

TASK 5: Implement various Searching and Sorting Operations in python programming

5(a) Linear Search

Given an array arr[] of n elements, write a function to search a given element x in arr[] using Linear search.

Examples :

Input : arr[] = {10, 20, 80, 30, 60, 50,
110, 100, 130, 170}

x = 110;

Output : 6

Element x is present at index 6

Input : arr[] = {10, 20, 80, 30, 60, 50,
110, 100, 130, 170}

x = 175;

Output : -1

Element x is not present in arr[].

PROGRAM


```
def search(arr, x):  
    for i in range(len(arr)):  
        if arr[i] == x:  
            return i  
    return -1
```

Example usage:

```
arr = [10, 20, 80, 30, 60, 50, 110, 100, 130, 170]  
x = 110  
result = search(arr, x)  
if result != -1:  
    print("Element", x, "is present at index", result)  
else:  
    print("Element", x, "is not present in the array")
```

OUTPUT

```
= RESTART: E:/SUBJECT MATERIALS/veltech/subjects  
actise/Task 5/5b.py  
Element 110 is present at index 6  
>> |
```



5b Binary Search

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return -1. Apply an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: `nums = [-1,0,3,5,9,12]`, `target = 9`

Output: 4

Explanation: 9 exists in `nums` and its index is 4

Example 2:

Input: `nums = [-1,0,3,5,9,12]`, `target = 2`

Output: -1

Explanation: 2 does not exist in `nums` so return -1

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 < \text{nums}[i], \text{target} < 10^4$
- All the integers in `nums` are **unique**.
- `nums` is sorted in ascending order.

PROGRAM

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

```
    left=0
```

```
    right=len(nums)- 1
```

```
    while left <= right:
```

```
        mid = left + (right - left) // 2
```

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

```
            return mid
```

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

```
            left = mid + 1
```

```
        else:
```

```
            right = mid - 1
```

```
    return -1
```

List is initialised

```
nums = [-1, 0, 3, 5, 9, 12]
```

```
target = 9
```

```
print(binary_search(nums, target))
```

OUTPUT

```
>>> 4
```

python

Copy code

```
nums = [-1, 0, 3, 5, 9, 12]
```

```
target = 9
```

. Initialize `'left'` and `'right'` pointers:

- `'left = 0'` (index of the leftmost element)

- `'right = 5'` (index of the rightmost element)

. Enter the `'while'` loop:

- Calculate `'mid = 0 + (5 - 0) // 2 = 2'`.

- Compare `'nums[2]'` with `'target'`:

- `'nums[2]'` is '3', which is less than '9', so adjust `'left = mid + 1 = 3'`.

. `'left = 3', 'right = 5'`. Calculate `'mid = 3 + (5 - 3) // 2 = 4'`.

- Compare `'nums[4]'` with `'target'`:

- `'nums[4]'` is '9', which is equal to the `'target'`. Return `'4'`.

The target '9' is found at index '4'. So, the output of the function will be '4'.

5 (c) Find the sequence of given integers

Sort the given set of integers using bubble sort and find the smallest and largest among given set of integers.

Step 1: Get the number of integers

Step 2: Receive the integers

Step 3: Sort the integers using bubble sort

Step 4: Print the start and end of sequence

Test case 1

INPUT

4
99
5
6
97

OUTPUT

Sequence of integers: 4 to 99

WHAT IS BUBBLE SORT?

Bubble Sort algorithm, which sorts an array by repeatedly comparing adjacent elements and swapping them if they are in the wrong order. The algorithm iterates through the array multiple times, with each pass pushing the largest unsorted element to its correct position at the end.

PROGRAM

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

def find_sequence(arr):
    smallest = arr[0]
    largest = arr[-1]
    print("Sequence of integers:", smallest, "to", largest)
```

Step 1: Get the number of integers

```
n = int(input("Enter the number of integers: "))
```

Step 2: Receive the integers

```
integers = []
print("Enter the integers:")
for _ in range(n):
    integers.append(int(input()))
```

Step 3: Sort the integers using bubble sort

```
bubble_sort(integers)
```

Step 4: Print the start and end of sequence

```
find_sequence(integers)
```

OUTPUT

```
Enter the number of integers: 4
Enter the integers:
10
50
70
20
Sequence of integers: 10 to 70
>
```

5d) SELECTION SORT

Imagine you are working on a project where you need to sort a list of students' exam scores in ascending order. You decide to use the Selection Sort algorithm for this task.

INPUT

exam_scores = [87, 65, 92, 78, 55, 70, 82]

OUTPUT

Initial exam scores: [87, 65, 92, 78, 55, 70, 82]

Sorting using Selection Sort:

After iteration 1 : [55, 65, 92, 78, 87, 70, 82]

After iteration 2 : [55, 65, 92, 78, 87, 70, 82]

After iteration 3 : [55, 65, 70, 78, 87, 92, 82]

After iteration 4 : [55, 65, 70, 78, 87, 92, 82]

After iteration 5 : [55, 65, 70, 78, 82, 92, 87]

After iteration 6 : [55, 65, 70, 78, 82, 87, 92]

After iteration 7 : [55, 65, 70, 78, 82, 87, 92]

Final sorted exam scores: [55, 65, 70, 78, 82, 87, 92]

WHAT IS SELECTION SORT

Selection Sort is a simple sorting algorithm that repeatedly selects the smallest (or largest) element from the unsorted portion of the array and moves it to the beginning (or end) of the sorted portion.

SELECTION SORT

Here's a simple example of Selection Sort in action:

Consider the array `[64, 25, 12, 22, 11]`:

1. **Initial State:** Sorted portion: [] Unsorted portion: [64, 25, 12, 22, 11]
2. **Iteration 1:** Find the smallest element in the unsorted portion (11) and swap it with the leftmost unsorted element (64). Sorted portion: [11] Unsorted portion: [25, 12, 22, 64]
3. **Iteration 2:** Find the smallest element in the unsorted portion (12) and swap it with the leftmost unsorted element (25). Sorted portion: [11, 12] Unsorted portion: [25, 22, 64]
4. **Iteration 3:** Find the smallest element in the unsorted portion (22) and swap it with the leftmost unsorted element (25). Sorted portion: [11, 12, 22] Unsorted portion: [25, 64]
5. **Iteration 4:** Find the smallest element in the unsorted portion (25) and swap it with the leftmost unsorted element (64). Sorted portion: [11, 12, 22, 25] Unsorted portion: [64]
6. **Iteration 5:** Find the smallest element in the unsorted portion (64) and swap it with the leftmost unsorted element (64). Sorted portion: [11, 12, 22, 25, 64] Unsorted portion: []

The array is now sorted.

PROGRAM:

```
def selection_sort(scores):
    n = len(scores)
    for i in range(n):
        min_index = i
        for j in range(i+1, n):
            if scores[j] < scores[min_index]:
                min_index = j
        scores[i], scores[min_index] = scores[min_index], scores[i]
        print("After iteration", i+1, ":", scores)

exam_scores = [87, 65, 92, 78, 55, 70, 82]
print("Initial exam scores:", exam_scores)
print("Sorting using Selection Sort:")
selection_sort(exam_scores)
print("Final sorted exam scores:", exam_scores)
```

OUTPUT

```
= RESTART: E:/SUBJECT MATERIALS/veltech/subjects/WS 23-24/python/lab task/lab pr
actise/Task 5/5dselection sort.py
Initial exam scores: [87, 65, 92, 78, 55, 70, 82]
Sorting using Selection Sort:
After iteration 1 : [55, 65, 92, 78, 87, 70, 82]
After iteration 2 : [55, 65, 92, 78, 87, 70, 82]
After iteration 3 : [55, 65, 70, 78, 87, 92, 82]
After iteration 4 : [55, 65, 70, 78, 87, 92, 82]
After iteration 5 : [55, 65, 70, 78, 82, 92, 87]
After iteration 6 : [55, 65, 70, 78, 82, 87, 92]
After iteration 7 : [55, 65, 70, 78, 82, 87, 92]
Final sorted exam scores: [55, 65, 70, 78, 82, 87, 92]
```