

5.	20/08/25	programming. Implement various searching and Sorting operations in python programming.	15	20/08/25

20/8/25

Task-5: Implement various searching and sorting operations In Python Programming

(a) Library Book Search

Aim: To search for a book in a library catalog using linear search for unsorted lists and binary search for sorted lists.

Algorithms:

1. Linear Search:

- Check each book in the list one by one.
- If found, return its position.
- If not found after checking all books, return -1.

2. Binary Search:

- Requires the list to be sorted.
- Compare the target with middle element.
- If equal, return the position.
- If target is smaller, search the left half.
- If target is larger, search the right half.
- Repeat until found or search space exhausted.

Program:

Library Book Search

```
def linear_search (books, target):  
    for i in range (len (books)):  
        if books[i] == target:  
            return i
```

```
    return -1
```

```
def binary_search (books, target):
```

```
    low, high = 0, len (books) - 1
```

```
    while low < high:
```

```
        mid = (low + high) // 2
```

```
        if books[mid] == target:
```

```
            return mid
```

```
        elif books[mid] < target:
```

```
            low = mid + 1
```

```
    else:
```

```
        high = m - 1
```

```
    return -1
```

The search for a book in a library is a linear search. It involves checking each book in the list until the desired book is found. If the book is not found, it returns a message indicating that the book is not in the library.

The following code implements a linear search algorithm in Python:

```

def linear_search(library, book):
    for i in range(len(library)):
        if library[i] == book:
            return i
    return -1

library = ['python', 'Java', 'C++', 'JavaScript', 'HTML']
book = 'C++'
position = linear_search(library, book)

if position != -1:
    print(f'Book found at position {position}.')
else:
    print('Book not found.')
  
```

In this code, the `linear_search` function takes a list of books (`library`) and a target book (`book`) as input. It iterates through the list using a `for` loop. If it finds the book at index `i`, it returns `i`. If it reaches the end of the list without finding the book, it returns `-1`. The main part of the code creates a library list, chooses a book to search for, and prints the result.

Output:

Library Books: ['python', 'Java', 'C++', 'JavaScript', 'HTML']
 List is sorted: False
 Enter book to search: C++
 Book found at position 3: (using linear search)


```

books = ["Python", "Java", "C++", "JavaScript", "HTML"]
print ("Library Books:", books)
is_sorted = books == sorted (books)
print ("list is sorted:", is_sorted)
target = input ("tutor book to search :")
if is_sorted:
    result = binary_search (books, target)
    method = "Binary search"
else:
    result = linear_search (books, target)
    method = "linear search"
if result != -1:
    print (Books found out position {result + 1} using {method}")
else:
    print ("Book not found")

```

Result:-

The program to implement both search algorithms, to detect that the list was not sorted and used linear search to find "C++" at position 3 was successfully executed.

1/2/25

(b) Student Grade Organizer

Aim - To sort student grades using different algorithms and display the top 3 scores.

Algorithm:-

1. Bubble sort (Ascending):

- Compare adjacent elements & swap it in wrong order
- Repeat until no more swaps are needed.

2. Selection sort (Descending):

- Find the max element and swap it first position.
- Repeat for remaining elements

3. Top 3 scores:

- After sorting in descending order, the first three elements are the top scores.

Program:-

```
def bubble_sort_asc(grades):
    n = len(grades)
    for i in range(n):
        for j in range(0, n-i-1):
            if grades[j] > grades[j+1]:
                grades[j], grades[j+1] = grades[j+1], grades[j]
    return grades

def selection_sort_desc(grades):
    n = len(grades)
    for i in range(n):
        max_idx = i
        for j in range(i+1, n):
            if grades[j] > grades[max_idx]:
                max_idx = j
        grades[i], grades[max_idx] = grades[max_idx], grades[i]

grades = [85, 92, 78, 90, 65, 88, 72]
print("original grades:", grades)
asc_sorted = bubble_sort_asc(grades.copy())
desc_sorted = selection_sort_desc(grades.copy())
print("Ascending order (Bubble sort):", asc_sorted)
```


Output:-

Original Grades : [85, 92, 78, 90, 65, 88, 72]

Ascending order (Bubble sort) : [65, 72, 78, 85, 88, 90, 92]

Descending order (selection sort) : [92, 90, 88, 85, 78, 72, 65]

Top 3 scores : [92, 90, 88]

```
print("Descending order (selection sort)!", desc - sorted)
print ("Top 3 scores!", desc - sorted [:3])
```

VEL TECH - CSE	
EX NO.	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
CORD (5)	5
TOTAL (20)	25
DATE	

Result:-

Thus, the student grade organizer in both ascending and descending order using different algorithms are executed successfully.