Here’s a detailed list of **viva questions with answers** for your **Selection Sort** code. These are categorized into:

1. ✅ **Expected Questions (based on the code and problem statement)**
2. 🧠 **Basic/Related Conceptual Questions (commonly asked in viva)**

**✅ 1. Expected Questions (Code + Problem Statement)**

**1. What is Selection Sort?**

**Answer:**  
Selection Sort is a comparison-based sorting algorithm. It repeatedly selects the minimum (or maximum) element from the unsorted part of the array and places it at the beginning. This process continues until the array is sorted.

**2. How does your code implement selection sort?**

**Answer:**

* The outer loop iterates over each position in the array.
* The inner loop finds the index of the smallest element in the unsorted part of the array.
* Then it swaps that smallest element with the current position from the outer loop.
* This process repeats until the array is fully sorted.

**3. What is the time complexity of Selection Sort?**

**Answer:**

* **Best case:** O(n²)
* **Average case:** O(n²)
* **Worst case:** O(n²)  
  Selection Sort always compares all pairs regardless of the input's order.

**4. What is the space complexity of Selection Sort?**

**Answer:**  
O(1) — Selection Sort is **in-place** (does not use extra memory).

**5. Does Selection Sort work for negative numbers or duplicate elements?**

**Answer:**  
Yes, Selection Sort works for **any integer values**, including negative numbers and duplicates. It just compares values using <.

**6. What is the role of the swap function in your code?**

**Answer:**  
The swap function exchanges the values at the two indices — it places the smallest element found into its correct position in the sorted part of the array.

**7. Why do we check if (minIndex != i) before swapping?**

**Answer:**  
This check avoids unnecessary swapping when the smallest element is already in the correct position, optimizing the code slightly.

**8. What is the purpose of printVector in the code?**

**Answer:**  
printVector is a helper function that prints the array/vector elements in order, used to show the array before and after sorting.

**9. What would happen if the swap() was not done inside the if (minIndex != i) check?**

**Answer:**  
The algorithm would still work, but it would perform **unnecessary swaps**, reducing efficiency.

**10. Is Selection Sort a stable sorting algorithm?**

**Answer:**  
**No.**  
Selection Sort is **not stable** by default, because swapping can change the relative order of equal elements.

**🧠 2. Basic and Conceptual Viva Questions (with Answers)**

**1. What is a stable sorting algorithm?**

**Answer:**  
A sorting algorithm is **stable** if two equal elements retain their original relative order after sorting.

**2. Give some examples of stable and unstable sorting algorithms.**

**Answer:**

* **Stable:** Bubble Sort, Insertion Sort, Merge Sort
* **Unstable:** Selection Sort, Quick Sort, Heap Sort

**3. What are the advantages and disadvantages of Selection Sort?**

**Answer:**

**Advantages:**

* Easy to implement
* Performs well on small datasets
* Does not require extra space

**Disadvantages:**

* Inefficient for large datasets (O(n²))
* Not stable

**4. When should you use Selection Sort?**

**Answer:**  
When the dataset is small and memory usage is a concern. It’s also useful in teaching sorting algorithms because of its simplicity.

**5. What is an in-place sorting algorithm?**

**Answer:**  
An algorithm that uses only a constant amount of extra memory (O(1)) for sorting. Selection Sort is in-place.

**6. Can you sort in descending order using selection sort?**

**Answer:**  
Yes. Just modify the comparison condition from arr[j] < arr[minIndex] to arr[j] > arr[maxIndex].

**7. How many total swaps are performed in Selection Sort?**

**Answer:**  
At most **(n - 1)** swaps — one for each pass, only when a new minimum is found.

**8. Is it possible to optimize Selection Sort?**

**Answer:**  
Only slightly — by avoiding unnecessary swaps using if (minIndex != i), but the time complexity remains O(n²).

**9. What are the real-world applications of Selection Sort?**

**Answer:**  
Due to its inefficiency, it's rarely used in practice for large datasets, but it's useful in embedded systems with low memory or as an introductory algorithm for learning sorting.

**10. What is the difference between Bubble Sort and Selection Sort?**

**Answer:**

| **Feature** | **Bubble Sort** | **Selection Sort** |
| --- | --- | --- |
| Swaps | More frequent | Fewer swaps |
| Stable | Yes | No |
| Time Complexity | O(n²) | O(n²) |
| Logic | Compare adjacent | Select min in unsorted |

Would you like a **PDF or printable viva revision sheet** based on this for quick access before your practical?

**Code explanation of Job Sheduling :**Sure, Vishwas! Let’s go through the **Job Scheduling Problem code step by step** in **detail**, explaining **every part of it**, so you’re fully prepared to explain it in a viva or interview.

## ✅ ****🔶 Problem Summary****

We are given a set of N jobs, each with:

* A **job ID** (like A, B, C...)
* A **deadline** (an integer, before which the job must be completed)
* A **profit** (earned only if the job finishes **on or before its deadline**)

**Goal:** Schedule jobs to **maximize total profit**.

* Each job takes **unit time**.
* Only **one job can run at a time** (single processor).
* Use a **greedy strategy** to choose the most profitable jobs first.

## ✅ ****🔷 Code Breakdown****

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

**Explanation:**

* #include <iostream>: For input/output (cin/cout).
* #include <vector>: To use vector as a dynamic array.
* #include <algorithm>: To use sort() and max() functions.
* using namespace std;: To avoid typing std:: every time.

### 🧱 ****Step 1: Define the Job structure****

struct Job {

char id;

int deadline;

int profit;

};

* We define a Job struct that holds the **job ID (char)**, **deadline (int)**, and **profit (int)**.
* This makes each job a single object we can sort and manipulate.

### 🧠 ****Step 2: Comparator Function for Sorting Jobs by Profit****

bool compare(Job a, Job b) {

return a.profit > b.profit;

}

* This function is used by sort() to arrange the jobs in **descending order of profit**.
* So jobs with **higher profit come first**.

### ⚙️ ****Step 3: The Job Scheduling Function****

void jobScheduling(vector<Job>& jobs) {

This function receives a vector of Job objects and performs the main logic.

### 🧮 ****Step 4: Sort Jobs by Profit****

sort(jobs.begin(), jobs.end(), compare);

* This sorts the jobs list using the compare() function.
* Now we’ll process the **most profitable jobs first** — **Greedy Strategy**.

### 🧱 ****Step 5: Find the Maximum Deadline****

int n = jobs.size();

int maxDeadline = 0;

for (auto job : jobs)

maxDeadline = max(maxDeadline, job.deadline);

* n: Number of jobs.
* maxDeadline: We loop through all jobs and find the **largest deadline**.
* This is needed to create a **schedule array of appropriate size**.

### 📅 ****Step 6: Initialize the Schedule****

vector<char> schedule(maxDeadline, '-'); // '-' means empty slot

int totalProfit = 0;

* schedule: A vector of size maxDeadline, initialized with '-' (indicates empty slots).
* Each index in this vector represents **a unit of time**.
  + For example, if max deadline is 5, we have 5 slots: [0, 1, 2, 3, 4]
* totalProfit: Initially zero.

### 🧩 ****Step 7: Schedule Jobs****

for (auto job : jobs) {

// Try to find a free slot from job.deadline - 1 to 0

for (int j = job.deadline - 1; j >= 0; j--) {

if (schedule[j] == '-') {

schedule[j] = job.id;

totalProfit += job.profit;

break;

}

}

}

#### 🔁 Outer Loop: for (auto job : jobs)

* We go through the sorted list of jobs (by decreasing profit).

#### 🔁 Inner Loop: for (int j = job.deadline - 1; j >= 0; j--)

* We try to find the **latest available slot** **on or before the job's deadline**.
* We check from job.deadline - 1 backward to 0.
  + (Slot indices are 0-based)

#### ✅ If a slot is empty:

if (schedule[j] == '-') {

schedule[j] = job.id; // Schedule the job

totalProfit += job.profit; // Add its profit

break; // Exit the loop after scheduling

}

* The job is scheduled in the latest possible free slot.
* Profit is added to the total.

### 📤 ****Step 8: Output the Scheduled Jobs****

cout << "\nScheduled Jobs: ";

for (char c : schedule)

if (c != '-')

cout << c << " ";

cout << "\nTotal Profit: " << totalProfit << endl;

}

* We display the scheduled job IDs.
* Also print the total profit.

### 🏁 ****Step 9: Main Function****

int main() {

int n;

cout << "Enter number of jobs: ";

cin >> n;

vector<Job> jobs(n);

cout << "Enter Job ID, Deadline and Profit for each job:\n";

for (int i = 0; i < n; ++i) {

cout << "Job " << i + 1 << ": ";

cin >> jobs[i].id >> jobs[i].deadline >> jobs[i].profit;

}

jobScheduling(jobs);

return 0;

}

#### 🔷 What happens here:

* Take user input for number of jobs.
* For each job, input id, deadline, and profit.
* Call the jobScheduling() function to process and print results.

## 🧪 ****🔎 Sample Input/Output****

### 👨‍💻 Input:

Enter number of jobs: 5

Enter Job ID, Deadline and Profit for each job:

Job 1: A 2 100

Job 2: B 1 19

Job 3: C 2 27

Job 4: D 1 25

Job 5: E 3 15

### 📤 Output:

Scheduled Jobs: A C E

Total Profit: 142

## 📊 Summary of Key Concepts:

| **Concept** | **Explanation** |
| --- | --- |
| Greedy Algorithm | Choose job with highest profit first |
| Deadline | Job must finish **on or before** this time |
| Unit Time | Every job takes exactly 1 time unit |
| Schedule Vector | Represents available time slots (size = max deadline) |
| Comparator | Sorts jobs by descending profit |
| Total Profit | Sum of profits of scheduled jobs |
| Time Complexity | O(n log n) for sorting + O(n²) for scheduling loop |

Would you like me to help you visualize this with a **dry-run table or diagram** for viva revision?

Sure, Vishwas! Based on your **employee performance evaluation code** in C++, here is a complete set of:

**✅ Expected Viva Questions & Answers**

**→ Based on your problem statement, code, logic, and C++ concepts.**

**🔷 PART 1: CODE-SPECIFIC & LOGIC-BASED QUESTIONS**

**1. ❓ What is the objective of your program?**

**Answer:**  
The program evaluates an employee’s performance based on four criteria — **Punctuality**, **Task Completion**, **Teamwork**, and **Innovation** — each rated from 1 to 10. Based on the number of criteria with a score ≥ 8, it outputs the performance level as **Excellent**, **Good**, **Average**, or **Needs Improvement**.

**2. ❓ How is the performance evaluated?**

**Answer:**  
Each category rated **8 or more** adds 1 to the total score. The final evaluation is:

* **4 points** → Excellent
* **3 points** → Good
* **2 points** → Average
* **0 or 1** → Needs Improvement

**3. ❓ What is the data type of the evaluation result?**

**Answer:**  
It is a string that holds values like "Excellent", "Good", etc., and is returned from the evaluatePerformance function.

**4. ❓ What are the input values in your program?**

**Answer:**  
The user provides **4 integers** (from 1 to 10), one for each performance metric:

* Punctuality
* Task Completion
* Teamwork
* Innovation

**5. ❓ Why is the threshold for each metric set to 8?**

**Answer:**  
A threshold of **8 or more** indicates high performance in that metric. This helps in categorizing employees based on strong ratings in majority criteria.

**6. ❓ What is the return type of your function and why?**

**Answer:**  
The return type of the function is string because we return textual performance evaluations like "Good" or "Average".

**7. ❓ What happens if the user enters a score greater than 10 or less than 1?**

**Answer:**  
Currently, the program does **not include input validation**, so invalid values like -1 or 15 will be accepted, which can lead to incorrect evaluations. Input validation should be added.

**🔷 PART 2: BASIC C++ CONCEPTUAL QUESTIONS**

**8. ❓ What is a function in C++?**

**Answer:**  
A function is a **block of reusable code** that performs a specific task. In this program, evaluatePerformance() is a user-defined function that returns a performance level based on inputs.

**9. ❓ What is the role of main() in C++?**

**Answer:**  
The main() function is the **entry point** of any C++ program. The program starts executing from main().

**10. ❓ What is cin and cout?**

**Answer:**

* cin is used to take **input from the user**.
* cout is used to **display output** on the screen.  
  Both are part of the iostream library.

**11. ❓ What is the purpose of the using namespace std; line?**

**Answer:**  
It allows you to use standard C++ library names (like cin, cout, string) **without prefixing them with std::**.

**12. ❓ What is the scope of a variable?**

**Answer:**  
The **scope** refers to where a variable is accessible in the program. For example, variables like punctuality in main() are **local to main**, and score in the function is **local to that function**.

**13. ❓ What is the difference between int and string data types?**

**Answer:**

* int is used to store **integer values**.
* string is used to store **text (sequence of characters)**.

**14. ❓ What is a conditional statement? Which ones are used here?**

**Answer:**  
Conditional statements allow decision-making in code. Here, if, else if, and else statements are used to check performance scores and return a result.

**15. ❓ What will the program output if all ratings are below 8?**

**Answer:**  
If all ratings are below 8, score will remain 0, and the output will be:

Performance Evaluation Result: Needs Improvement

**🔷 PART 3: THEORY & LOGICAL EXTENSION QUESTIONS**

**16. ❓ How can this program be extended in the future?**

**Answer:**

* Add more performance metrics (e.g., leadership, problem-solving)
* Use input validation (restrict range 1–10)
* Store evaluations in a file or database
* Add graphical interface for evaluation dashboard

**17. ❓ What is the space and time complexity of this program?**

**Answer:**

* **Time complexity:** O(1) – as it performs fixed operations
* **Space complexity:** O(1) – constant number of variables used

**18. ❓ What are the advantages of using functions?**

**Answer:**

* **Code reusability**
* **Modularity** (break down into small parts)
* **Ease of debugging and testing**

**19. ❓ What is the importance of performance evaluation in real-world applications?**

**Answer:**  
Performance evaluation helps in:

* **Identifying strong and weak areas** of employees
* **Promotions and appraisals**
* **Training needs assessment**
* **Motivating improvement**

**20. ❓ Can we replace the string return type with integers or enums?**

**Answer:**  
Yes, instead of returning strings like "Excellent", we can return:

* Integers like 4, 3, 2, 1
* Or better: use enum to define categories like EXCELLENT, GOOD, etc., for better performance and type safety.

**✅ Summary Table**

| **Category** | **Question Examples** |
| --- | --- |
| Logic & Code | How is score calculated? Why ≥8? What is returned? |
| C++ Basics | What is main()? cin vs cout? Data types? |
| Conditions | Role of if-else, why strings? |
| Extensions | Can we improve this code? Real-world use? |
| Evaluation | What does the result mean practically? |

Would you like this compiled in a **PDF viva sheet** for print/revision?

Great observation, Vishwas! Since your project is related to **Performance Evaluation**, it can also be categorized under **Expert Systems** in AI (because it's rule-based decision-making). So yes, **viva questions related to Expert Systems and basic AI** might be asked too — especially if the context is "Decision Support System" or "Rule-Based Evaluation".

Here’s an **extended set of viva questions and answers** including **Expert Systems and AI-related theory** that can be asked based on your project:

**🔷 PART 4: Expert System & AI-Related Viva Questions**

**21. ❓ What is an Expert System?**

**Answer:**  
An **Expert System** is an AI-based program that simulates the decision-making ability of a human expert. It uses **rules** (IF-THEN) to solve problems in a specific domain. In your project, the evaluation logic is similar to an expert system since it makes decisions based on input values.

**22. ❓ Is your program an expert system? Why or why not?**

**Answer:**  
Yes, it is a **basic expert system** because:

* It uses **rule-based logic** (e.g., if punctuality ≥ 8, increase score)
* It evaluates and gives **human-like decisions** ("Good", "Excellent", etc.)
* It applies **domain knowledge** (employee performance rules)

**23. ❓ What are the main components of an Expert System?**

**Answer:**

1. **Knowledge Base** – contains facts and rules (in our case, rules for evaluation)
2. **Inference Engine** – applies logic to derive conclusions (our evaluatePerformance() function)
3. **User Interface** – allows user to input data and receive output (cin and cout in our case)

**24. ❓ What is the Inference Engine?**

**Answer:**  
The **Inference Engine** is the part of an expert system that applies logical rules to the knowledge base to derive new information. In our program, the function evaluatePerformance() acts like an inference engine.

**25. ❓ What is the difference between AI and Expert System?**

**Answer:**

| **Aspect** | **AI** | **Expert System** |
| --- | --- | --- |
| Learning | Can learn from data (e.g., ML) | Rule-based, does not learn |
| Flexibility | More flexible | Fixed rules |
| Example | ChatGPT, self-driving car | Medical diagnosis system, performance evaluation |

**26. ❓ What is a rule-based system?**

**Answer:**  
A **rule-based system** uses predefined "IF-THEN" rules to make decisions. Your program uses rules like:

if (punctuality >= 8) score++;

This is a simple rule-based decision-making logic.

**27. ❓ How would you improve your system using AI?**

**Answer:**

* Use **Machine Learning** to learn from historical data instead of fixed rules.
* Use **Natural Language Processing** to accept input in user-friendly ways.
* Store employee performance history and analyze trends using AI.

**28. ❓ What is Knowledge Representation?**

**Answer:**  
**Knowledge Representation** is the way information (rules, facts) is structured in a system. In our program, the rules are hard-coded as if statements, which is a form of procedural knowledge representation.

**29. ❓ What is the difference between Forward and Backward Chaining?**

**Answer:**

* **Forward Chaining**: Starts with known facts and applies rules to infer conclusions.
* **Backward Chaining**: Starts with a goal and works backward to see if facts support it.

In our case, we use **forward chaining** — we apply known input values to rules to reach a conclusion (performance result).

**30. ❓ Give a real-life example of an expert system.**

**Answer:**

* **MYCIN** – medical diagnosis expert system.
* **Loan approval systems** in banks.
* **HR performance evaluations** (like our program).
* **Diagnostic systems** in hardware servicing.

**✅ BONUS: Combined Technical-Theoretical Question**

**31. ❓ Can your system be considered a Decision Support System (DSS)?**

**Answer:**  
Yes. A **DSS** helps managers and users make informed decisions based on data. Our system evaluates performance and helps HR/managers decide employee ratings or promotions.

**✅ Summary Table: Extended Viva Scope**

| **Category** | **Sample Questions** |
| --- | --- |
| Expert System | What is it? Components? Is this one? |
| AI Concepts | Difference between AI and Expert Systems |
| Rule-Based Logic | What are rules? Inference engine? |
| Knowledge Representation | How is knowledge stored? |
| Chaining Techniques | Forward vs Backward chaining |
| Real-world Examples | Medical, HR, Financial expert systems |
| DSS Integration | How your system can help in decision-making? |

Would you like me to generate a **full PDF file with all these viva questions and answers formatted neatly** for revision or printing?

**✅ 1. Expected Questions (Based on Code + Problem Statement)**

**Q1: What is the Job Scheduling Problem?**

**Answer:**  
The Job Scheduling Problem aims to schedule a set of N jobs on a single processor such that:

* Each job has a **deadline** and a **profit**.
* Each job takes **unit time**.
* A job earns **profit only if completed on or before its deadline**.  
  The goal is to maximize the **total profit** by scheduling jobs optimally.

**Q2: What strategy or algorithm is used to solve this problem?**

**Answer:**  
This problem is solved using a **Greedy Algorithm**:

* Jobs are sorted in **descending order of profit**.
* Each job is scheduled in the **latest available time slot before or at its deadline**.
* This ensures high-profit jobs are prioritized.

**Q3: Why are jobs sorted in descending order of profit?**

**Answer:**  
Because the **greedy strategy** is to **pick the most profitable job first** and try to schedule it before its deadline. This increases the total profit.

**Q4: What is the purpose of the schedule vector in your code?**

**Answer:**  
The schedule vector represents the time slots available (1 slot per time unit up to the maximum deadline). Each index holds the ID of the scheduled job or '-' if the slot is empty.

**Q5: What does the inner for loop (for (int j = job.deadline - 1; j >= 0; j--)) do?**

**Answer:**  
It tries to find the **latest available time slot** (before or at the job's deadline) to schedule the job. If an empty slot is found, the job is scheduled there.

**Q6: How is total profit calculated?**

**Answer:**  
Whenever a job is scheduled successfully within its deadline, its profit is **added to totalProfit**.

**Q7: What does maxDeadline represent and how is it used?**

**Answer:**  
maxDeadline is the **maximum deadline** among all jobs. It defines the **number of time slots** required to schedule the jobs.

**Q8: Is your algorithm optimal?**

**Answer:**  
For the given constraints (unit time per job, profit only if scheduled before deadline), the **greedy algorithm is optimal** and gives the correct result.

**Q9: What happens if multiple jobs have the same deadline?**

**Answer:**  
The algorithm chooses the one with the **higher profit first** because of the sorting step. The job with lower profit will only be scheduled if a free time slot is still available.

**Q10: What is the time complexity of this algorithm?**

**Answer:**

* Sorting the jobs: **O(n log n)**
* Scheduling jobs: In worst case, each job may check all previous slots ⇒ **O(n²)**  
  So total time complexity is **O(n²)** (but more optimized than brute force).

**🧠 2. Basic/Conceptual Questions (with Answers)**

**Q1: What is a Greedy Algorithm?**

**Answer:**  
A Greedy Algorithm makes **locally optimal choices** at each step with the hope of finding the **global optimum**. It builds up a solution piece by piece.

**Q2: What are the characteristics of problems solvable by greedy methods?**

**Answer:**

* **Greedy choice property**: A globally optimal solution can be arrived at by choosing the locally optimal choice.
* **Optimal substructure**: Optimal solution to the problem contains optimal solutions to subproblems.

**Q3: What are real-life applications of job scheduling algorithms?**

**Answer:**

* CPU task scheduling
* Deadline-based task management
* Advertising job slots
* Cloud computing resources scheduling
* Print job prioritization in printers

**Q4: What is the difference between greedy and dynamic programming?**

**Answer:**

| **Aspect** | **Greedy** | **Dynamic Programming** |
| --- | --- | --- |
| Strategy | Local optimum | Global optimum (by solving subproblems) |
| Optimality | Not always guaranteed | Always optimal (if conditions met) |
| Memory | Low (no table) | High (uses table/memoization) |

**Q5: Can you name other problems solved using greedy approach?**

**Answer:**

* Activity Selection Problem
* Fractional Knapsack Problem
* Huffman Coding
* Minimum Spanning Tree (Prim's & Kruskal's)
* Dijkstra’s Shortest Path (with heap)

**Q6: Can a job be scheduled after its deadline?**

**Answer:**  
**No.** If a job is not scheduled within its deadline, it is considered missed and **earns zero profit**.

**Q7: What modifications would be needed if jobs took variable time instead of unit time?**

**Answer:**  
Then it becomes a **more complex optimization problem**, possibly requiring **Dynamic Programming or other techniques** (e.g., Weighted Job Scheduling).

**Q8: Why do we check schedule[j] == '-' before scheduling?**

**Answer:**  
To make sure the time slot is **free**. If it's already occupied by another job, we continue checking earlier slots.

**Q9: Can this algorithm be modified to minimize penalty instead of maximize profit?**

**Answer:**  
Yes. Instead of maximizing profit, we can consider **penalty for missing jobs** and try to **minimize total penalty** — it’s just a variant of the same problem.

**Q10: Why do we use vector<char> schedule(maxDeadline, '-')?**

**Answer:**  
To create a vector of maxDeadline slots, each initialized to '-' to indicate that the slot is **initially empty**.

Would you like a **flowchart or dry-run diagram** of this algorithm for your viva or exam prep?