Task 5: implement various searching and sorting operations in python programing

Aim: To implement various searching and sorting operations in python programming.

5.1) A company stores employee records in a list of dictionaries, where each dictionary contains id, name and department. write a function find-employee-by-id that takes this list and a target employee ID as arguments and returns the dictionary of the employee with the matching ID, or None if no such employee is found.

Algorithm:

1. Input Definition:
2. Define the function find-employee-by-id that takes two parameters.
   a. A list of dictionaries (employees), where each dictionary represents an employee record. keys, id, names, and department.
   b. An integer (target-id) representing the employee ID to be Searched.
3. Iterate through the list.
   Use a for loop to iterate through each dictionary in the employees list
   in the employees
4. Check for matching Record or ID with in the loop, check if the id field of the current dictionary matches the target_id
5. Return matching Record.
   If a matching is found, return the current dictionary
6. Handle No match
   If the loop completes without finding a match, return None.

Program 5.1

```
def find-employee-by-id (employees, target_id):
    for employee in employees:
        if employee['id'] == target_id
```

Out Put:-

{'id': 2, 'name': 'Bob', 'department': 'Engineering'}

```
    return employee
return None
# Test the function
employees = [
{'id': 1, 'name': 'Alice', 'department': 'HR'},
{'id': 2, 'name': 'Bob', 'department': 'Engineering'},
{'id': 3, 'name': 'Charlie', 'department': 'Sales'},
]
print(find_employee_by_id(employees, 2))
```

You are developing a grade management system for a school. The system maintains a list of student records, where each record is represented as a dictionary containing a student's name and score.

Algorithm:

1. Initialization
   * Get the length of the students list and store it in n
2. Outer Loop:
   * Iterate from i=0 to n-1 (inclusive). This loop represents the number of passes through the list
3. Track swaps:
   * Initialize a boolean variable swapped to False. This variable will track if any swaps are made in the current pass.
4. Inner Loop:
   * Iterate from i=0 to n-i-2 (inclusive). This loop compares adjacent elements in the list and performs swaps if necessary.
5. Compare and swap:
   * For each pair of adjacent elements (i.e, students[j] and students[j+1]):
   - Compare their score values.
   - If students[j]['score'] > students[j+1]['score'], swap the two elements.
   - Set swapped to true to indicate that a swap was made.
6. Early Termination.
   - After each pass of the inner loop, check if swapped is false. If no swaps were made during the pass, the

list is already sorted, and you can break out of the on the loop early.
* completion
* The function modifies the students list in place, sorting it by score.

Program 5.2:-

```
def bubble_sort_scores(students):
    n = len(students)
    for i in range(n):
# Track if any swap is made in this pass
        swapped = False
    for j in range(0, n-i-1):
    if students[j]['score'] > students[j+1]['score']:
# Swap if the score of the current student is greater
than the text students[j], students[j+1] = students[j+1],
students[j]  swapped = True
# If no two elements were swapped, the list is already
sorted
        if not swapped:
        break
# Example usage
students = [
{'name': 'Alice', 'score': 88},
{'name': 'Bob', 'score': 95},
{'name': 'Charlie', 'score': 75},
{'name': 'Diana', 'score': 88}
]
print("Before sorting:")
for student in students:
print(student)
bubble_sort_scores(students)
print("\n After sorting:")
for student in students:
```

Output:

Before Sorting:

{'name': 'Alice', 'score': 88}

{'name': 'Bob', 'score': 95}

{'name': 'charlie', 'score': 75}

{'name': 'Diana', 'score': 85}

After sorting:

{'name': 'charlie', 'score': 75}

{'name': 'Diana', 'score': 85}

{'name': 'Alice', 'score': 88}

{'name': 'Bob', 'score': 95}

print (student)

VEL TECH	
EX. NO.	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	15
SIGNATURE WITH DATE	3/9/25

Result: Thus, the program for various searching and sorting operations is executed and verified succesfully.