MIVA OPEN UNIVERSITY

Algorithms and Complexity Analysis

Full Course Summary

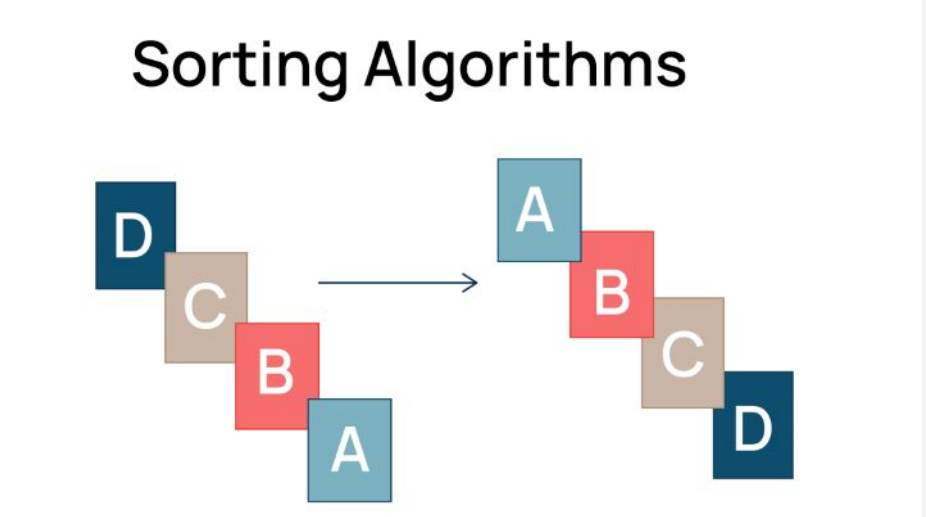
School of Computing

WEEKS 1 - 4

Personal Reflection

- How do you think understanding algorithm efficiency can impact your future programming decisions?
- How does the time complexity of linear search compare to that of binary search in large datasets?
- How do Bubble Sort and Selection Sort compare in terms of memory usage and algorithm complexity for small datasets?
- How does Big O notation help in understanding the scalability of algorithms, and why is it important to express both time and space complexity in this way?

CALL TO ACTION



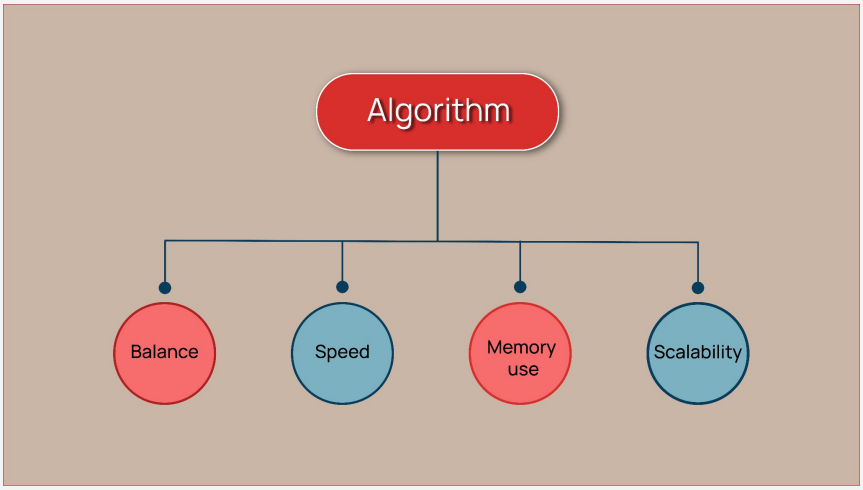
Apply step counting to measure the theoretical complexity of sorting algorithms like Bubble Sort and Merge Sort, then use empirical measurement to validate these results on different input sizes and hardware setups.

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 1 - 4



Questions to ponder on

1. How would you apply the concept of time and space complexity in optimising algorithms for real-time systems?
2. How do you determine the load factor of a hash table, and why is it important for performance?
3. How would you optimise Bubble Sort for slightly better performance, especially for nearly sorted lists?
4. In which situations would you prioritise empirical measurement over step counting when analysing an algorithm?

Skills and Competencies you have acquired after this lesson:

- Algorithm efficiency analysis
- Collision handling techniques
- Designing efficient hash functions
- Problem-solving skills
- Data retrieval
- Theoretical analysis of algorithms

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 1 - 4

Bullet point summary

- Algorithms are step-by-step procedures or formulas used to perform tasks or solve problems in computing.
- In daily life, algorithms are ubiquitous, from following a recipe to navigating a city; in computing, they drive essential tasks such as search engines and email sorting.
- A well-designed algorithm not only solves a problem but does so efficiently, optimising both time and space complexities.
- Search algorithms are vital techniques in computing for efficiently finding specific elements in data structures like arrays, linked lists, or hash tables.
- Linear Search is suitable for small or unsorted datasets, while Binary Search is ideal for large sorted datasets due to its logarithmic efficiency.
- Sorting algorithms like Bubble Sort and Selection Sort play a crucial role in arranging data in a specific order, either ascending or descending, to enable efficient data processing and search operations.
- Both Bubble Sort and Selection Sort are in-place algorithms, meaning they do not require additional memory, but they are inefficient for large datasets.
- Algorithm complexity analysis is essential to evaluate how well an algorithm performs, particularly as the size of the problem increases. It helps in assessing both time complexity and space complexity of algorithms.
- Best-case, worst-case, and average-case complexities provide insight into how an algorithm performs under different conditions. For example, Binary Search has a best-case time complexity of O(1), whereas Linear Search has a worst-case time complexity of O(n).

KEY REFERENCES:

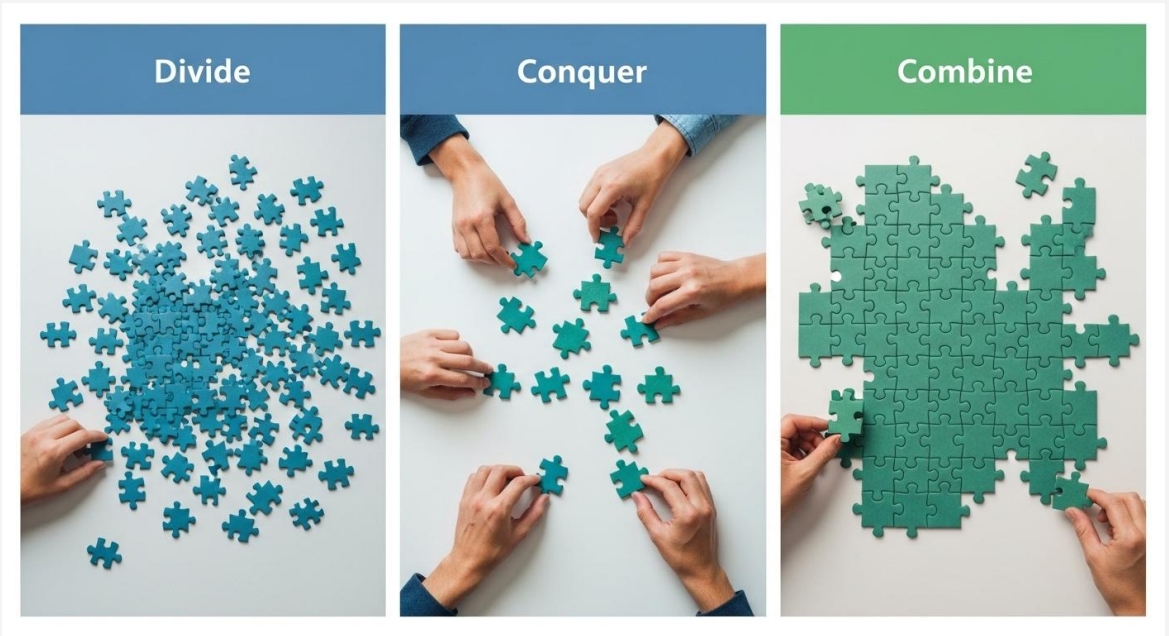
University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 5-8

In Week 5, the course focused on algorithm complexity analysis, which evaluates an algorithm's performance through time and space complexity. Time complexity measures how long an algorithm takes, with notations like O(1), O(n), and $O(n^2)$ describing different growth rates. Space complexity measures memory usage, accounting for both auxiliary and input space. Asymptotic notations like Big O, Big Ω, and Big Θ describe algorithm performance in worst, best, and average cases. Trade-offs between time and space complexity were discussed, highlighting how algorithms like Quick Sort and Merge Sort balance speed and memory usage.

In Week 6, the course explored divide-and-conquer algorithms like Merge Sort and QuickSort. Merge Sort divides the array into two halves, recursively sorts them, and merges them with a time complexity of O(n log n) and space complexity of O(n). It is stable and predictable, making it ideal for large datasets where stability is important. QuickSort, which also follows the divide-and-conquer approach, is faster on average but has a worst-case time complexity of $O(n^2)$, particularly when the pivot selection is poor. It is more space-efficient, with a space complexity of O(log n). The choice between Merge Sort and QuickSort depends on the specific needs, such as stability or memory efficiency.



KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 5-8

In Week 7, the course explored greedy algorithms, which solve optimisation problems by making locally optimal choices at each step. While effective for problems like the fractional knapsack, they don't guarantee optimal solutions for all cases, as shown in the 0/1 knapsack problem. Dynamic programming (DP), which handles overlapping subproblems, offers a more efficient solution for such issues. DP uses techniques like memoisation and tabulation to optimise performance, reducing time complexity. It applies to real-world scenarios like routing and machine learning. Greedy algorithms are best for problems with a clear global optimum, while DP is suited for problems with optimal substructure and overlapping subproblems.

In Week 8, the course discussed approximation algorithms, which are used for NP-hard problems where exact solutions are computationally expensive. These algorithms offer near-optimal solutions in polynomial time, making them practical for large datasets, unlike exact algorithms that may require exponential time. Heuristic algorithms are fast but don't guarantee optimal results, while approximation algorithms provide a performance guarantee, expressed by the approximation ratio. Randomised algorithms, including Las Vegas and Monte Carlo types, use randomness to improve efficiency, with applications in QuickSort and complex problems like numerical integration. Monte Carlo methods are particularly useful in fields like finance and machine learning for optimisation tasks.

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 5-8

Bullet point summary

- Time complexity measures the amount of time an algorithm takes to run as a function of the input size. Common time complexities include O(1) (constant time), O(n) (linear time), O(n²) (quadratic time), O(log n) (logarithmic time), and O(n log n) (linearithmic time).
- Space complexity measures the amount of memory an algorithm uses relative to the input size. It accounts for both auxiliary space (extra space used by the algorithm) and input space (memory used to store the input data).
- Merge Sort is stable, meaning it preserves the relative order of equal elements, whereas QuickSort is not stable by default.
- Las Vegas algorithms always produce correct results, but their runtime varies.
- Some hybrid sorting algorithms, like Introsort, combine QuickSort and HeapSort to optimise performance under different conditions.
- In the fractional knapsack problem, where fractional items can be taken, the greedy approach works optimally by selecting items with the highest value-to-weight ratio.
- In contrast, for the 0/1 knapsack problem, where only whole items can be selected, the greedy approach fails, and methods like dynamic programming are more effective.
- Monte Carlo algorithms run in a fixed time but may produce incorrect results with a certain probability.

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 5-8

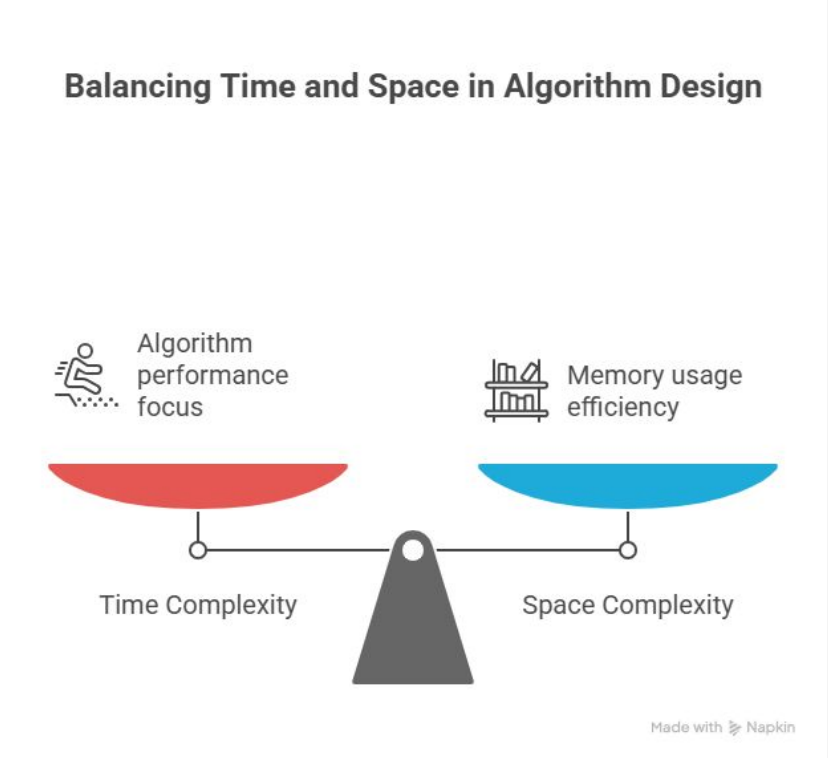
Case studies with images

Case Study 1

Real-world Examples of Algorithmic Complexity

This case study examines the importance of algorithmic complexity in real-world applications, focusing on sorting, pathfinding, and searching algorithms across three fictitious companies. QuickMart, an e-commerce platform, optimised product searches using quicksort and insertion sort, ensuring fast performance during high traffic.

FastTrack Logistics improved its navigation system using A* and Dijkstra's algorithms to provide real-time route updates. HealthSync, a healthcare provider, employed binary search trees, hash tables, and full-text search algorithms to enhance patient record retrieval. These examples highlight the role of efficient algorithms in improving system performance, scalability, and user satisfaction across diverse industries.



KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

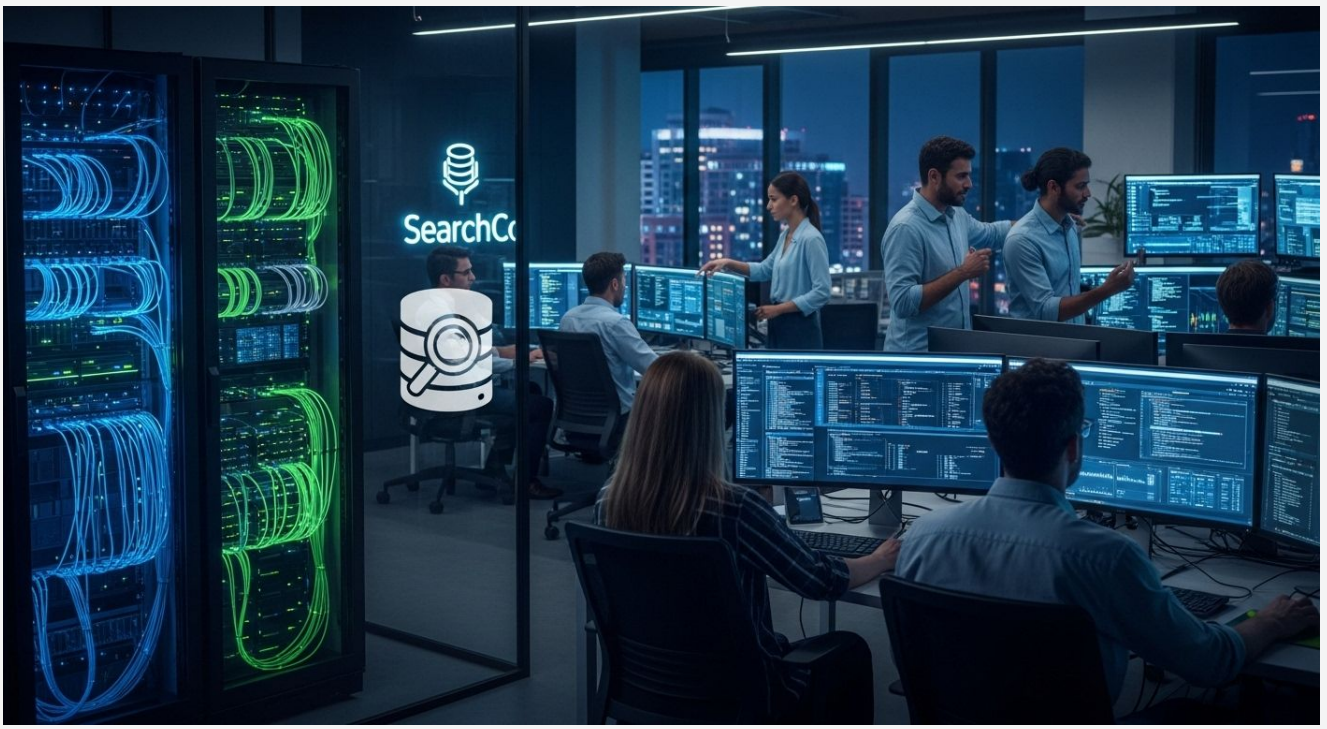
WEEKS 5-8

Case studies with images

Case Study 2

APPLICATIONS OF DIVIDE AND CONQUER IN ALGORITHMIC PROBLEMS

This case study highlights the effectiveness of the divide and conquer technique in solving complex algorithmic problems. TechSort, an e-commerce company, utilised merge sort to efficiently sort large datasets, achieving an optimal time complexity of O(n log n) and enhancing performance, even with large volumes of data. Similarly, SearchCo improved its search engine by implementing binary search, reducing query times from O(n) to O(log n) by dividing sorted datasets. These examples demonstrate how divide and conquer optimises algorithm efficiency, offering scalable solutions for handling large-scale data and complex tasks in real-world applications.



KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 5-8

These case studies demonstrate the importance of choosing the right algorithm to optimise performance and scalability in different fields.

Questions to ponder on

1. How does Big O notation help to estimate the worst-case performance of an algorithm?
2. How does the divide-and-conquer technique make both Merge Sort and QuickSort more efficient compared to simpler sorting algorithms like Bubble Sort?
3. How does dynamic programming handle overlapping subproblems, and why does this make it more efficient than plain recursion?
4. How do approximation algorithms balance accuracy and computational efficiency, and why is this trade-off essential for solving NP-hard problems?

Skills and competencies you should have acquired after this lesson:

- Time complexity analysis
- Space complexity analysis
- Practical application of hybrid sorting algorithms
- Problem-solving skills
- Practical implementation of randomised algorithms

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 5-8

Personal Reflection

Big O, Big Ω, and Big Θ notations are essential for understanding an algorithm's scalability, helping to assess its performance under different conditions. Big O gives insight into the worst-case scenario, Big Ω into the best-case, and Big Θ provides an overall performance view. Considering all three allows for a well-rounded understanding, ensuring the right algorithm choice for varying input sizes and performance needs.

Optimising for time at the expense of space can lead to high memory usage, which is a concern in memory-limited environments like mobile devices. For real-time systems or large datasets, balancing time and space complexities is crucial. How do you think you would approach this trade-off in your own projects, considering the constraints you might face?

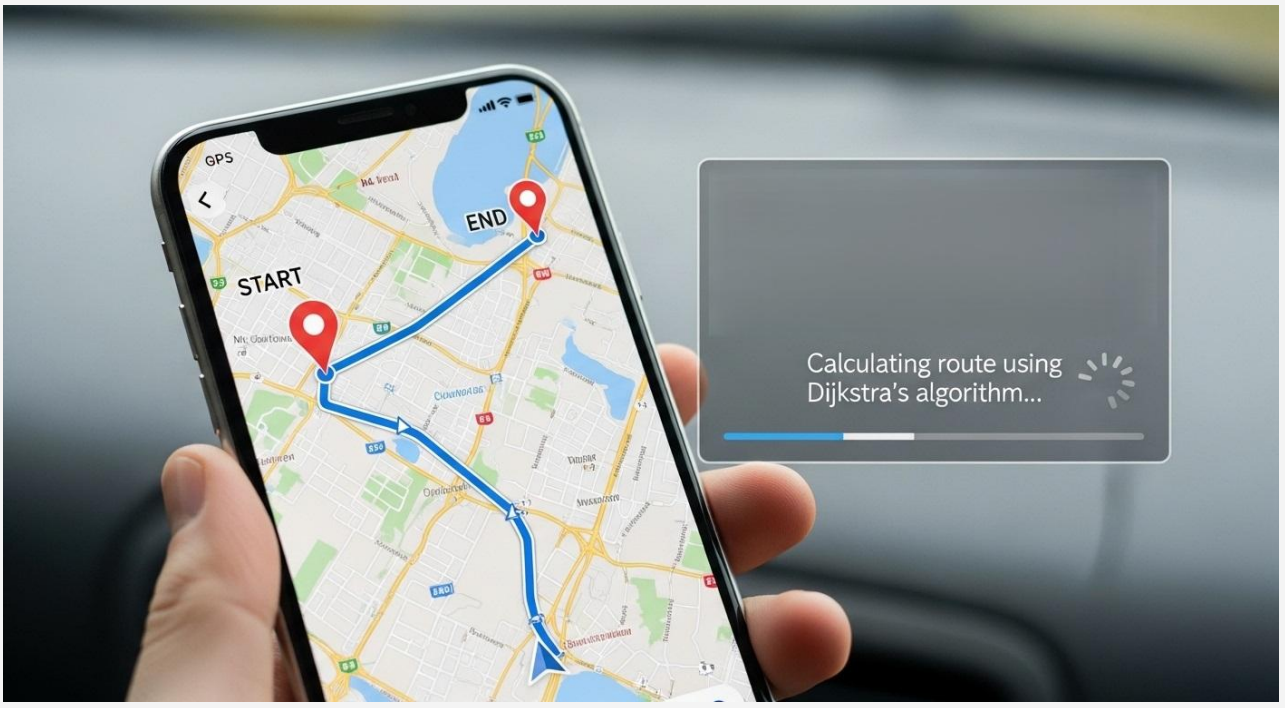
When analysing space complexity, both auxiliary and input space must be considered. Ignoring either can lead to underestimating memory requirements, which could affect performance in data-heavy or real-time applications. How do you prioritise memory management in your own algorithmic approaches?

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 9-12



In Week 9, the course covered NP-hard and NP-complete problems, which are difficult to solve. NP-complete problems can be verified in polynomial time but are as hard as any NP problem. Reduction is used to prove NP-completeness by transforming one NP-complete problem into another. Approximation algorithms provide near-optimal solutions, sacrificing accuracy for efficiency. They are useful in real-world applications like logistics and scheduling. The approximation ratio measures how close a solution is to the optimal. Greedy algorithms often balance solution accuracy and computational efficiency.

In Week 10, the course focused on shortest path algorithms, such as Dijkstra's and Bellman-Ford. Dijkstra's algorithm is efficient for graphs with non-negative weights, selecting the node with the smallest tentative distance and updating neighboring nodes. It has a time complexity of O(E log V) and is widely used in GPS navigation and network routing. Bellman-Ford, on the other hand, handles negative edge weights and detects negative weight cycles, though it is slower with a time complexity of O(V * E). The choice between these algorithms depends on the graph's characteristics and specific problem needs.

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 9-12

In Week 11, the course focused on optimising algorithm performance in operating systems. Time and space complexities evaluate an algorithm's duration and memory usage. Scheduling algorithms like Round Robin and FCFS manage CPU tasks, while memory management algorithms optimise memory. Database algorithms improve data operations, with techniques like indexing and sorting ensuring efficiency. Join and query optimisation algorithms enhance execution time, while concurrency control and transaction management maintain data consistency and resolve deadlocks.

In Week 12, the course covered parallel and distributed algorithms. Parallel algorithms split tasks for simultaneous execution, speeding up processes in fields like machine learning. Distributed algorithms work across multiple machines, tackling challenges like leader election and mutual exclusion. These algorithms are essential in areas such as blockchain and cloud computing, with the CAP theorem outlining trade-offs between consistency, availability, and partition tolerance in distributed systems.

Bullet point summary

- NP-hard problems are computationally as difficult as the hardest problems in NP (Nondeterministic Polynomial time). Solving an NP-hard problem in polynomial time would allow all NP problems to be solved in polynomial time.
- Approximation algorithms are particularly useful in real-world applications like network design, logistics, and scheduling, where exact solutions are computationally expensive or infeasible.
- Shortest path algorithms are used to find the most efficient route between two points in a connected system, such as a road network, flight map, or digital network.
- Bellman-Ford is useful in applications like financial models or economic systems where negative edge weights exist or when detecting negative weight cycles is crucial.
- Database algorithms are key to efficient data operations, from searching and sorting to query optimisation and concurrency control.

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 9-12

Bullet point summary

- Parallel algorithms aim to solve complex problems more efficiently by dividing tasks into smaller sub-tasks and executing them simultaneously across multiple processors or cores.
- Distributed algorithms run on multiple independent machines (nodes) that communicate via message passing to achieve a common goal.
- Optimisation techniques like load balancing and priority scheduling are employed to improve system responsiveness and efficiency.
- Sorting algorithms like external merge sort are used for large datasets that don't fit into memory, ensuring efficient sorting even on disk.

Case studies with images

Case Study 3

REAL-WORLD ALGORITHMIC PROBLEM-SOLVING APPROACHES

This case study explores various algorithmic problem-solving approaches, including greedy algorithms, dynamic programming, and backtracking, applied to real-world scenarios. QuickMove Logistics utilised a greedy algorithm for route optimisation, improving delivery times during peak hours by selecting the nearest city at each step. Genomics BioTech applied dynamic programming to efficiently compute the longest common subsequence (LCS) between DNA sequences, significantly improving processing times for large datasets. ChessAI Solutions used backtracking to solve the N-Queens problem, efficiently exploring possible solutions and pruning invalid ones.

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 9-12

Questions

1. How does greedy approximation compare with other approximation methods in terms of time complexity and accuracy?
2. How would you modify Dijkstra's algorithm to handle negative edge weights if necessary?
3. How can compression algorithms reduce storage and I/O costs in large-scale databases, and what are some real-world applications of these techniques?
4. How does parallelism in sorting algorithms reduce execution time, and what limitations does it face when applied to real-world problems?

Skills and competencies you should have acquired after this lesson:

- Time and space complexity analysis
- Implementation of graph algorithms
- Understanding algorithm limitations
- Understanding parallel algorithms
- Designing distributed systems
- Practical implementation of blockchain technologies

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.

WEEKS 9-12

Personal Reflection

- The approximation ratio helps assess how close an algorithm's solution is to the optimal. A ratio greater than 1 means the solution is suboptimal but still acceptable. Approximation algorithms are useful when exact solutions are too costly. How would you balance accuracy and efficiency when choosing an algorithm for your projects?
- Reduction demonstrates NP-completeness by transforming a known problem into another. In real-world scenarios, approximation solutions are often good enough for NP-hard problems. Bellman-Ford can handle negative weights, unlike Dijkstra's, making it more versatile. When might you prefer Bellman-Ford over Dijkstra's in your applications?
- Time and space complexities impact scheduling and system performance. External merge sort is better for large datasets, and hash joins are more efficient than nested loops. Query optimisation and transaction management ensure better performance and consistency. How do you prioritise time and space complexity in your algorithmic decisions?

KEY REFERENCES:

University of Lagos. (2013). Nigerian Peoples and Cultures. Lagos: Centre for General Studies.

Odinaka, C. (2020). Nation-Building and National Integration in Nigeria: A Historical Study.