decision Trees in Big Data**

**Fraud Detection**:

Detect anomalies or suspicious activities in massive transactional datasets. Example: Credit card companies identifying fraudulent transactions.

**Healthcare Analytics**:

Diagnose diseases or predict patient outcomes using medical data. Example: Decision support systems for personalized treatment recommendations.

**Financial Forecasting**:

Predict market trends or risks in large financial datasets. Example: Stock price prediction or credit risk assessment.

**Sentiment Analysis**:

Analyze customer reviews or social media data to classify sentiment. Example: Sentiment classification for brand monitoring.

**Manufacturing Optimization**:

Analyze production data to optimize processes or predict machine failures. Example: Predictive maintenance in industrial IoT.

**Environmental Monitoring**:

Classify environmental data for climate change research or disaster prediction. Example: Predicting forest fires or floods using sensor data.

**Educational Analytics**:

Predict student performance and suggest interventions using academic data. Example: Identifying at-risk students in massive online courses.

**Large Scale Numerical Calculation:**

**Definition:**

It relates to a big set of data where special algorithm and computational efficiency is counted. As the size of the dataset grows, the computational demands for processing and analyzing the data increase exponentially. Handling massive data sets needs efficient data management strategies and computational techniques to avoid bottlenecks and ensure timely results.

**Importance in Big Data:**

* + Data Analysis and Insights
  + Scalability
  + Optimization
  + Modeling and Simulations
  + Machine Learning and AI
  + High-Performance Computing (HPC) Integration
  + Decision-Making
  + Efficiency and Cost-Effectiveness.

**Real Life examples:**

**Weather Forecasting**

* Description: Predicting weather patterns involves processing data from millions of sensors, satellites, and radars.
* Numerical Techniques: Solving partial differential equations (PDEs) in numerical weather prediction models.
* Example: The European Centre for Medium-Range Weather Forecasts (ECMWF) runs global models to predict weather days or weeks ahead.

**Climate Modeling**

* Description: Simulating long-term climate trends, including global warming, ice cap melting, and ocean circulation.
* Numerical Techniques: Finite element methods and Monte Carlo simulations to model atmospheric and oceanic interactions.
* Example: NASA's Earth Exchange (NEX) performs simulations to study climate change impacts.

**Transportation and Logistics Optimization**

* Description: Optimizing delivery routes, supply chains, and air traffic management.
* Numerical Techniques: Linear programming and graph algorithms for shortest path optimization.
* Example: Amazon uses numerical calculations for real-time route optimization in delivery logistics.

**Space Exploration**

* Description: Planning trajectories for spacecraft and simulating space environments.
* Numerical Techniques: Solving orbital mechanics equations and fluid dynamics for spacecraft re-entry.
* Example: NASA's Perseverance Rover Mission used numerical calculations for landing on Mars.

**Genomics and Bioinformatics**

* Description: Analyzing DNA sequences, protein structures, and evolutionary patterns.
* Numerical Techniques: Parallel processing for genome assembly and alignment algorithms.
* Example: The Human Genome Project involved large-scale numerical calculations to sequence the entire human genome.

**Apache Spark:**

It is an open-source, distributed computing system designed for big data processing. It provides a fast, general-purpose, and scalable platform for handling large datasets across clusters of computers. Spark is widely used for tasks such as data processing, machine learning, stream processing, and more.

**Key Features of Apache Spark**

**Speed**

Spark performs in-memory computations, which make it up to 100 times faster than traditional disk-based processing systems like Hadoop MapReduce for certain tasks.It uses efficient query execution and advanced DAG (Directed Acyclic Graph) optimizations.

**Distributed Computing**

Spark splits data across a cluster of computers and processes it in parallel, making it highly scalable for large datasets.

**Unified Platform**

Spark supports multiple workloads with a single unified engine:

* + **Batch Processing**: Processes large datasets stored on disk.
  + **Stream Processing**: Handles real-time data streams.
  + **Machine Learning**: Offers a library called MLlib for scalable machine learning algorithms.
  + **Graph Processing**: Provides GraphX for graph computations.
  + **SQL Processing**: Uses Spark SQL for querying structured data.

**Ease of Use**

Spark supports APIs in popular programming languages like Scala, Python, Java, and R, making it easy to a wide range of developers.

It provides an interactive shell for quick experimentation and debugging.

**Integration**

Spark integrates with many big data tools and platforms, including:

* + **Hadoop**: Can run on Hadoop clusters and access HDFS (Hadoop Distributed File System).
  + **Kafka**: For streaming data.
  + **Hive**: To query data stored in Hive tables.
  + **Cassandra, HBase**: For NoSQL database access.

**Real life example:**

Apps and many other things that process big data and requires real- time analytics use Apache spark as an accessible tool.

 Real-Time Data Processing: Fraud detection, dynamic pricing

 Batch Processing: Process large datasets for analytics

 Machine Learning and AI: Train models on big data

 Graph Processing: Analyze graph data

 Data Integration: Integrate data from multiple sources

 Interactive Analytics: Run queries on large datasets

 Handling Semi-Structured Data: Process JSON, XML, and free-text data

 Genomic Data Processing: Analyze genomic data for research

But for small data sets or simple workflows and with limited computational resources do not need to use Apache spark. Because it uses

 In-Memory Computation

 Directed Acyclic Graph (DAG) Execution

 Parallel Processing

 Resilient Distributed Dataset (RDD)

 Advanced Query Optimization

 Columnar Storage Format (Parquet, ORC)

 Integration with Accelerators

 Built-in Libraries (MLlib, GraphX, Spark Streaming)

 Cluster Management and Resource Allocation

 Support for Data Partitioning

 Advanced Caching

**Apache spark in Numerical calculation:**

Apache spark runs numerical calculation through distributed computing(MapReduce in Hadoop splits large data processing), in memory(RAM) processing and parallel execution(Multiple tasks are executed simultaneously across different processors or cores).

**Time Complexity and Efficiency:**

**Time Complexity**

Time complexity measures how the running time of an algorithm increases as the input size (n) grows. It helps to understand how an algorithm will perform with large inputs. Time complexities are commonly expressed using **Big O notation**, which provides an upper bound on the algorithm’s growth rate.

* **O (1)**: Constant time. The algorithm’s execution time does not depend on the size of the input.
* **O (log n)**: Logarithmic time. The execution time increases logarithmically with input size (binary search).
* **O(n)**: Linear time. The execution time increases directly with the input size (iterating through an array).
* **O (n log n)**: Log-linear time. Common in efficient sorting algorithms like merge sort and quicksort.
* **O(n^2)**: Quadratic time. Execution time increases quadratically with input size (bubble sort).
* **O(2^n)**: Exponential time. The execution time doubles with each additional input element (brute-force solutions for combinatorial problems).
* **O(n!)**: Factorial time. The execution time grows extremely fast as input size increases, making algorithms with this complexity impractical for large inputs. ( Traveling Salesman Problem (TSP))

**Efficiency**

Efficiency means how well an algorithm uses resources such as time and memory to solve a problem. It includes time efficiency (how quickly it runs) and space efficiency (how much memory it uses).

* **Time Efficiency**: This is about reducing the time complexity of an algorithm because efficiency solely depends on time. If an algorithm uses more time than any other algorithm it will be counted as inefficient. For example, an algorithm with O(n!) time complexity is much less efficient than one with O (n log n) or O(n).
* **Space Efficiency**: The specific amount of memory that an algorithm uses. Efficient algorithms try to use as small memory as possible to solve any problem.

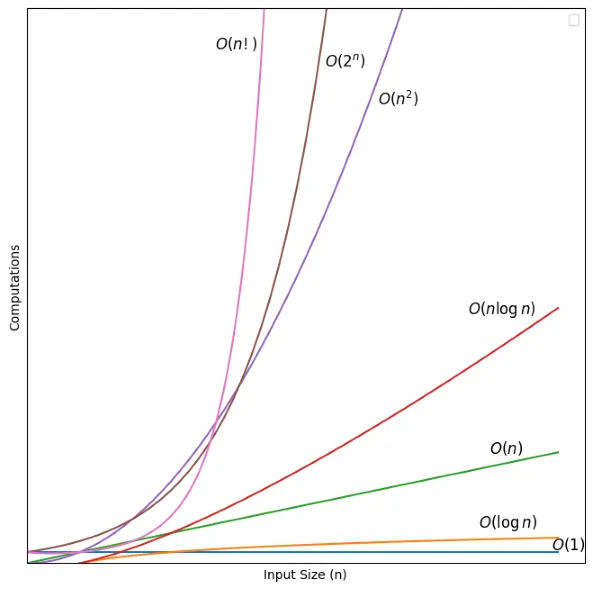


Fig: Comparison between different time complexity

A diagram of a function

Description automatically generated

Fig: comparison between different space complexity.

**Importance in big data:**

* Scalability: Ensures handling of growing datasets without performance degradation.
* Processing Speed: Enables timely analysis of massive data.
* Cost Reduction: Minimizes computational time and resources.
* Real-Time Analysis: Supports time-sensitive applications and insights.
* System Performance: Prevents slowdowns or crashes due to inefficiency.
* Energy Efficiency: Reduces power consumption for sustainability.
* Improved Decision-Making: Facilitates faster insights for critical decisions.
* Avoiding Infeasibility: Makes large-scale problems computationally solvable.

**Real Life examples of Time complexity:**

 Retrieving a customer's account details by their ID (O(1)).

 Searching for a product in a sorted e-commerce database (O(log n)).

 Summing up the sales figures for all transactions (O(n)).

 Sorting millions of customer records by last name (O(n log n)).

 Comparing every pair of users in a social network for mutual connections (O(n²)).

 Calculating the shortest path between all nodes in a transportation network (O(n³)).

 Evaluating all possible combinations of product bundles for optimal pricing (O(2ⁿ)).

 Determining the best delivery route for a set of locations (O(n!)).