CECOS UNIVERSITY

DATA STRUCTURE AND ALGORITHUMS

LAB TASK 2

Submitted by:
Misbah Ullah Jan
CU-4273-2023

Submitted to: HIJAB DURRANI
Date: October 30 , 2025

1: Write a program to implement Binary Search using recursion instead of iteration.

Algorithm: Binary Search using Recursion

1: Start with a sorted array and a target element to search.

2: Set low = 0 and high = n - 1.

3: If low > high, return "Not Found".

4: Find mid = (low + high) // 2.

5: If arr[mid] == target, return mid.

6: If arr[mid] > target, recursively call binary search on the **left half** (low to mid - 1).

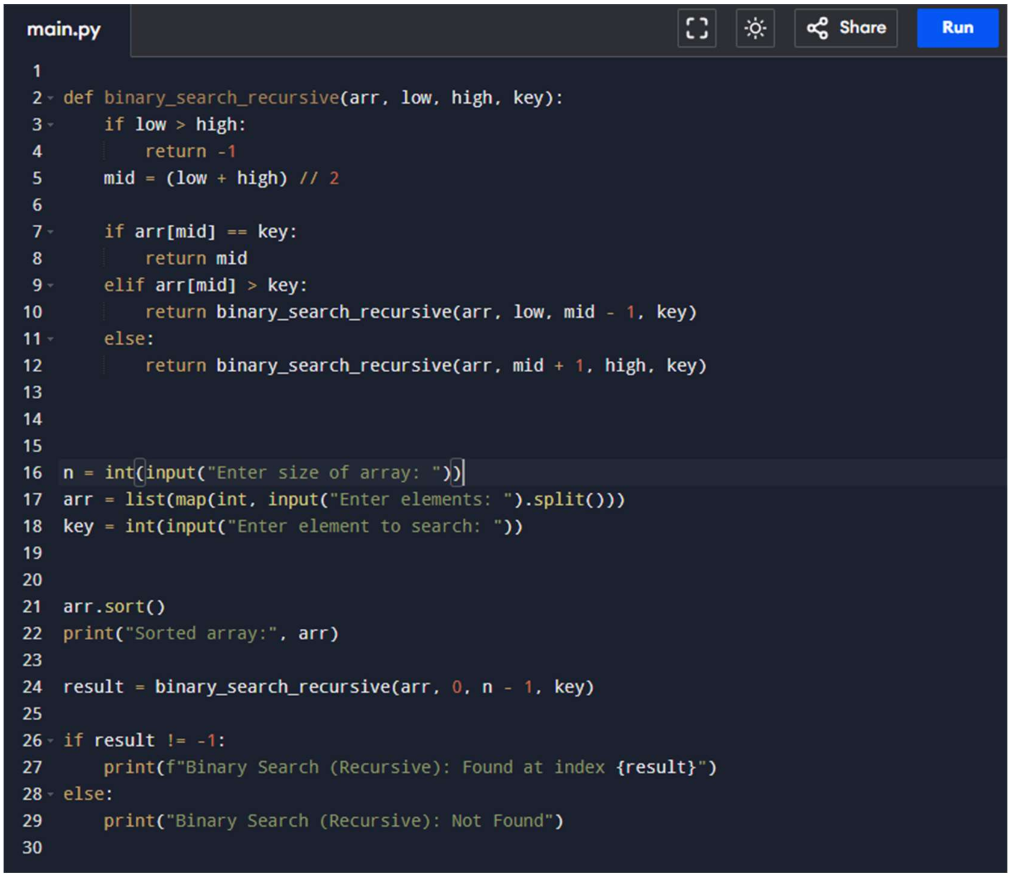
7: Else, recursively call binary search on the **right half** (mid + 1 to high).

8: Repeat steps 3–7 until the element is found or the search space is empty.

Python Code: Binary Search (Recursive):

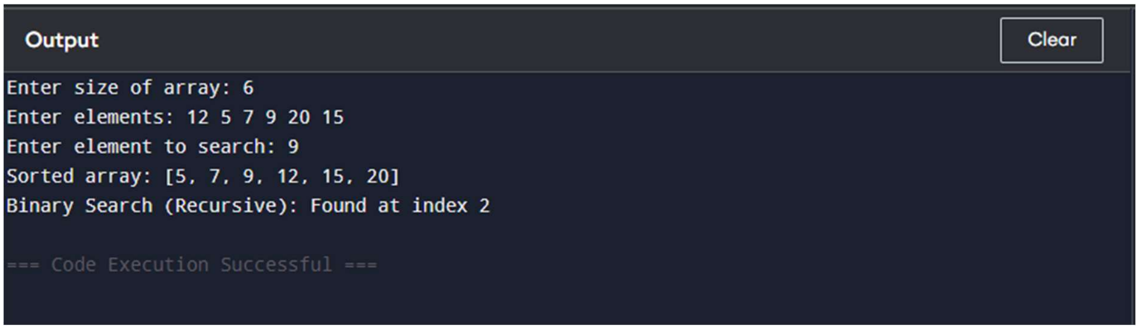
```
def binary_search_recursive(arr, low, high, key):
    if low > high:
        return -1
    mid = (low + high) // 2

    if arr[mid] == key:
        return mid
    elif arr[mid] > key:
        return binary_search_recursive(arr, low, mid - 1, key)
    else:
        return binary_search_recursive(arr, mid + 1, high, key)



n = int(input("Enter size of array: "))
```

```
arr = list(map(int, input("Enter elements: ").split()))

key = int(input("Enter element to search: "))



arr.sort()

print("Sorted array:", arr)


result = binary_search_recursive(arr, 0, n - 1, key)


if result != -1:

    print(f"Binary Search (Recursive): Found at index {result}")

else:

    print("Binary Search (Recursive): Not Found")
```



```

def binary_search_recursive(arr, low, high, key):
    if low > high:
        return -1
    mid = (low + high) // 2

    if arr[mid] == key:
        return mid
    elif arr[mid] > key:
        return binary_search_recursive(arr, low, mid - 1, key)
    else:
        return binary_search_recursive(arr, mid + 1, high, key)



n = int(input("Enter size of array: "))
arr = list(map(int, input("Enter elements: ").split()))
key = int(input("Enter element to search: "))


arr.sort()
print("Sorted array:", arr)

result = binary_search_recursive(arr, 0, n - 1, key)

if result != -1:
    print(f"Binary Search (Recursive): Found at index {result}")
else:
    print("Binary Search (Recursive): Not Found")

```

OUTPUT:



```
Enter size of array: 6
Enter elements: 12 5 7 9 20 15
Enter element to search: 9
Sorted array: [5, 7, 9, 12, 15, 20]
Binary Search (Recursive): Found at index 2

=== Code Execution Successful ===
```

2: Modify both Linear Search and Binary Search to count and print the number of comparisons made while searching. Compare results for the same input

Algorithms

1. Linear Search Algorithm

1. **Start from index 0.**
2. **Compare the target with each element sequentially.**
3. **Count each comparison.**
4. **If element matches, return index and comparison count.**
5. **If end of array is reached, return "Not Found" and total comparisons.**

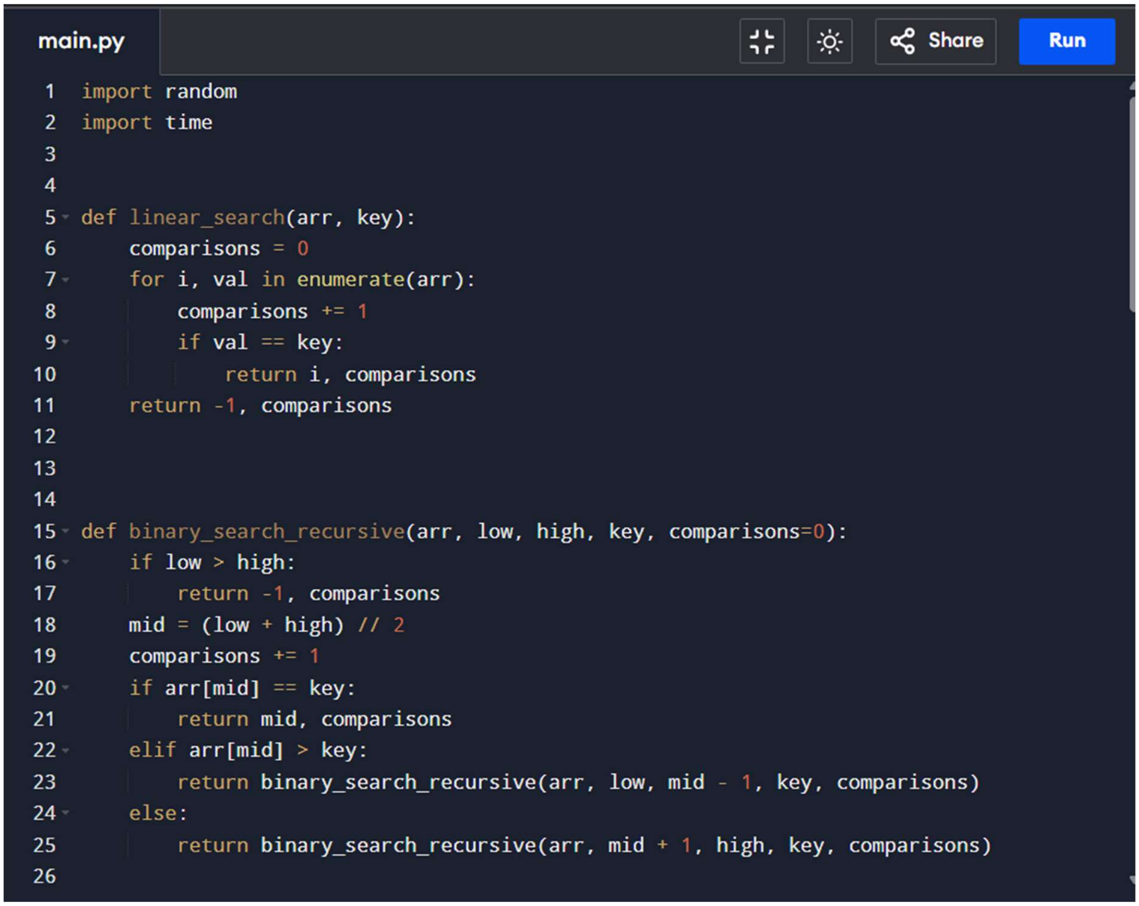
2. Recursive Binary Search Algorithm

1. **Set low = 0, high = n – 1.**
2. **If low > high, return "Not Found".**
3. **Find mid = (low + high) // 2.**
4. **Count this comparison.**
5. **If arr[mid] == target, return index.**
6. **If arr[mid] > target, recursively search the left half.**
7. **Else, recursively search the right half.**
8. **Return total comparisons.**

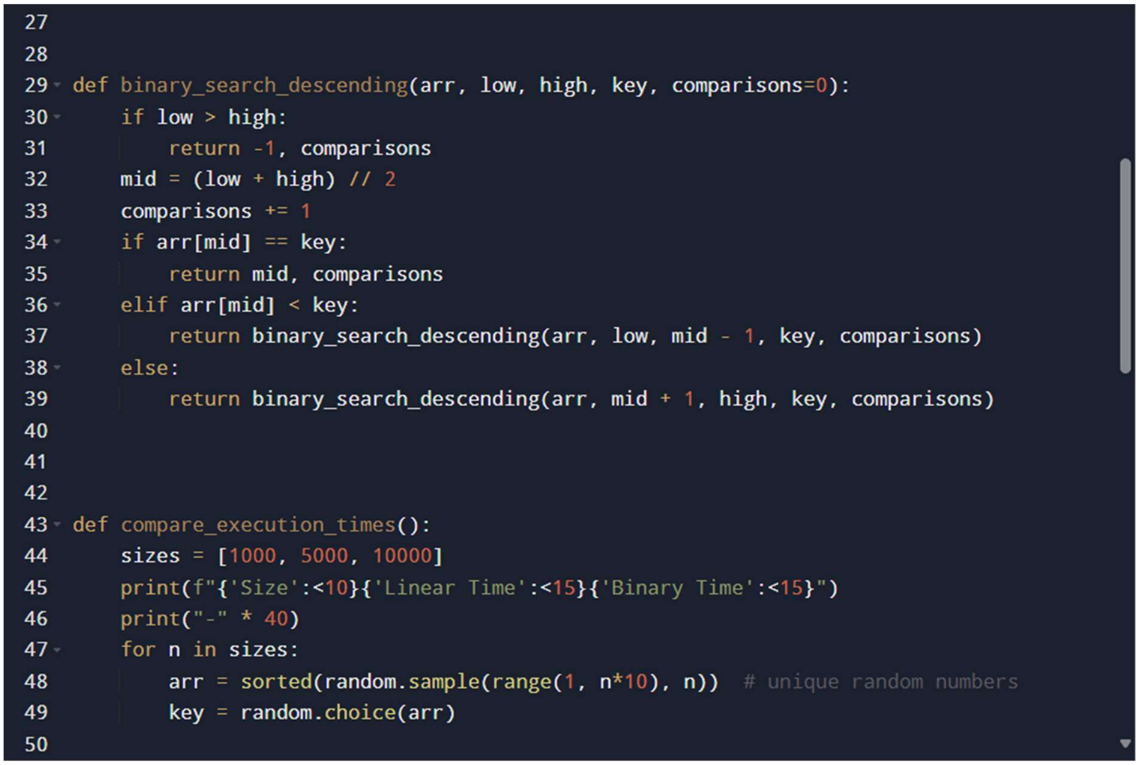
3. Modified Binary Search (Descending Order)

Same as above, but reverse comparisons:

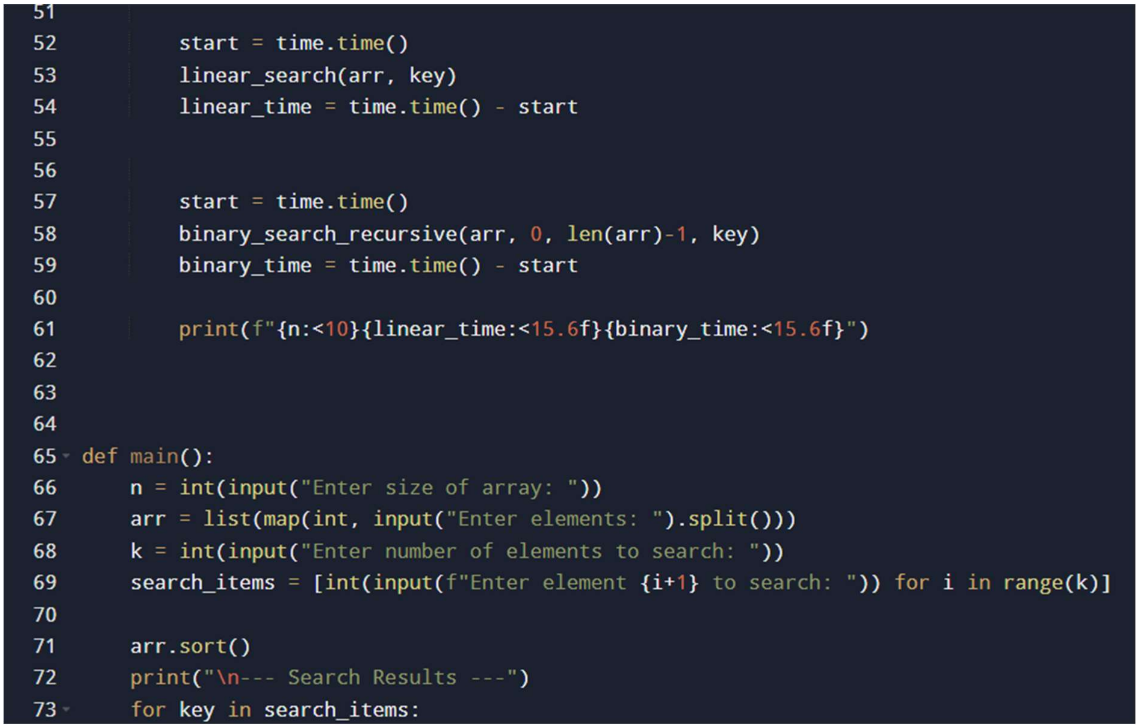
- **If arr[mid] < target, search left half.**
- **If arr[mid] > target, search right half.**



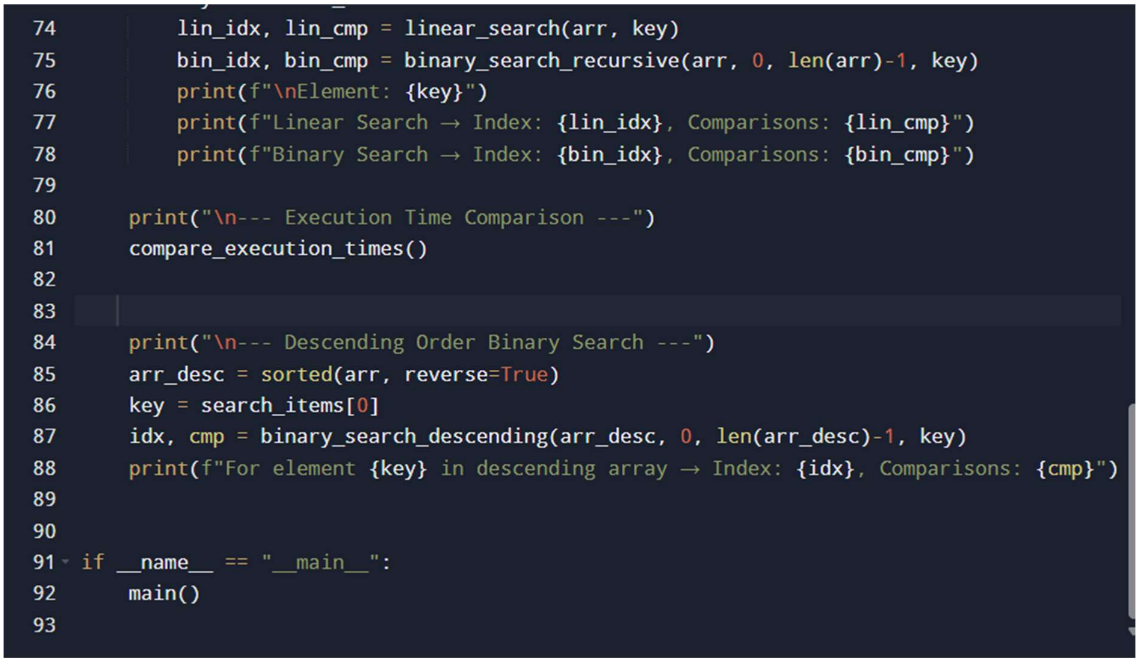
```python
import random
import time


def linear_search(arr, key):
    comparisons = 0
    for i, val in enumerate(arr):
        comparisons += 1
        if val == key:
            return i, comparisons
    return -1, comparisons



def binary_search_recursive(arr, low, high, key, comparisons=0):
    if low > high:
        return -1, comparisons
    mid = (low + high) // 2
    comparisons += 1
    if arr[mid] == key:
        return mid, comparisons
    elif arr[mid] > key:
        return binary_search_recursive(arr, low, mid - 1, key, comparisons)
    else:
        return binary_search_recursive(arr, mid + 1, high, key, comparisons)

```



```python


def binary_search_descending(arr, low, high, key, comparisons=0):
    if low > high:
        return -1, comparisons
    mid = (low + high) // 2
    comparisons += 1
    if arr[mid] == key:
        return mid, comparisons
    elif arr[mid] < key:
        return binary_search_descending(arr, low, mid - 1, key, comparisons)
    else:
        return binary_search_descending(arr, mid + 1, high, key, comparisons)



def compare_execution_times():
    sizes = [1000, 5000, 10000]
    print(f"{'Size':<10}{'Linear Time':<15}{'Binary Time':<15}")
    print("-" * 40)
    for n in sizes:
        arr = sorted(random.sample(range(1, n*10), n))  # unique random numbers
        key = random.choice(arr)

```

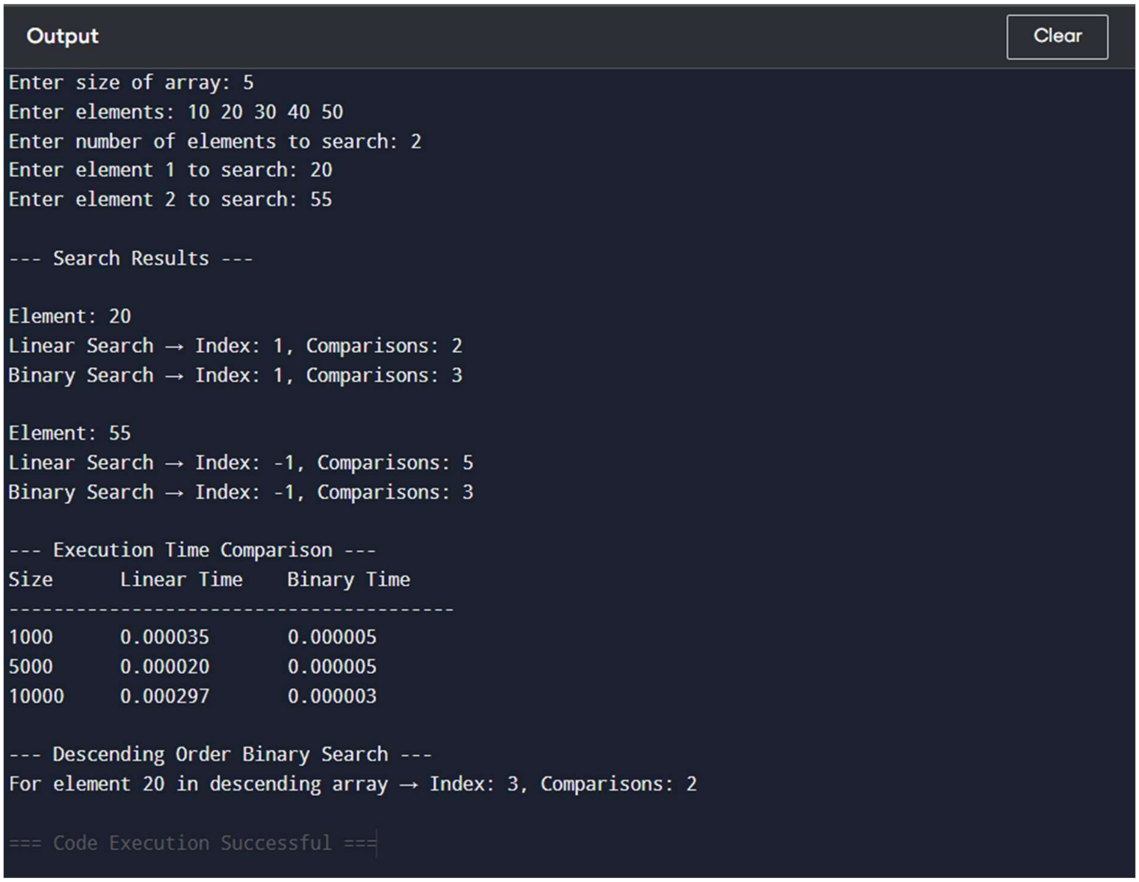


```

        start = time.time()
        linear_search(arr, key)
        linear_time = time.time() - start


        start = time.time()
        binary_search_recursive(arr, 0, len(arr)-1, key)
        binary_time = time.time() - start

        print(f"{n:<10}{linear_time:<15.6f}{binary_time:<15.6f}")



def main():
    n = int(input("Enter size of array: "))
    arr = list(map(int, input("Enter elements: ").split()))
    k = int(input("Enter number of elements to search: "))
    search_items = [int(input(f"Enter element {i+1} to search: ")) for i in range(k)]

    arr.sort()
    print("\n--- Search Results ---")
    for key in search_items:
```



```
        lin_idx, lin_cmp = linear_search(arr, key)
        bin_idx, bin_cmp = binary_search_recursive(arr, 0, len(arr)-1, key)
        print(f"\nElement: {key}")
        print(f"Linear Search → Index: {lin_idx}, Comparisons: {lin_cmp}")
        print(f"Binary Search → Index: {bin_idx}, Comparisons: {bin_cmp}")

    print("\n--- Execution Time Comparison ---")
    compare_execution_times()


    print("\n--- Descending Order Binary Search ---")
    arr_desc = sorted(arr, reverse=True)
    key = search_items[0]
    idx, cmp = binary_search_descending(arr_desc, 0, len(arr_desc)-1, key)
    print(f"For element {key} in descending array → Index: {idx}, Comparisons: {cmp}")


if __name__ == "__main__":
    main()

```

OUTPUT:



```
Enter size of array: 5
Enter elements: 10 20 30 40 50
Enter number of elements to search: 2
Enter element 1 to search: 20
Enter element 2 to search: 55

--- Search Results ---

Element: 20
Linear Search → Index: 1, Comparisons: 2
Binary Search → Index: 1, Comparisons: 3

Element: 55
Linear Search → Index: -1, Comparisons: 5
Binary Search → Index: -1, Comparisons: 3

--- Execution Time Comparison ---
Size      Linear Time    Binary Time
----------------------------------------
1000      0.000035       0.000005
5000      0.000020       0.000005
10000     0.000297       0.000003

--- Descending Order Binary Search ---
For element 20 in descending array → Index: 3, Comparisons: 2

=== Code Execution Successful ===
```

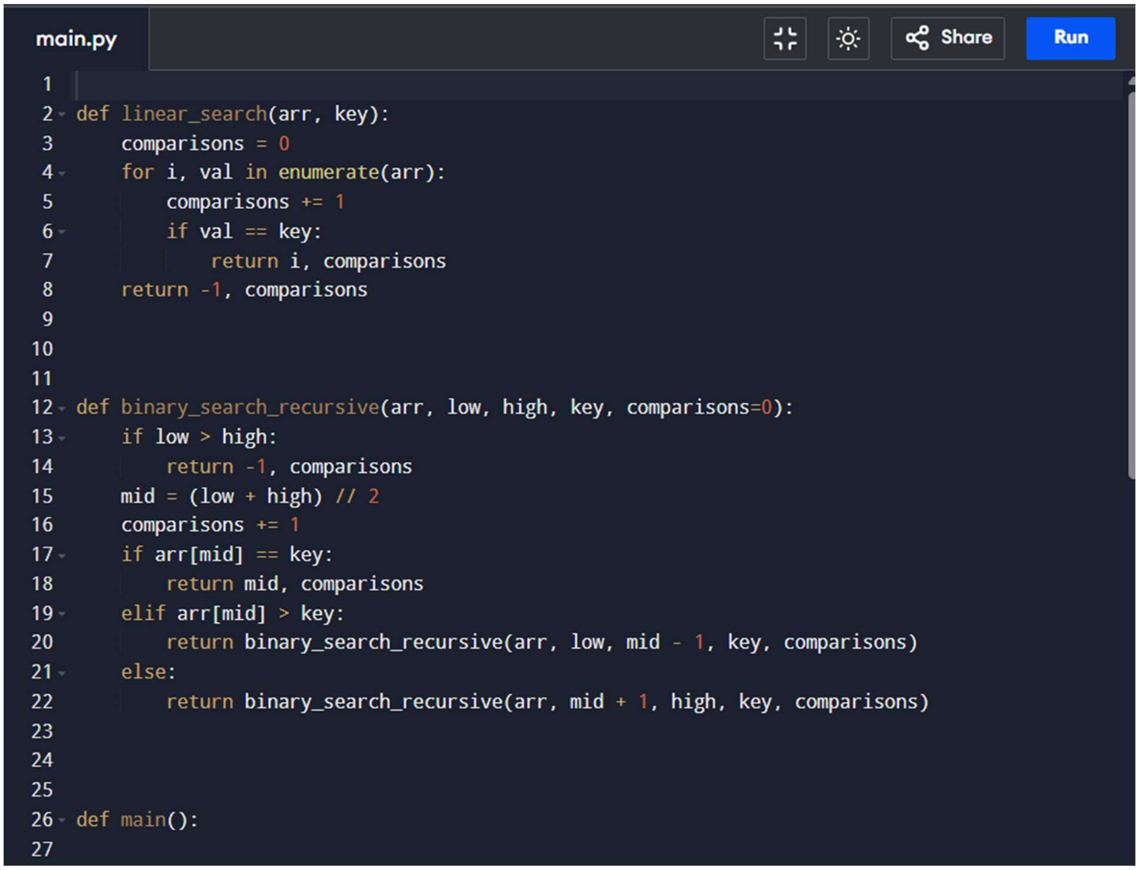
3: Extend the program to allow the user to enter k elements to search.

Algorithm

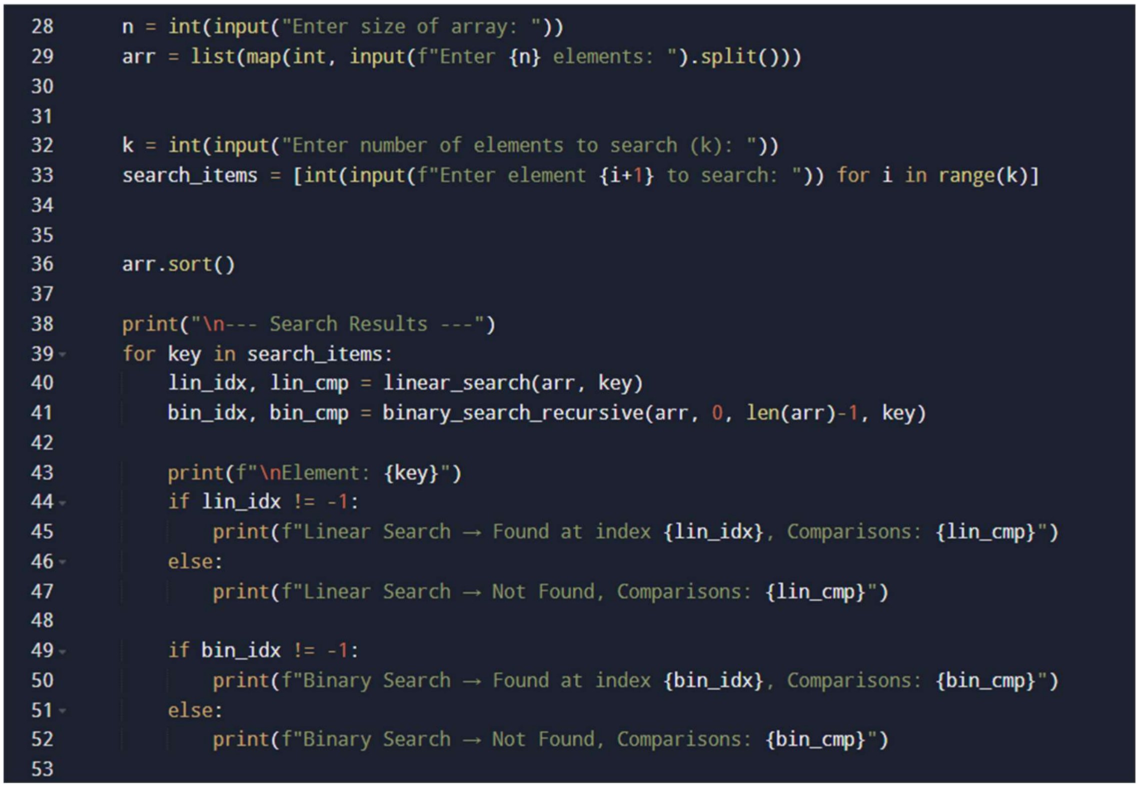
Algorithm: Linear Search and Binary Search for k elements

```
Step 1: Start the program.
Step 2: Input the size of the array n.
Step 3: Input n integers and store them in an array arr.
Step 4: Input the number of elements to search, k.
Step 5: Input the k search elements and store them in another list search_items.
Step 6: Sort the array before performing Binary Search.
Step 7: For each element in search_items:
    a. Perform Linear Search:
       - Compare the target with each element one by one.
       - Count each comparison.
       - If found, display the index and comparison count; otherwise display "Not Found".
    b. Perform Recursive Binary Search:
       - Set low = 0, high = n - 1.
       - Calculate mid = (low + high) // 2.
       - If arr[mid] == key, return index.
```

```
          - If arr[mid] > key, recursively search the left half.
          - Else, recursively search the right half.
          - Count and display comparisons.
Step 8: Repeat for all k search elements.
Step 9: Stop the program.
```



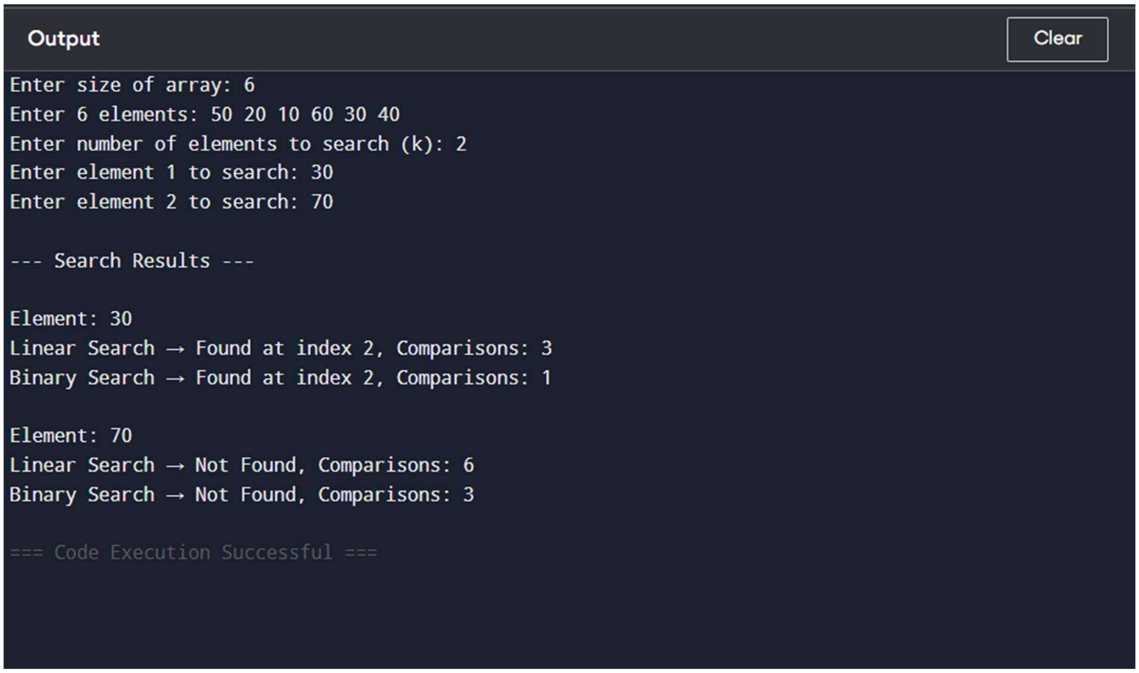
```python

def linear_search(arr, key):
    comparisons = 0
    for i, val in enumerate(arr):
        comparisons += 1
        if val == key:
            return i, comparisons
    return -1, comparisons



def binary_search_recursive(arr, low, high, key, comparisons=0):
    if low > high:
        return -1, comparisons
    mid = (low + high) // 2
    comparisons += 1
    if arr[mid] == key:
        return mid, comparisons
    elif arr[mid] > key:
        return binary_search_recursive(arr, low, mid - 1, key, comparisons)
    else:
        return binary_search_recursive(arr, mid + 1, high, key, comparisons)



def main():

```



```python
    n = int(input("Enter size of array: "))
    arr = list(map(int, input(f"Enter {n} elements: ").split()))


    k = int(input("Enter number of elements to search (k): "))
    search_items = [int(input(f"Enter element {i+1} to search: ")) for i in range(k)]


    arr.sort()

    print("\n--- Search Results ---")
    for key in search_items:
        lin_idx, lin_cmp = linear_search(arr, key)
        bin_idx, bin_cmp = binary_search_recursive(arr, 0, len(arr)-1, key)

        print(f"\nElement: {key}")
        if lin_idx != -1:
            print(f"Linear Search → Found at index {lin_idx}, Comparisons: {lin_cmp}")
        else:
            print(f"Linear Search → Not Found, Comparisons: {lin_cmp}")

        if bin_idx != -1:
            print(f"Binary Search → Found at index {bin_idx}, Comparisons: {bin_cmp}")
        else:
            print(f"Binary Search → Not Found, Comparisons: {bin_cmp}")

```

```python

if __name__ == "__main__":
    main()

```

OUTPUT:

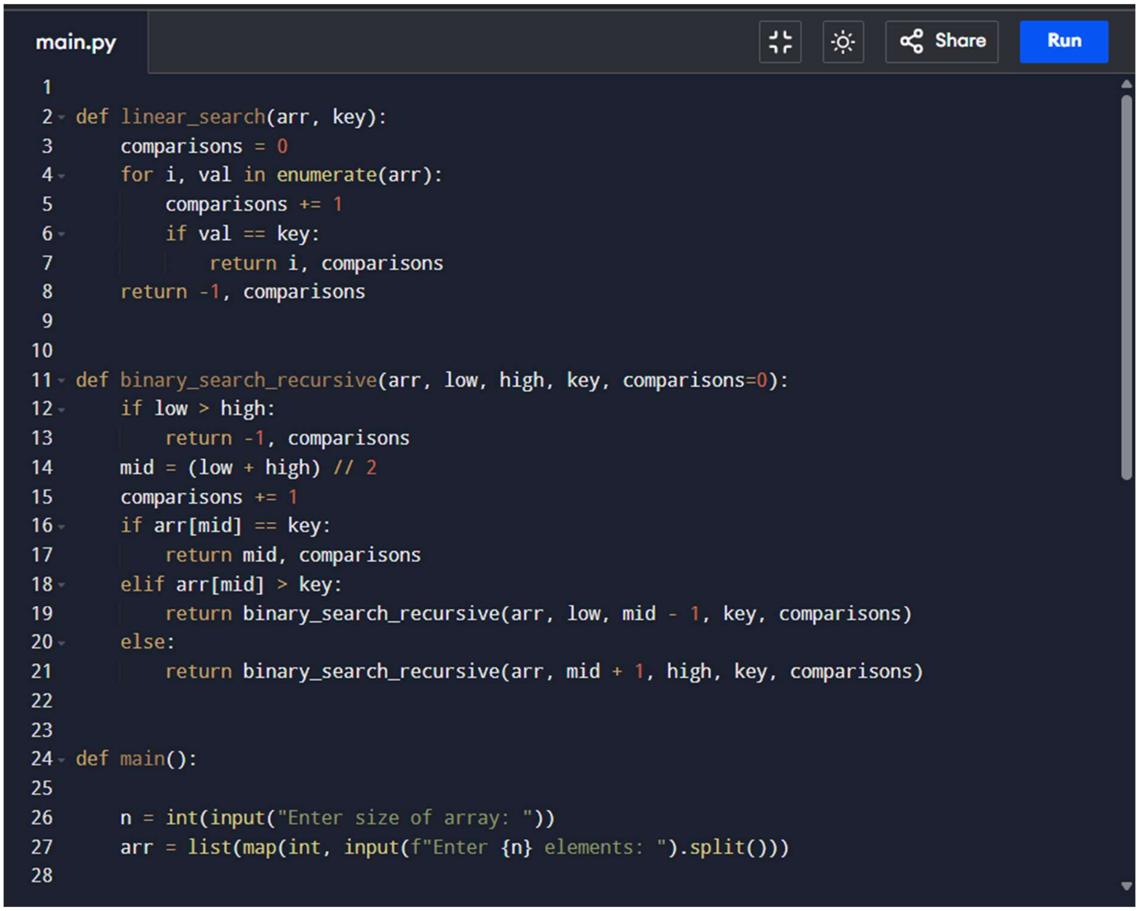


```
Enter size of array: 6
Enter 6 elements: 50 20 10 60 30 40
Enter number of elements to search (k): 2
Enter element 1 to search: 30
Enter element 2 to search: 70

--- Search Results ---

Element: 30
Linear Search → Found at index 2, Comparisons: 3
Binary Search → Found at index 2, Comparisons: 1

Element: 70
Linear Search → Not Found, Comparisons: 6
Binary Search → Not Found, Comparisons: 3

=== Code Execution Successful ===
```

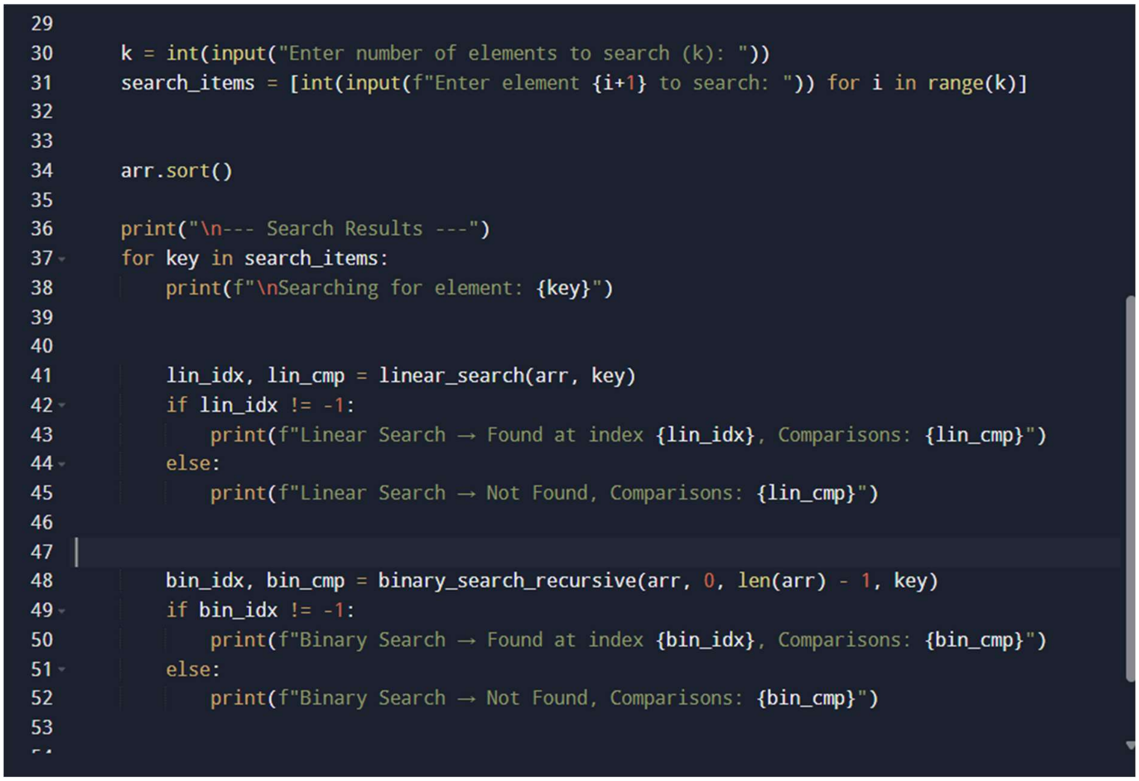
4: Perform Linear Search and Binary Search for each element.

Algorithm: Perform Linear and Binary Search for Each Element

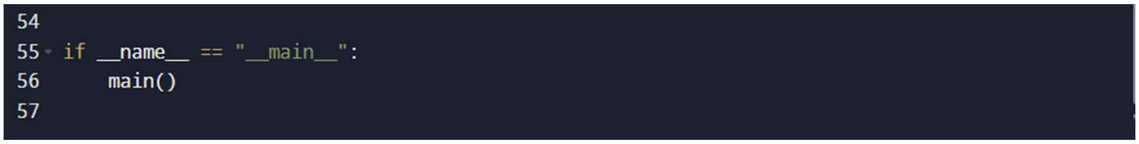
```
Step 1: Start the program.
Step 2: Input the size of the array n.
Step 3: Input n elements and store them in an array arr.
Step 4: Input the number of search elements k.
Step 5: Input the k search elements and store them in a list search_items.
Step 6: Sort the array before applying Binary Search.
Step 7: For each element key in search_items:
        a. Apply Linear Search on the array:
            • Compare key with each element.
            • Count comparisons.
            • If found, return index.
            • Otherwise, display “Not Found.”
        b. Apply Binary Search (Recursive) on the sorted array:
            • Find the middle element.
            • Compare key with the middle element.
            • If equal → return index.
            • If smaller → search left subarray.
            • If larger → search right subarray.
            • Count total comparisons.
        c. Display both search results for that element.
Step 8: Repeat for all k elements.
Step 9: Stop the program.
```



```python

def linear_search(arr, key):
    comparisons = 0
    for i, val in enumerate(arr):
        comparisons += 1
        if val == key:
            return i, comparisons
    return -1, comparisons


def binary_search_recursive(arr, low, high, key, comparisons=0):
    if low > high:
        return -1, comparisons
    mid = (low + high) // 2
    comparisons += 1
    if arr[mid] == key:
        return mid, comparisons
    elif arr[mid] > key:
        return binary_search_recursive(arr, low, mid - 1, key, comparisons)
    else:
        return binary_search_recursive(arr, mid + 1, high, key, comparisons)


def main():

    n = int(input("Enter size of array: "))
    arr = list(map(int, input(f"Enter {n} elements: ").split()))

```

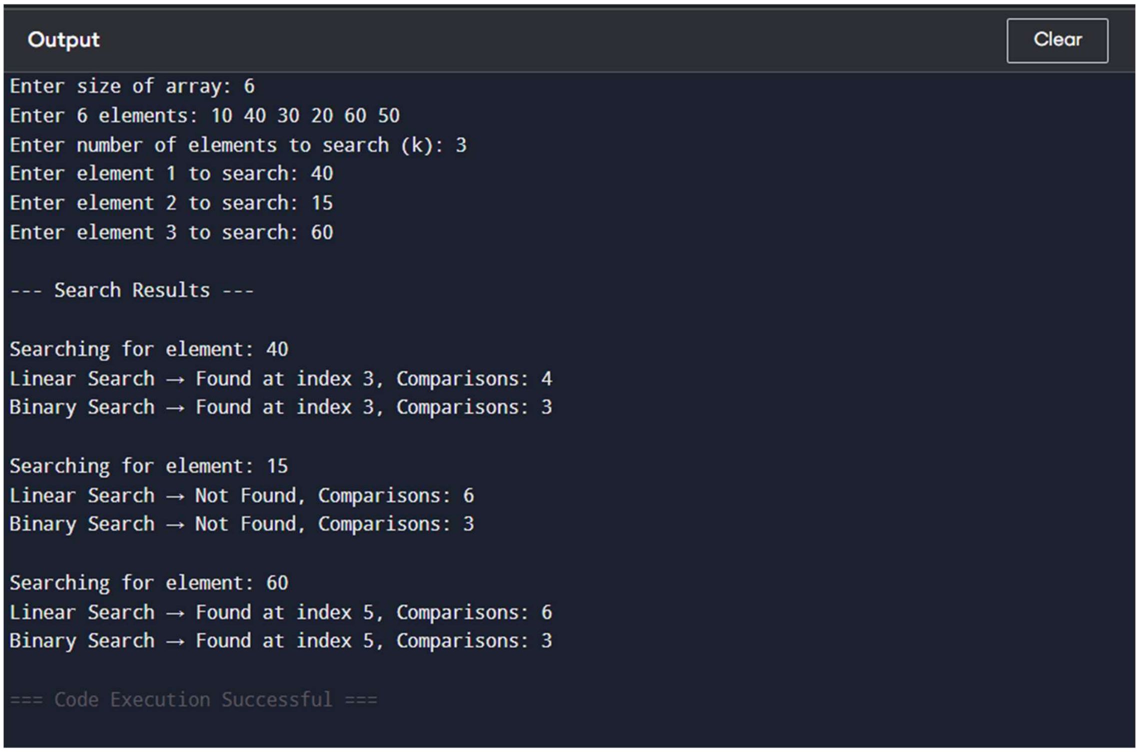


```python

    k = int(input("Enter number of elements to search (k): "))
    search_items = [int(input(f"Enter element {i+1} to search: ")) for i in range(k)]


    arr.sort()

    print("\n--- Search Results ---")
    for key in search_items:
        print(f"\nSearching for element: {key}")


        lin_idx, lin_cmp = linear_search(arr, key)
        if lin_idx != -1:
            print(f"Linear Search → Found at index {lin_idx}, Comparisons: {lin_cmp}")
        else:
            print(f"Linear Search → Not Found, Comparisons: {lin_cmp}")


        bin_idx, bin_cmp = binary_search_recursive(arr, 0, len(arr) - 1, key)
        if bin_idx != -1:
            print(f"Binary Search → Found at index {bin_idx}, Comparisons: {bin_cmp}")
        else:
            print(f"Binary Search → Not Found, Comparisons: {bin_cmp}")

```



```
if __name__ == "__main__":
    main()
```

OUTPUT:



```
Enter size of array: 6
Enter 6 elements: 10 40 30 20 60 50
Enter number of elements to search (k): 3
Enter element 1 to search: 40
Enter element 2 to search: 15
Enter element 3 to search: 60

--- Search Results ---

Searching for element: 40
Linear Search → Found at index 3, Comparisons: 4
Binary Search → Found at index 3, Comparisons: 3

Searching for element: 15
Linear Search → Not Found, Comparisons: 6
Binary Search → Not Found, Comparisons: 3

Searching for element: 60
Linear Search → Found at index 5, Comparisons: 6
Binary Search → Found at index 5, Comparisons: 3

=== Code Execution Successful ===
```

5: Execution Time Comparison

o For input sizes n = 1000, 5000, 10000:

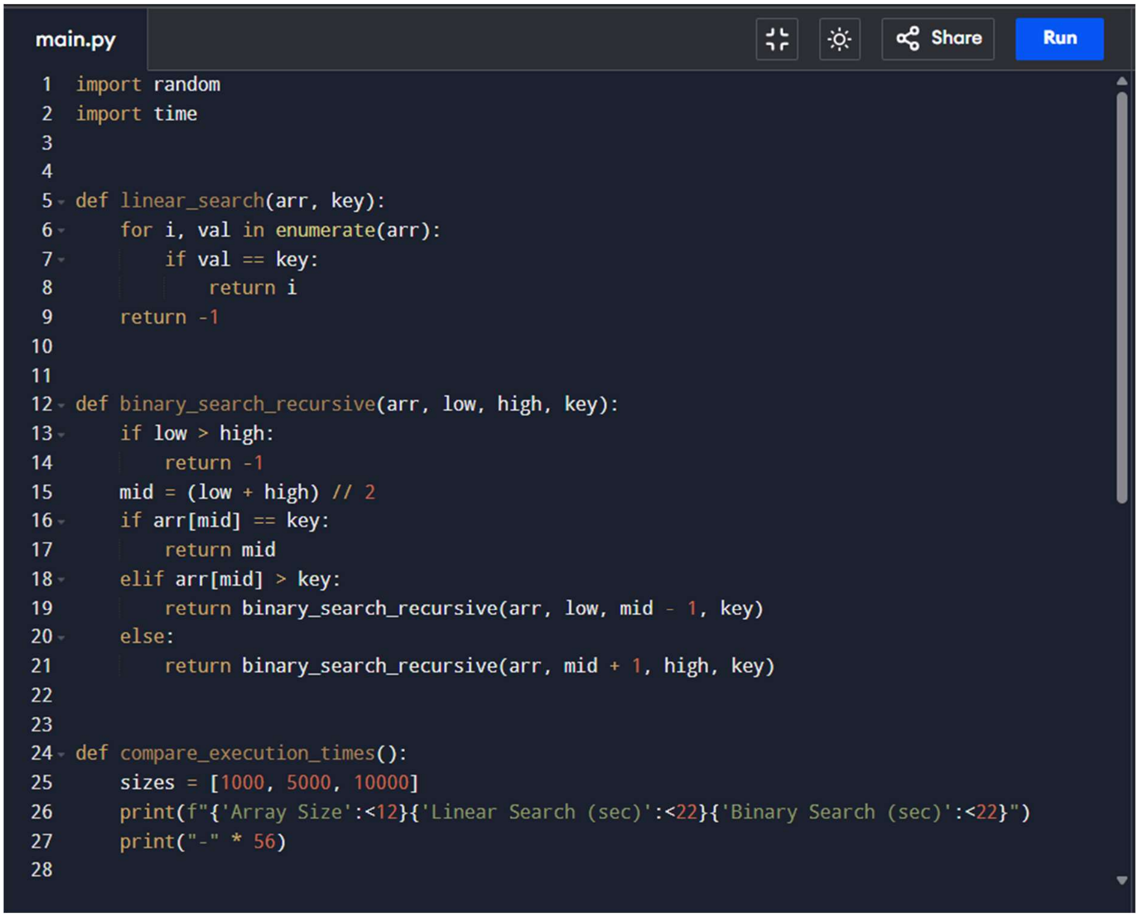
- **□ Generate arrays with random numbers.**
- **□ Run Linear and Binary Search.**
- **□ Measure execution time using &lt;ctime&gt; in C++ or time module in Python.**
- **□ Display results in a table.**

- ## Algorithm: Execution Time Comparison

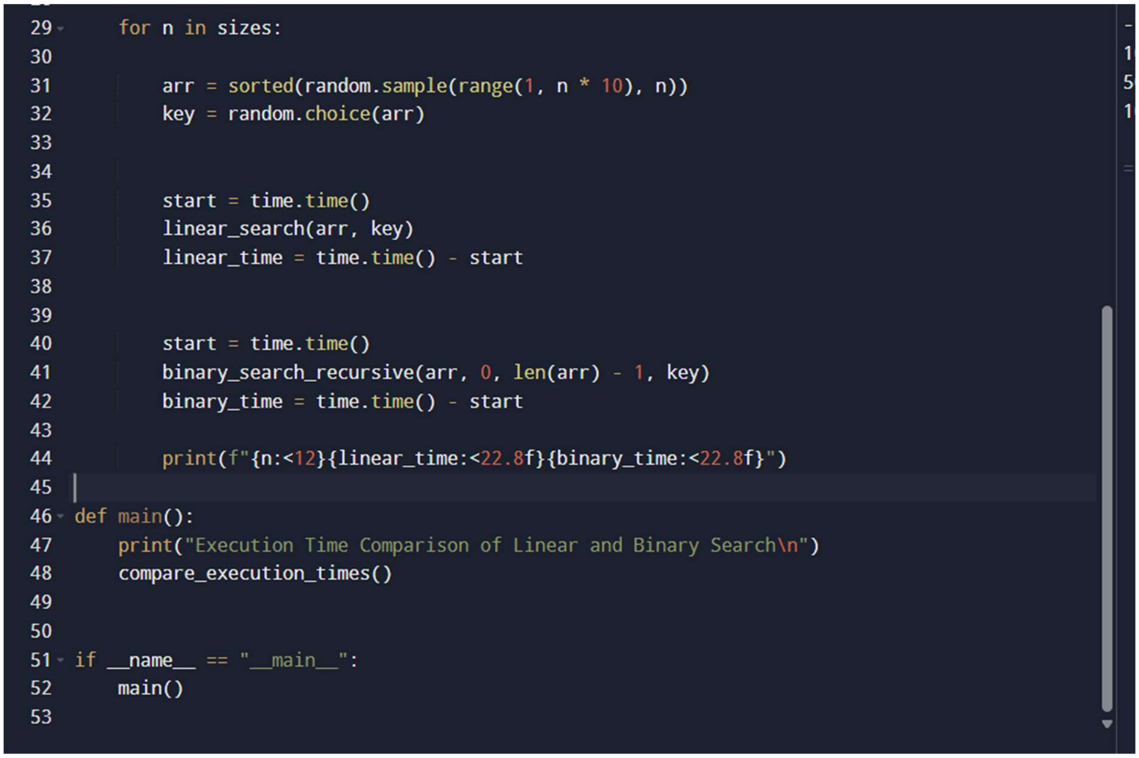
  Step 1: Start the program.
  Step 2: Define three array sizes → 1000, 5000, 10000.
  Step 3: For each size n:
      a. Generate a random array of n integers.
      b. Sort the array for Binary Search.
      c. Choose a random element from the array as the search key.

d. Record the start time.
e. Perform Linear Search and record the end time.
f. Calculate `linear_time = end - start`.
g. Record the start time again.
h. Perform Binary Search and record the end time.
i. Calculate `binary_time = end - start`.
j. Display both times in a formatted table.

Step 4: Stop the program.

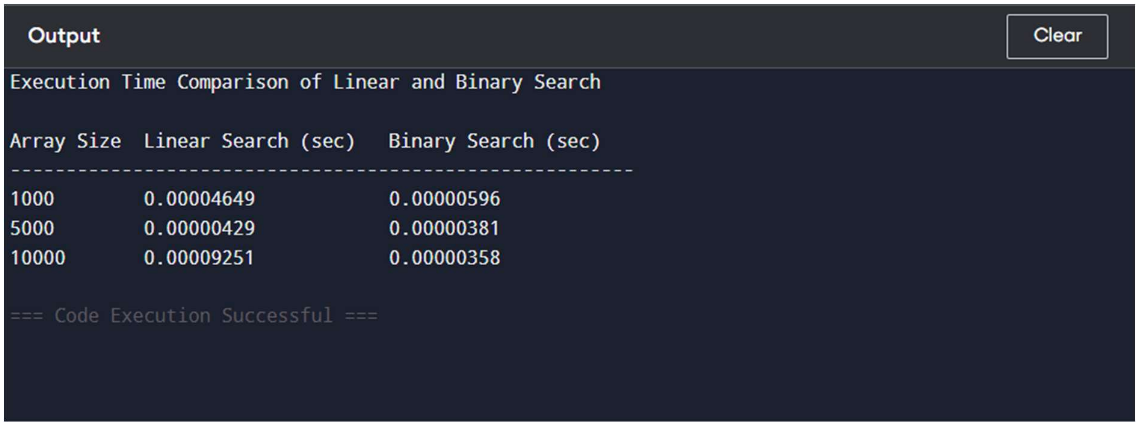


```python
import random
import time


def linear_search(arr, key):
    for i, val in enumerate(arr):
        if val == key:
            return i
    return -1


def binary_search_recursive(arr, low, high, key):
    if low > high:
        return -1
    mid = (low + high) // 2
    if arr[mid] == key:
        return mid
    elif arr[mid] > key:
        return binary_search_recursive(arr, low, mid - 1, key)
    else:
        return binary_search_recursive(arr, mid + 1, high, key)


def compare_execution_times():
    sizes = [1000, 5000, 10000]
    print(f"{'Array Size':<12}{'Linear Search (sec)':<22}{'Binary Search (sec)':<22}")
    print("-" * 56)

```



```
    for n in sizes:

        arr = sorted(random.sample(range(1, n * 10), n))
        key = random.choice(arr)


        start = time.time()
        linear_search(arr, key)
        linear_time = time.time() - start


        start = time.time()
        binary_search_recursive(arr, 0, len(arr) - 1, key)
        binary_time = time.time() - start

        print(f"{n:<12}{linear_time:<22.8f}{binary_time:<22.8f}")

def main():
    print("Execution Time Comparison of Linear and Binary Search\n")
    compare_execution_times()


if __name__ == "__main__":
    main()

```

OUTPUT:



```
Execution Time Comparison of Linear and Binary Search

Array Size  Linear Search (sec)   Binary Search (sec)
--------------------------------------------------------
1000        0.00004649            0.00000596
5000        0.00000429            0.00000381
10000       0.00009251            0.00000358

=== Code Execution Successful ===
```

6: Modify Binary Search algorithm to work when the array is sorted in descending order.

Algorithm: Binary Search for Descending Order Array

```
Step 1: Start the program.
Step 2: Input a sorted array in descending order (highest → lowest).
Step 3: Set low = 0, high = n - 1.
Step 4: Repeat until low <= high:
   a. Calculate mid = (low + high) // 2.
```

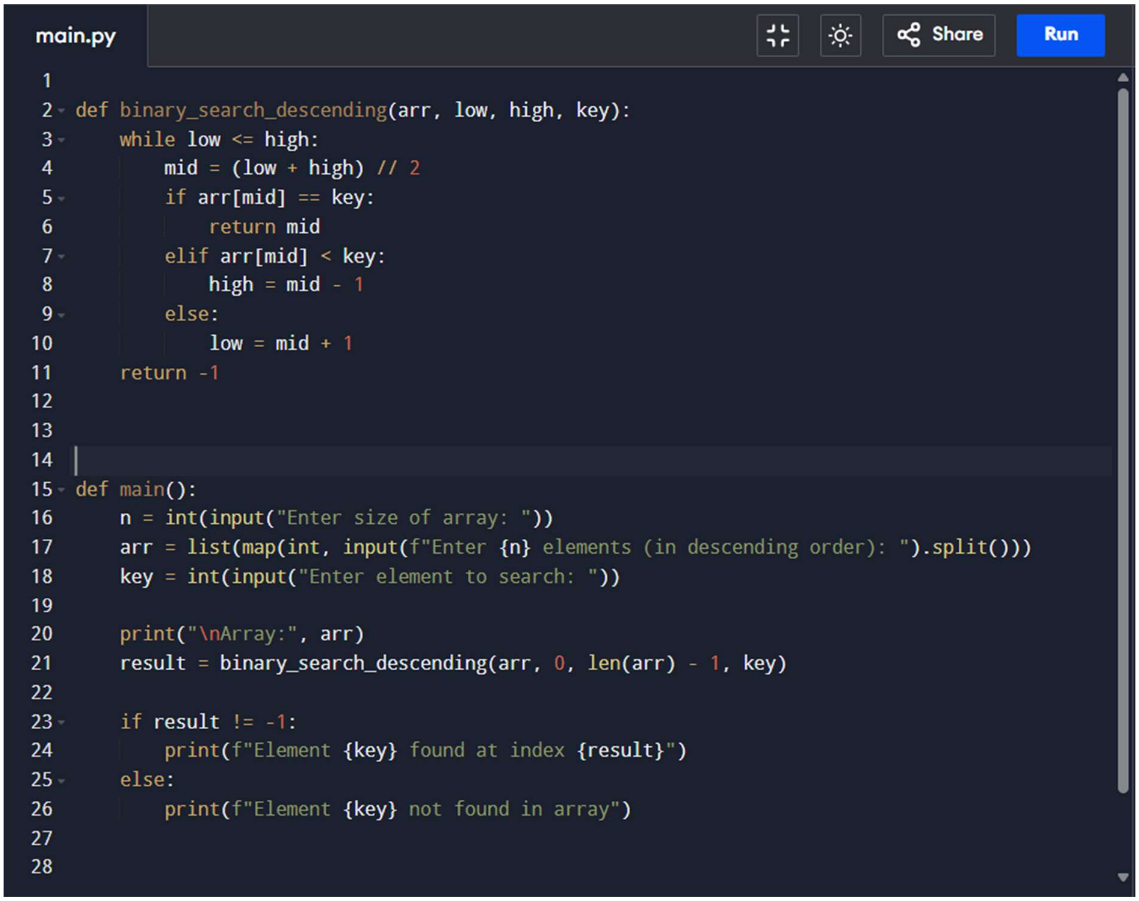
b. If arr[mid] == key, return index (found).
c. If arr[mid] < key, search left half (set high = mid - 1).
d. Else if arr[mid] > key, search right half (set low = mid + 1).

Step 5: If element is not found, return “Not Found”.

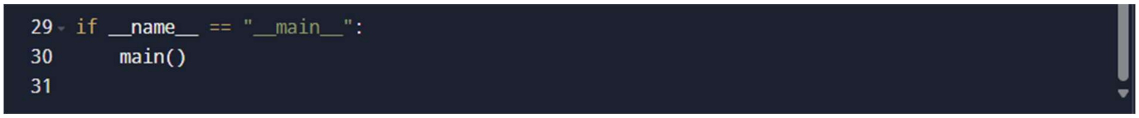
Step 6: Stop the program.

Note:

In descending order, comparison directions reverse — because smaller numbers are on the right, and larger ones are on the left.

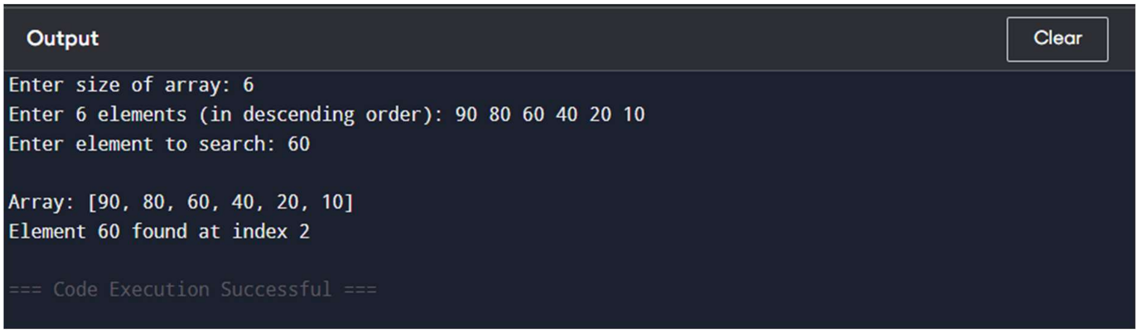


```python

def binary_search_descending(arr, low, high, key):
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            high = mid - 1
        else:
            low = mid + 1
    return -1



def main():
    n = int(input("Enter size of array: "))
    arr = list(map(int, input(f"Enter {n} elements (in descending order): ").split()))
    key = int(input("Enter element to search: "))

    print("\nArray:", arr)
    result = binary_search_descending(arr, 0, len(arr) - 1, key)

    if result != -1:
        print(f"Element {key} found at index {result}")
    else:
        print(f"Element {key} not found in array")


```



```python
if __name__ == "__main__":
    main()

```

OUTPUT:



```
Enter size of array: 6
Enter 6 elements (in descending order): 90 80 60 40 20 10
Enter element to search: 60

Array: [90, 80, 60, 40, 20, 10]
Element 60 found at index 2

=== Code Execution Successful ===
```

7: Buggy Code (Given in Lab)

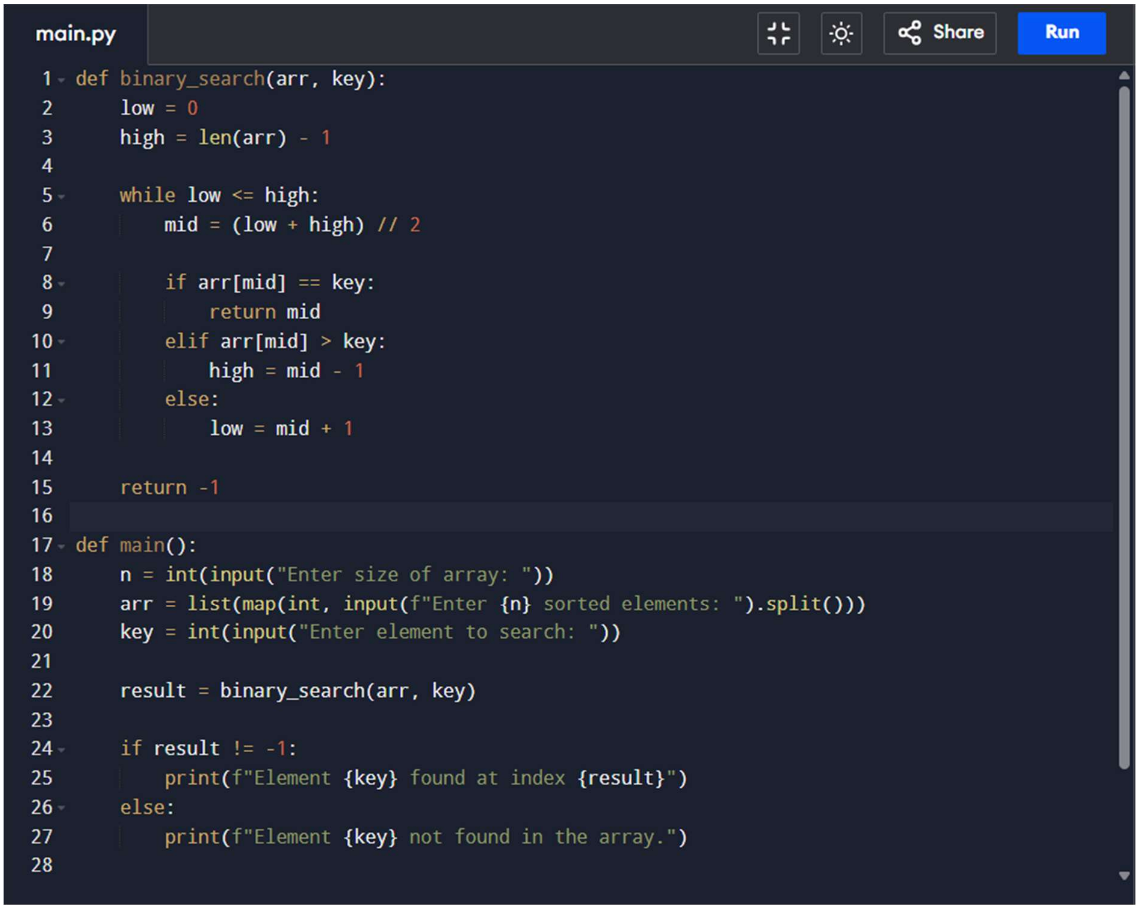
```
int binarySearch(int arr[], int n, int key) {
  int low = 0, high = n;
  while(low < high) {
    int mid = low + high / 2;
    if(arr[mid] = key)
      return mid;
    else if(arr[mid] > key)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return -1;
}
```

Algorithm (Debugged Binary Search):

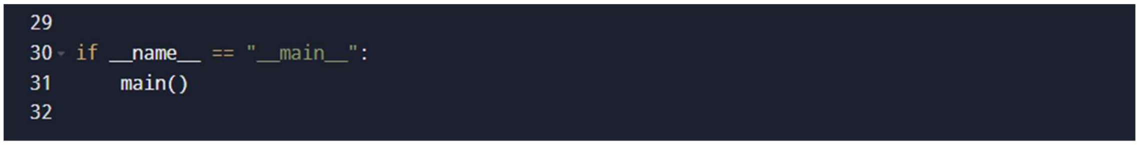
```
Step 1: Start the program.
Step 2: Input the array elements and the target value (key).
Step 3: Set low = 0, high = n - 1.
Step 4: While low <= high:
   a. Compute mid = (low + high) // 2.
   b. If arr[mid] == key, return the index.
   c. If arr[mid] > key, set high = mid - 1.
   d. Else, set low = mid + 1.
```

Step 5: If element not found, return -1.
Step 6: Stop.

Code in python fix version:

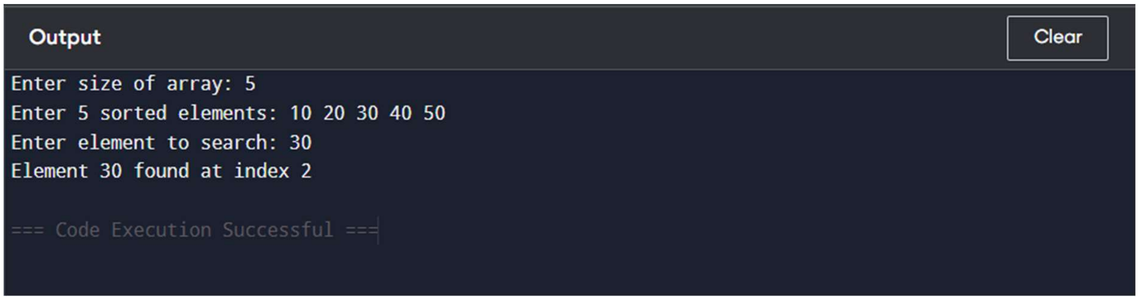


```python
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == key:
            return mid
        elif arr[mid] > key:
            high = mid - 1
        else:
            low = mid + 1

    return -1

def main():
    n = int(input("Enter size of array: "))
    arr = list(map(int, input(f"Enter {n} sorted elements: ").split()))
    key = int(input("Enter element to search: "))

    result = binary_search(arr, key)

    if result != -1:
        print(f"Element {key} found at index {result}")
    else:
        print(f"Element {key} not found in the array.")

```



```python

if __name__ == "__main__":
    main()

```

Output:



```
Enter size of array: 5
Enter 5 sorted elements: 10 20 30 40 50
Enter element to search: 30
Element 30 found at index 2

=== Code Execution Successful ===
```