



OCR A Level Computer Science



Your notes

7.2 Computational Methods

Contents

- * Computational Methods
- * Problem Recognition
- * Problem Decomposition
- * Divide & Conquer Algorithms
- * Use of Abstraction
- * Backtracking Algorithms
- * Heuristics for Problem Solving
- * Performance Modelling
- * Pipelining as a Computational Method
- * Visualisation for Problem Solving



Your notes

Computational Methods

Features of Computation

- It's important to consider if problems can be solved using algorithms and programming code
- For example:
 - Sorting a list of numbers is feasible using algorithms like [quick sort](#) or merge sort
 - But ethical problems or social problems, like determining if a loan should be approved, may incorporate some human input that algorithms could oversimplify or misinterpret
- Problems that can be solved with algorithms and programming code are **computational problems**

Real-world constraints on computable problems

- Practical limitations such as computing power, speed, and memory can affect whether a problem is solvable
- Running complex machine learning models on a regular laptop might be constrained by limited processing power and memory

Challenges with resource-intensive problems

- Some problems are theoretically solvable but not practical due to resource limitations
- For example, calculating Pi to a billion decimal places is theoretically possible but impractical due to the amount of computational resources needed

Advances in technology

- Technological improvements have expanded the types of problems that can be computationally solved
- For example, **genome sequencing** has become quicker and more affordable due to advances in technology

Example

- Below is a table of **problems** identified for an online grocery business
- The table shows which problems are computational and a reason why



Your notes

Problem Description	Computational Problem (Yes or No)	Justification
Inventory levels are not updated in real-time	Yes	Real-time syncing can be achieved through algorithms
High rate of employee turnover	No	Root causes are likely cultural or managerial, not algorithmic
Incorrectly sorted products in the delivery van	Yes	Sorting algorithms can optimise placement for efficiency
Long wait times for customer service	Yes	Queue algorithms can improve response times
Poor route optimisation for delivery trucks	Yes	Routing algorithms exist for this specific problem
Inadequate marketing strategies	No	Marketing strategies often require creative and human-centric solutions



Worked Example

The table below outlines various challenges in a public transportation system.

Evaluate each problem and decide whether it is computational and justify each of your decisions.

Problem Description	Computational (Yes or No)	Justification
Frequent train delays		
Overcrowding in peak hours		
Difficulty in finding the shortest route		
Vandalism in the stations		



Your notes

Inconsistent fare pricing		
Inaccurate timetable		

How to Answer This Question

- 1. Review the Table:** Take a moment to read through the problems listed and think about whether a computational solution would be possible
- 2. Fill in the Table:** In the "computational" column, specify whether the problem should or should not be solved computationally
- 3. Justify your answers:** In the last column, justify your decision for each problem briefly

Answer:

Problem Description	Computational (Yes or No)	Justification
Frequent train delays	Yes	Real-time tracking and predictive algorithms can help in better scheduling
Overcrowding in peak hours	Yes	Load-balancing algorithms can redistribute passengers or add more trains during peak hours
Difficulty in finding the shortest route	Yes	Route-finding algorithms can quickly identify the most efficient path
Vandalism in the stations	No	Requires human intervention such as increased security personnel
Inconsistent fare pricing	Yes	Dynamic pricing algorithms can ensure fare consistency
Inaccurate timetable	Yes	Timetabling algorithms can generate more accurate and optimised timetables



Your notes

Problem Recognition

Problem Recognition

What is Problem Recognition?

- Problem recognition is determining if there is a problem that needs to be solved
- For example, if customers of an online shop complain that a checkout process is slow then problem recognition can be used to determine that there is a problem to be solved and what the problem actually is
- In this example, the core problems might relate to how many steps exist in the process or the poor organisation of information on the page
- Both of these problems are possible to solve through a software solution
- Code can be rewritten so that the number of steps is reduced and the information can be arranged in a simpler way
- But not all problems should or can be solved with software

Example 1: Slow response time in a GP surgery booking system



A GP Surgery recently implemented an online appointment booking system to make booking an appointment with a doctor easier. While the new system was expected to make the process more efficient, patients have reported slow response times when attempting to confirm their online appointments. Some have even mentioned waiting several minutes for the confirmation screen to appear, leading to frustration and abandoned bookings. The system's sluggishness is particularly noticeable during peak hours between 8am – 9am when multiple users try booking appointments simultaneously.

The big problem

- Slow response time: The primary issue is the slow response time in the online appointment booking system, leading to user frustration and abandoned bookings

Sub problems



Your notes

1. **Peak hour bottlenecks:** The system experiences a significant peak-hour **slowdown**, indicating potential **scalability** issues
2. **Delayed confirmation:** Patients must wait several minutes for a confirmation screen to appear, which could point to inefficient processing of booking requests
3. **User experience:** Frustrated users may lose faith in the system or the GP surgery, possibly affecting its reputation
4. **Abandoned bookings:** The slow system is causing patients to abandon their appointments, meaning they are not seeking help with their health

Example 2: Low employee morale and high turnover rates in a sales team



A medium-sized company specialising in consumer electronics is experiencing high turnover rates in its sales team. Employee morale is notably low, and several exit interviews reveal that team members feel undervalued and overstressed. Sales targets are frequently unmet, and the remaining team members feel demoralised by the constant departures. Management is concerned that this ongoing issue may start affecting the overall performance and reputation of the company.

The big problem

- **Low employee morale and high turnover:** The core issue is the high employee turnover rate and declining morale, which affects both team performance and the company's reputation

Sub problems

1. **Unrealistic sales targets:** Employees feel that the sales goals set by management are too challenging to achieve, causing stress
 2. **Lack of support:** There is a feeling among team members that they lack the necessary support and resources to perform well
 3. **Poor work-life balance:** Long working hours and high stress levels contribute to an imbalanced work-life equation
 4. **Team cohesion:** The frequent departures disrupt team cohesion, affecting collaborative efforts and reducing overall productivity
- In this case, problems like low morale, unrealistic targets, and lack of emotional support are largely cultural and psychological issues and can't be directly solved with software

- Some software solutions may exist to help, such as project management, or somewhere to record and analyse why employees are leaving
- But the big problem requires human-centric solutions like revised management practices, team-building exercises, and counselling services



Your notes



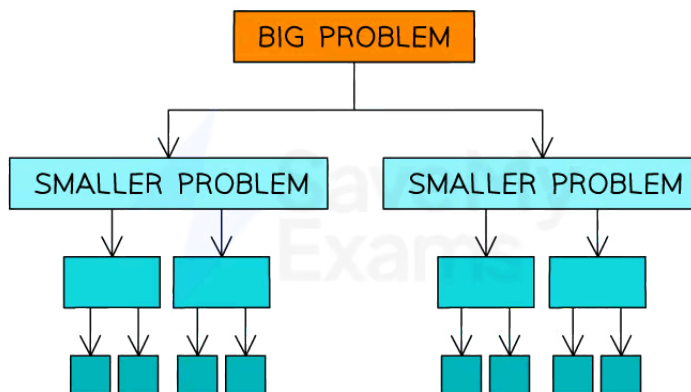
Your notes

Problem Decomposition

Problem Decomposition

What is Decomposition?

- Decomposition is **breaking down** a big problem into smaller problems so that they can be solved independently
- Programmers use decomposition to:
 - Break problems down
 - Identify the steps, parts or processes involved in a problem
 - Identify reusable components
 - Split tasks between programmers
- In the case of slow appointment booking, you could decompose the problem into issues such as server limitations during peak hours, inefficient backend algorithms, and user interface delays



Copyright © Save My Exams. All Rights Reserved

Process of decomposition to break problems down

Using abstraction to design a solution

- Before we start to break problems down, it is often useful to apply the rules of abstraction to the problem



Your notes

- Applying abstraction will remove non-essential elements and that programmers can focus on critical aspects for problem-solving
- When addressing the slow booking system, a programmer could ignore elements like the system's colour scheme or graphics and focus on critical performance metrics such as server response time and database query efficiency

Decomposing slow response times problem

1. **Big problem:** Slow response times during the online appointment booking process
2. **Decompose** into sub-problems:
 - Server limitations causing bottlenecks during peak hours
 - Inefficient algorithms leading to delayed confirmation screens
 - Poor user interface contributing to an overall bad user experience
3. **Prioritise** sub-problems:
 1. Address server limitations, as solving this could have a broad impact on system performance
 2. Look into optimising algorithms to speed up the booking and confirmation process
 3. Lastly, make user interface improvements to enhance user experience, possibly mitigating some of the frustration caused by slow response times



Worked Example

A popular online retail platform has recently faced significant problems with its recommendation system. Customers complain that the recommendations are not relevant, and this is affecting sales figures.

Analyse the problems facing the recommendation system of the online retail platform.
Decompose the problem into smaller components and discuss possible software solutions.
What are the limitations of relying solely on software to improve the recommendation system?

How to answer this question:

1. **Introduction:** Introduce the scenario and pinpoint the main problem: the ineffective recommendation system affecting sales
2. **Decomposition:** Break down the issue into sub-problems such as data quality, algorithmic inefficiency, and lack of user input
3. **Software Solutions:** Propose software-related solutions like machine learning algorithms, improved data collection, and user interface changes for better user feedback



Your notes

4. **Limitations:** Discuss why software solutions alone might not be sufficient to solve the problem completely

Answer:

The online retail platform faces a significant issue with its recommendation system, impacting customer satisfaction and sales. The problem can be decomposed into several smaller components: poor data quality, inefficient recommendation algorithms, and a lack of an interface for users to give feedback on recommendations. Software solutions can play a vital role in addressing these issues.

Machine learning algorithms can be developed to make more precise recommendations. Data quality can be improved by incorporating a more diverse set of user activities and perhaps by using third-party data.

User interface improvements could allow customers to provide immediate feedback on recommendations, which could further refine the algorithm. However, even the most advanced software solutions have limitations. For instance, without human input, they can't capture real-world influences on customer behaviour or adapt to rapid market changes.

Acceptable answer you could have given instead:

The online retailer is struggling with an ineffective recommendation system that needs improvement. This problem can be broken down into issues like inadequate data quality, outdated algorithms, and missing user feedback mechanisms. Software solutions are essential for solving these problems.

Upgrading the algorithms and improving data quality can make the recommendations more relevant. Moreover, incorporating a user feedback feature can also help refine the system. However, software alone might not be a complete solution as it can't account for every variable affecting customer behaviour or rapid changes in consumer trends.



Your notes

Divide & Conquer Algorithms

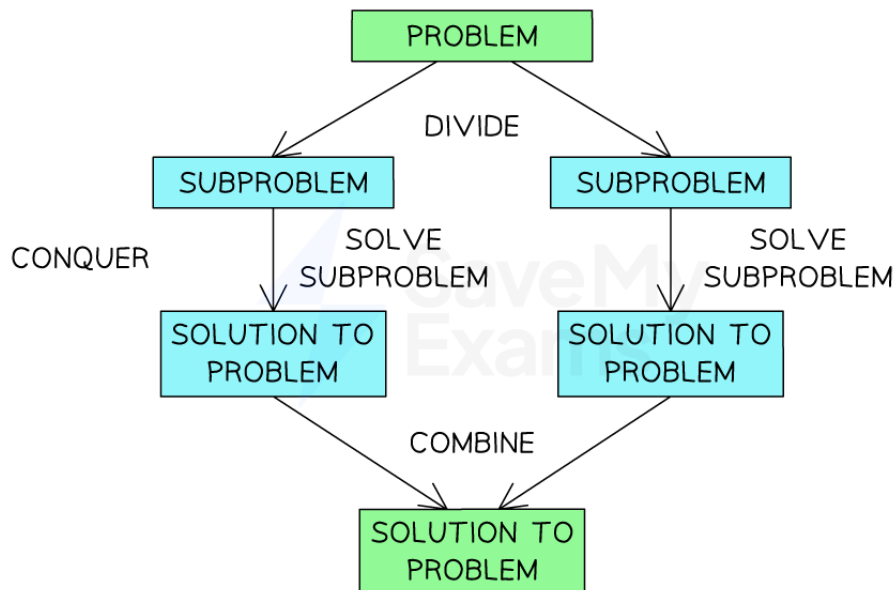
Divide & Conquer Algorithms

What is the Divide & Conquer Strategy?

- Divide and conquer is a strategy to make a complex task easier by breaking it into smaller, more manageable tasks
- For example, organising a large conference can be split into smaller tasks such as booking the venue, coordinating with speakers and marketing

Implementing divide and conquer

- Divide and conquer contains three steps:
 - **Divide** - The problem needs to be broken down into sub-problems
 - **Conquer** - The sub-problems then need be solved independently
 - **Combine** - The solutions to the sub-problems can then be combined to form the overall solution to the problem
- For example, in home renovation, the project can be broken down into individual rooms or specific elements such as flooring, painting and electrics



Copyright © Save My Exams. All Rights Reserved

Process of divide and conquer to solve problems

Your notes

Benefits	Drawbacks
Divide and conquer can make programs more time efficient.	Not all problems can be broken down and solved independently.
As problems are divided into sub-problems they can make effective use of cache memory.	It can possibly cause stack overflows if recursion is being used.

Task parallelism

- This is when several tasks or sub-tasks can be carried out concurrently (at the same time) to speed up the overall completion time
- For example, on a factory assembly line, different components of a product can be put together at the same time by different teams or machines



Worked Example

You are the lead software engineer for a company building an e-commerce platform. Your team has been tasked with improving the website's search functionality, which has recently been sluggish due to an exponential growth in the number of products available.

Describe how you would use the divide and conquer strategy and task parallelism to tackle this problem. Make sure you provide specific examples for each.

How to answer this question:

1. Introduction: Briefly explain the problem of sluggish search functionality in the e-commerce platform
2. Divide and conquer: Describe how you would break the problem into smaller, more manageable tasks
 - For example, consider breaking down the search process into tasks like query parsing, data retrieval, and front-end rendering
3. Task parallelism: Explain how tasks or sub-tasks can be executed simultaneously to speed up the problem-solving
 - For instance, data retrieval can be parallelised by dividing the product database into smaller chunks that can be searched simultaneously

Answer:



Your notes

The issue at hand is the slow search functionality on the e-commerce platform. This is a critical problem as it hampers the user experience and could decrease sales.

The problem can be divided into smaller, more manageable parts to approach this issue efficiently. Specifically, we can isolate query parsing, which involves translating user text input into a query. We could work on data retrieval, where the relevant product information is fetched from the database as well as front-end rendering which is responsible for displaying the products to the users.

Once the problem is segmented, we can apply task parallelism to expedite the resolution. During the data retrieval process, we can partition the product database into smaller index-based segments, allowing multiple server instances to search them simultaneously. Concurrently, front-end rendering can implement lazy loading techniques to begin displaying products as they are retrieved, thus improving the speed of the search function.

We can notably improve the search functionality and user experience by employing both divide and conquer and task parallelism.

Acceptable Answer You Could Have Given Instead

Our website's slow search function needs to be fixed. The problem could be broken down into a few parts:

- improving the search algorithm
- making our database faster
- optimising the user interface

We can work on multiple steps simultaneously to solve this problem. For instance, one team can focus on the database while another works on the user interface. This approach would help make the search function faster and satisfy our users.

Both answers effectively address the problem but vary in the depth and clarity with which they explain the concepts of divide and conquer and task parallelism.



Your notes

Use of Abstraction

Use of Abstraction

- Before revising the **Use of Abstraction** it is important to have a deep understanding of the concept of **Thinking Abstractly**

What is Abstraction?

- Abstraction is the **removal of unnecessary components** of a problem to allow focus on only those that are necessary
- Without abstraction, many real world applications would have far too many variables to take into consideration

Use of Abstraction – Computer Games



- Computer games use a large amount of abstraction, removing the elements that a user does not need to consider in order to enjoy playing the game



Your notes

- When using abstraction in computer games which are designed to simulate real life, the aim is to make the game realistic and visually appealing whilst keeping the game fun to play
- In a game that simulates a sport, it is important to the user that visually they recognise the environment and when they perform an action, they see a response
- However, users do not need to know the complex algorithms used to control the non player characters (NPCs)

Use of Abstraction – Cooking with a Recipe



- When cooking with a recipe, the concept of abstraction has already been applied
- The purpose of a recipe is for the user to follow it and end up with the desired result
- For example, in a recipe that asks a user to 'brown' an ingredient, they do not need to understand the chemistry behind the process, only the desired output in order to move on to the next step
- Abstraction gives a user the ability to focus on the recipes intent, rather than the science

Use of Abstraction – Driving a Car



Your notes



- When driving a car, the driver uses a key or button to start the engine, and uses pedals to start and stop
- The driver does not need to know the intricacies of how the engine works to complete the task
- Abstraction empowers people to utilise complex machines like cars



Your notes

Backtracking Algorithms

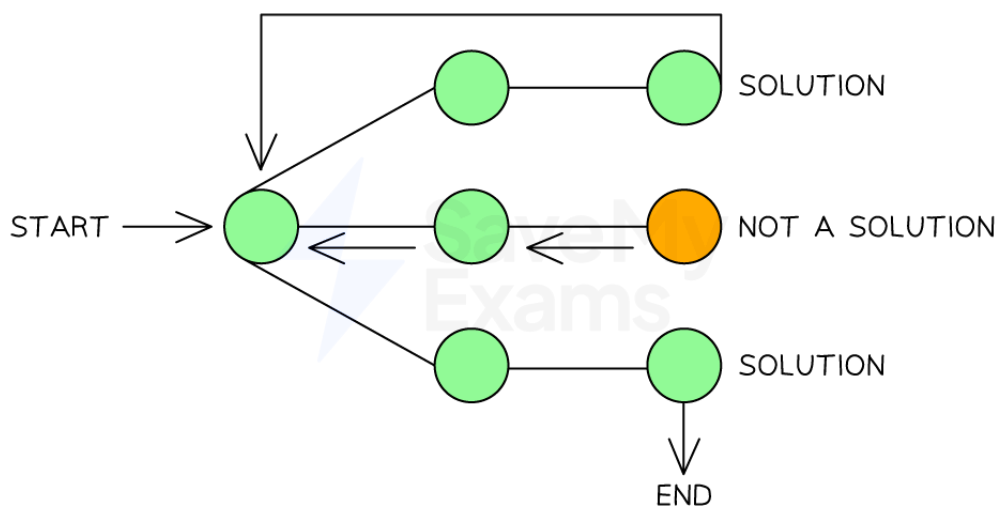
Backtracking Algorithms

What is Backtracking?

- Backtracking is a technique that is used when you don't have enough information to find a solution to a problem or if you have many possible ways of solving a problem
- It will therefore explore all possible paths in the search space to determine if these come to dead ends
- It will build up partial solutions to a large problem incrementally and then if the solution fails at some point, the partial solutions are abandoned and the search begins again at the last potentially successful point
- Backtracking is ideal if you are trying to solve logic problems, however, is not ideal for solving strategic problems
- For example, backtracking can be used to navigate around a maze to move forward until a wall is hit (dead end), then backtrack to try another route

Example use

- Backtracking can be particularly useful when traversing data structures such as trees or graphs as these have multiple paths that can lead to the desired solution
- In a binary search tree, backtracking helps to return from leaf nodes (dead ends) so that other subtrees can be explored



Copyright © Save My Exams. All Rights Reserved

Using backtracking to solve problems

Your notes

Benefits	Drawbacks
It is guaranteed to find a solution if one exists.	Is not ideal for solving strategic problems.
It is easy to implement and use.	Depending on the problem that you are trying to solve, it can have a high time complexity.
It can be applied to a variety of logic problems.	If there are lots of different solutions to a problem, it is not always the most efficient method.
It will comprehensively explore all possible paths to the desired solution.	It can consume a lot of memory.
	Although it may find a solution to the problem, the solution may not always be the best solution available.
	It will often require a complete search of the entire solution space.

**Worked Example**

You are developing a maze-solving application where a user-controlled character needs to navigate from the start point to the end point. The maze contains several dead ends and multiple pathways to reach the destination.

Describe how backtracking can be used to solve this problem. You should include the steps involved and any advantages or limitations of using backtracking for this scenario.

How to answer this question:

1. **Introduction:** Explain what is meant by the term 'backtracking' and why it's suitable for solving maze problems
2. **Algorithm Steps:** Give the steps the backtracking algorithm would follow to solve the maze problem
3. **Advantages:** Give the advantages of using backtracking
4. **Limitations:** Give any limitations of using backtracking



Your notes

Answer:

Backtracking is an algorithmic approach that builds a solution incrementally. It's ideal for solving maze problems because it explores each path until it reaches a dead end and then retreats (backtracks) to another pathway to explore. The backtracking algorithm would start at the maze entrance and move step-by-step towards the exit, marking visited areas of the maze. Upon reaching a dead end, the algorithm would backtrack to the last unvisited branch.

The primary advantage of backtracking is that it guarantees a solution if one exists. It's efficient and can be easily implemented. However, backtracking can be slow for complex or large maze problems and it may not always find the best solution available. Backtracking offers a reliable, straightforward way to solve maze problems despite its limitations.

Acceptable answer you could have given instead:

Backtracking is good for solving maze problems as it can find a path from start to finish. The algorithm starts at the entrance and moves through the maze, backtracking when it hits a wall (dead end) until it finds the exit. This method will find a solution if one is available, however, it may take longer for bigger mazes. Backtracking is a suitable method for solving mazes, although it may have some downsides, like speed and poor time complexities.



Your notes

Heuristics for Problem Solving

Heuristics for Problem Solving

What are Heuristics?

- Heuristics is making use of experience to find a solution to a problem quickly
- It uses concepts like 'rules of thumb' and 'educated guesses' to find a solution faster than traditional methods
- It prioritises speed and **not** accuracy
- It aims to find a solution that is 'good enough' rather than perfect

Trade-off between speed and accuracy

A game is called 'Hot and Cold' and the rules are as follows:

- A person (known as the searcher) tries to locate a hidden object by listening to clues from another person, who can only say "hotter" or "colder" based on the seeker's proximity to the hidden object
- "Hotter" or "colder" clues are indicators of where the object is, but they don't give an exact location

Thinking about the use of heuristics in this game:

- If the searcher misinterprets the clues, they may get stuck in a spot that seems "hot" but is not the actual target
- This predicament also happens in heuristic algorithms and is known as getting stuck in a local optimum
- The searcher responds to feedback and gains more intelligence to find the object, usually resulting in them finding the object
- The method finds the object more quickly than random searching, but it doesn't guarantee the quickest or most direct route will be taken
- This is the same for heuristic methods, where there is a trade-off between speed and accuracy

Heuristic methods in software

- The A* algorithm is a common example that uses heuristics in pathfinding and graph traversal
- The aim of the A* algorithm is to use heuristics to find a path from a start node to an end node quickly, however, the path that it finds may not always be the most efficient path possible
- Learn more about [A* Algorithm](#)



Your notes

Benefits	Drawbacks
Heuristics can usually find a solution close to the best solution available.	It will not guarantee that you will find the 'best' solution as it aims to find a solution quickly that is 'good enough.'
Heuristics save time as you may not to investigate every single possibility to get a definite answer.	There needs to be careful consideration to be made between accuracy and time.
Heuristics is very practical and can be easily implemented.	The heuristic values may be incorrect which can lead to inaccurate solutions being found.



Your notes

Performance Modelling

Performance Modelling

What is Performance Modelling?

- Performance modelling is when the behaviour of something is tested or simulated before it is used in the real world
- It is a systematic approach that can be used for evaluating and predicting the performance characteristics of a software system

Using Performance Modelling in Software Production

- Performance modelling can be used to help understand how the software will behave under different load conditions and configurations
- It uses various metrics like response time and throughput to identify potential bottlenecks which can be used by developers to address performance issues before they affect end-users
- It can be integrated into various stages of the software development including the:
 - Design phase** to make architectural decisions
 - Testing phase** to simulate real world scenarios and measure the software's performance

Benefits	Drawbacks
Stress testing can ensure a system can cope with a large set of data or a large number of users.	The outcome of performance modelling is only as useful as the accuracy of the data that is fed into it.
You are able to predict problems and act on them before the problems actually occur in the real world.	If the rules that made up the model are wrong then it will produce incorrect results.

Examples of Performance Modelling

Database Optimisation

- Performance modelling can be used to simulate different database architectures and query strategies to find the most efficient setup
- It can help select the best indexing strategy and estimate the query response times under varying loads

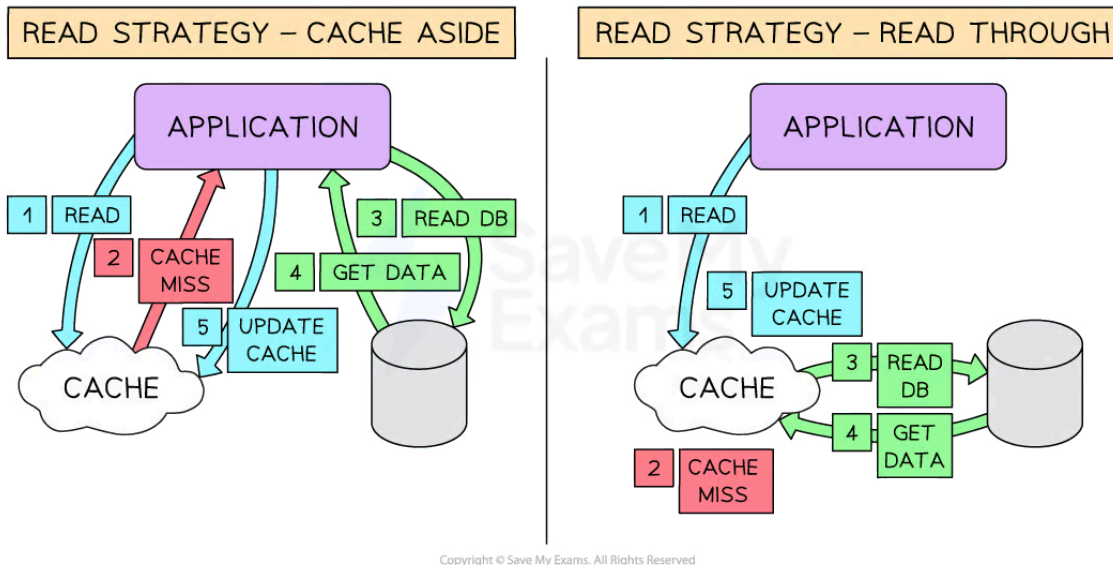
- It may help decide between a relational database design or a single table design

Caching Mechanisms

- By modelling how different caching strategies perform under various scenarios, optimal cache sizes can be determined
- Developers can assess the hit/miss ratio of caches and the latency improvements gained through caching



Your notes



Use of performance modelling to determine a caching mechanism

Energy Efficiency

- For mobile or embedded systems, performance modelling can be used to estimate power consumption under different usage patterns
- It can be useful for battery-powered devices where power consumption is a critical factor



Your notes

Pipelining as a Computational Method

Pipelining as a Computational Method

- Before revising **pipelining as a computational method**, it is important to revise the concept of **pipelining** within the CPU first.

What is Pipelining?

- Pipelining is the process of **carrying out multiple instructions concurrently**
- It improves the overall **efficiency** and **performance** of a task or instruction
- In order to utilise pipelining it can be broken down into the following stages:
 - Break down the task** – analyse the process carefully to identify the individual tasks that make up the overall job
 - Arrange in sequential order** – organise the tasks into a logical sequence, where the output of one task becomes the input for the next
 - Allow **multiple tasks to operate at the same time** – instead of waiting for one task to fully complete before starting the next one, allow multiple tasks to operate concurrently

Pipelining in Programming

- In languages such as Python and Java pipelining is **chaining multiple instructions together**
- Pipelining is a similar concept to an **assembly line** which is used to make the overall process more **efficient**
- Here is an example program written in Python that generates squared values of numbers from a list and then filters them to find only the even values

```
01 list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
02 squared = [x**2 for x in list]
```

```
03 filtered = [x for x in squared if x % 2 == 0]
```

```
04 print(filtered)
```

- Line 01 creates a list of ten numbers
- Line 02 uses **iteration** to square each value in the list **one by one** and generate a new list called 'squared'



Your notes

- Line 3 **filters** the new list, searching for each **even number** and adding them to a separate list called "filtered"
- On line 04 we print the filtered list, the final output would be:
[4, 16, 36, 64, 100]
- In the first example commands are separated by being on their own individual numbered lines, in other languages such as **Unix** the '|' symbol is used to **separate commands**
- Each command in the '**pipeline**' is separated and data flows left to right
- An example of the '|' symbol used in Unix:

grep "hello" docs | grep "world"

- Here two **grep** commands (search) are chained together to perform a specific task
- The first **grep** command **grep "hello" docs** searches for lines that contain the word "hello" in a list of files contained in a file named 'docs'
- The pipe symbol '|' connects the **output of the first command** to the **input of the second command** meaning only lines containing the word "hello" are passed to the second command
- The second **grep** command **grep "world"** searches for lines in the **filtered results** from command 1 that contain the word "world"



Worked Example

When a car is manufactured, the following activities take place:

- Designs drawn
- Engine developed
- Chassis developed
- Electronics planned
- Welding to create the body of the car
- Painting to protect the cars exterior
- Engine assembled
- Interior assembly
- Wheels attached
- Final inspection

Define the term 'pipelining' [2]

- Carrying out instructions concurrently [1]
- The result from one process/procedure feeds into the next [1]

Explain how the principles of pipelining can be applied to ensure a car is manufactured as quickly as possible [4]

- Identify tasks that can be carried out at the same time [1] for example, the engine and chassis can be developed separately but at the same time once the designs have been drawn [1]
- Identify tasks that must be completed in order [1] for example, painting once the body has been created/final inspection once all other tasks have been completed [1]

Describe one example of where pipelining is user in a computer system [2]

- Pipes to pass data between programs [1] for example, the | symbol used in Unix [1]



Your notes



Your notes

Visualisation for Problem Solving

Visualisation

What is Visualisation?

- Visualisation is when data or concepts are presented in simpler form for humans to understand
- It will often create a graphical or visual representation of something to understand complex systems or data

Benefits	Drawbacks
Visualisation can simplify concepts that are easier for humans to understand.	Although it can show data in a visual way, it cannot explain why something is the way that it is.
It can make it easier to spot new trends and patterns that have not been spotted before.	Different people may interpret the visualisations in different ways.
It can be used to explain complex situations to allow programmers to get a better understanding of the problems of a current system.	The way that the data is presented can have an impact on its understanding.

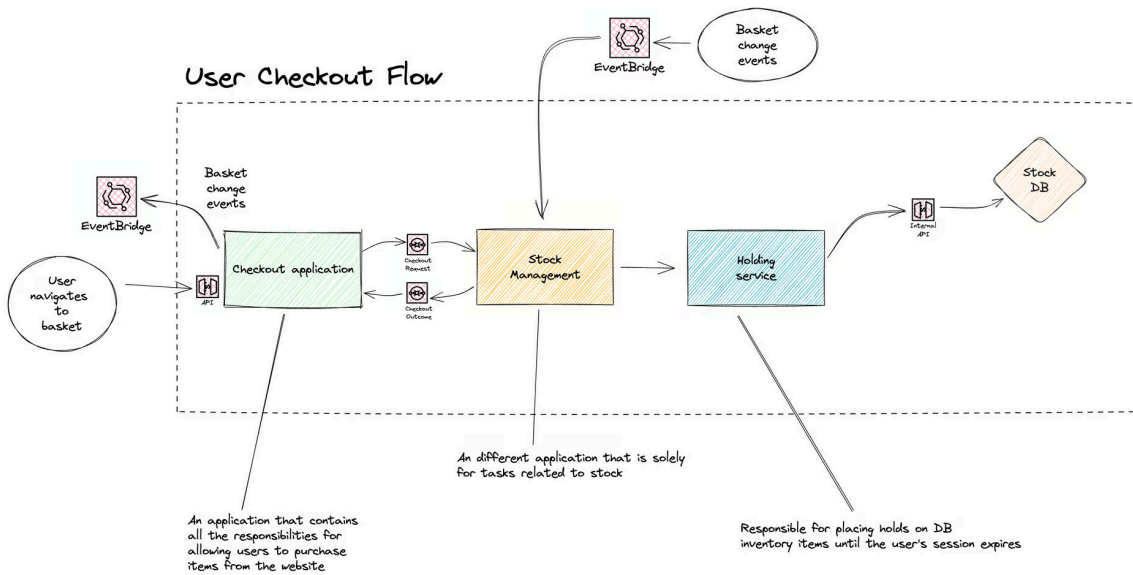
Example Uses of Visualisation

Flowcharts

- These are diagrams that represent workflows or processes in a system
- They help in mapping out each step, making it easier to predict outcomes
- In the example below, the checkout process is a complex user journey that has to ensure users don't check out for items that are no longer available in stock
- When buying concert tickets or limited-edition merchandise, these are usually highly volatile episodes for a checkout server, especially when users often have multiple baskets across multiple devices, abandoning journeys and editing basket contents



Your notes



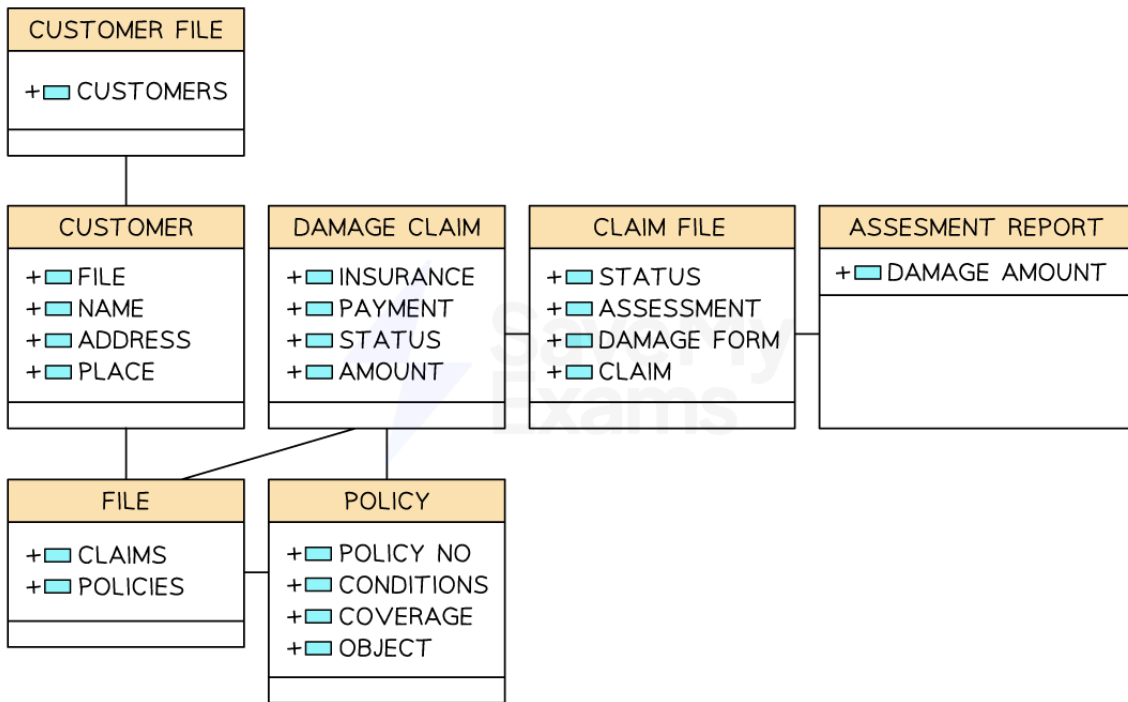
Using flowcharts to visualise a checkout process

Unified Modeling Language (UML) Diagrams

- These provide a standard way to visualise a system's architecture
- They are useful for understanding how classes and objects interact with each other



Your notes

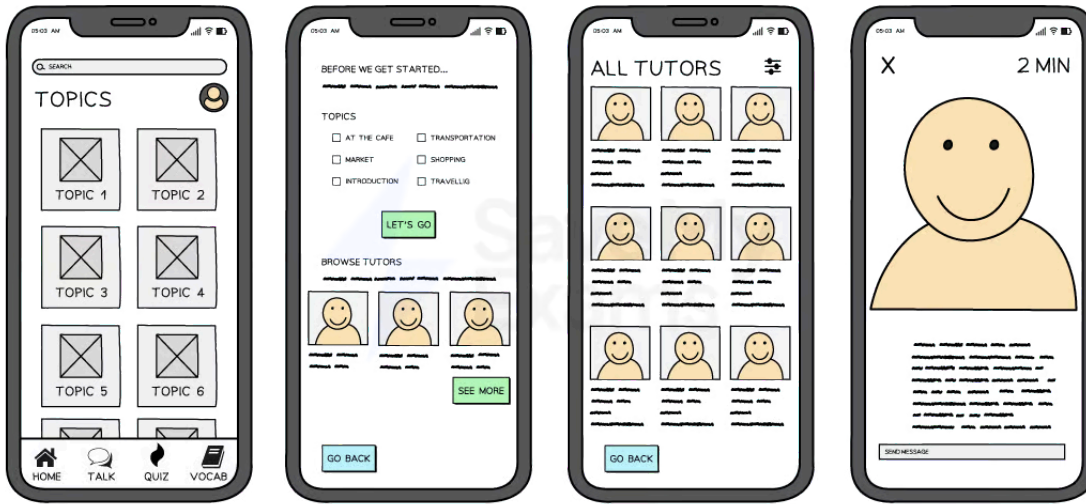


Copyright © Save My Exams. All Rights Reserved


Using UML diagrams to visualise how objects interact with each other in a system

Wireframes

- These are low-fidelity design plans that represent the skeletal framework of a program or website
- They help with layout planning and user experience design



Copyright © Save My Exams. All Rights Reserved


Your notes

Using wireframe diagrams to visualise how a user interface will look