



CECOS UNIVERSITY

DATA STRUCTURE AND ALGORITHMS

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")
```

```
main.py  [ ] [ ] [ ] Share Run
1
2- def binary_search_recursive(arr, low, high, key):
3-     if low > high:
4-         return -1
5-     mid = (low + high) // 2
6-
7-     if arr[mid] == key:
8-         return mid
9-     elif arr[mid] > key:
10-         return binary_search_recursive(arr, low, mid - 1, key)
11-     else:
12-         return binary_search_recursive(arr, mid + 1, high, key)
13-
14-
15-
16 n = int(input("Enter size of array: "))
17 arr = list(map(int, input("Enter elements: ").split()))
18 key = int(input("Enter element to search: "))
19
20
21 arr.sort()
22 print("Sorted array:", arr)
23
24 result = binary_search_recursive(arr, 0, n - 1, key)
25
26 if result != -1:
27     print(f"Binary Search (Recursive): Found at index {result}")
28 else:
29     print("Binary Search (Recursive): Not Found")
30
```

OUTPUT:

```
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.

main.py



Share

Run

```
1 import random
2 import time
3
4
5 def linear_search(arr, key):
6     comparisons = 0
7     for i, val in enumerate(arr):
8         comparisons += 1
9         if val == key:
10             return i, comparisons
11     return -1, comparisons
12
13
14
15 def binary_search_recursive(arr, low, high, key, comparisons=0):
16     if low > high:
17         return -1, comparisons
18     mid = (low + high) // 2
19     comparisons += 1
20     if arr[mid] == key:
21         return mid, comparisons
22     elif arr[mid] > key:
23         return binary_search_recursive(arr, low, mid - 1, key, comparisons)
24     else:
25         return binary_search_recursive(arr, mid + 1, high, key, comparisons)
26
27
28
29 def binary_search_descending(arr, low, high, key, comparisons=0):
30     if low > high:
31         return -1, comparisons
32     mid = (low + high) // 2
33     comparisons += 1
34     if arr[mid] == key:
35         return mid, comparisons
36     elif arr[mid] < key:
37         return binary_search_descending(arr, low, mid - 1, key, comparisons)
38     else:
39         return binary_search_descending(arr, mid + 1, high, key, comparisons)
40
41
42
43 def compare_execution_times():
44     sizes = [1000, 5000, 10000]
45     print(f"{'Size':<10}{'Linear Time':<15}{'Binary Time':<15}")
46     print("-" * 40)
47     for n in sizes:
48         arr = sorted(random.sample(range(1, n*10), n)) # unique random numbers
49         key = random.choice(arr)
50
```

```

51
52     start = time.time()
53     linear_search(arr, key)
54     linear_time = time.time() - start
55
56
57     start = time.time()
58     binary_search_recursive(arr, 0, len(arr)-1, key)
59     binary_time = time.time() - start
60
61     print(f"{n:<10}{linear_time:<15.6f}{binary_time:<15.6f}")
62
63
64
65 def main():
66     n = int(input("Enter size of array: "))
67     arr = list(map(int, input("Enter elements: ").split()))
68     k = int(input("Enter number of elements to search: "))
69     search_items = [int(input(f"Enter element {i+1} to search: ")) for i in range(k)]
70
71     arr.sort()
72     print("\n--- Search Results ---")
73     for key in search_items:

```

```

74         lin_idx, lin_cmp = linear_search(arr, key)
75         bin_idx, bin_cmp = binary_search_recursive(arr, 0, len(arr)-1, key)
76         print(f"\nElement: {key}")
77         print(f"Linear Search → Index: {lin_idx}, Comparisons: {lin_cmp}")
78         print(f"Binary Search → Index: {bin_idx}, Comparisons: {bin_cmp}")
79
80     print("\n--- Execution Time Comparison ---")
81     compare_execution_times()
82
83
84     print("\n--- Descending Order Binary Search ---")
85     arr_desc = sorted(arr, reverse=True)
86     key = search_items[0]
87     idx, cmp = binary_search_descending(arr_desc, 0, len(arr_desc)-1, key)
88     print(f"For element {key} in descending array → Index: {idx}, Comparisons: {cmp}")
89
90
91 if __name__ == "__main__":
92     main()
93

```

OUTPUT:

```
Output Clear
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 $\text{arr}[\text{mid}] > \text{key}$, 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.

main.py



Share

Run

```
1
2 def linear_search(arr, key):
3     comparisons = 0
4     for i, val in enumerate(arr):
5         comparisons += 1
6         if val == key:
7             return i, comparisons
8     return -1, comparisons
9
10
11
12 def binary_search_recursive(arr, low, high, key, comparisons=0):
13     if low > high:
14         return -1, comparisons
15     mid = (low + high) // 2
16     comparisons += 1
17     if arr[mid] == key:
18         return mid, comparisons
19     elif arr[mid] > key:
20         return binary_search_recursive(arr, low, mid - 1, key, comparisons)
21     else:
22         return binary_search_recursive(arr, mid + 1, high, key, comparisons)
23
24
25
26 def main():
27
28     n = int(input("Enter size of array: "))
29     arr = list(map(int, input(f"Enter {n} elements: ").split()))
30
31
32     k = int(input("Enter number of elements to search (k): "))
33     search_items = [int(input(f"Enter element {i+1} to search: ")) for i in range(k)]
34
35
36     arr.sort()
37
38     print("\n--- Search Results ---")
39     for key in search_items:
40         lin_idx, lin_cmp = linear_search(arr, key)
41         bin_idx, bin_cmp = binary_search_recursive(arr, 0, len(arr)-1, key)
42
43         print(f"\nElement: {key}")
44         if lin_idx != -1:
45             print(f"Linear Search → Found at index {lin_idx}, Comparisons: {lin_cmp}")
46         else:
47             print(f"Linear Search → Not Found, Comparisons: {lin_cmp}")
48
49         if bin_idx != -1:
50             print(f"Binary Search → Found at index {bin_idx}, Comparisons: {bin_cmp}")
51         else:
52             print(f"Binary Search → Not Found, Comparisons: {bin_cmp}")
53
```

```
54
55 ~ if __name__ == "__main__":
56     main()
57
```

OUTPUT:

Output Clear

```
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 \rightarrow return index.
 - If smaller \rightarrow search left subarray.
 - If larger \rightarrow 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.

main.py



Share

Run

```
1
2 def linear_search(arr, key):
3     comparisons = 0
4     for i, val in enumerate(arr):
5         comparisons += 1
6         if val == key:
7             return i, comparisons
8     return -1, comparisons
9
10
11 def binary_search_recursive(arr, low, high, key, comparisons=0):
12     if low > high:
13         return -1, comparisons
14     mid = (low + high) // 2
15     comparisons += 1
16     if arr[mid] == key:
17         return mid, comparisons
18     elif arr[mid] > key:
19         return binary_search_recursive(arr, low, mid - 1, key, comparisons)
20     else:
21         return binary_search_recursive(arr, mid + 1, high, key, comparisons)
22
23
24 def main():
25
26     n = int(input("Enter size of array: "))
27     arr = list(map(int, input(f"Enter {n} elements: ").split()))
28
29
30     k = int(input("Enter number of elements to search (k): "))
31     search_items = [int(input(f"Enter element {i+1} to search: ")) for i in range(k)]
32
33
34     arr.sort()
35
36     print("\n--- Search Results ---")
37     for key in search_items:
38         print(f"\nSearching for element: {key}")
39
40
41         lin_idx, lin_cmp = linear_search(arr, key)
42         if lin_idx != -1:
43             print(f"Linear Search → Found at index {lin_idx}, Comparisons: {lin_cmp}")
44         else:
45             print(f"Linear Search → Not Found, Comparisons: {lin_cmp}")
46
47
48         bin_idx, bin_cmp = binary_search_recursive(arr, 0, len(arr) - 1, key)
49         if bin_idx != -1:
50             print(f"Binary Search → Found at index {bin_idx}, Comparisons: {bin_cmp}")
51         else:
52             print(f"Binary Search → Not Found, Comparisons: {bin_cmp}")
53
54
```

```

54
55 if __name__ == "__main__":
56     main()
57

```

OUTPUT:

Output

Clear

```

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 `<time>` 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 $\rightarrow 1000, 5000, 10000$.

Step 3: For each size n :

- Generate a random array of n integers.
- Sort the array for Binary Search.
- 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.

```
main.py  [Icons] [Share] [Run]
1 import random
2 import time
3
4
5 def linear_search(arr, key):
6     for i, val in enumerate(arr):
7         if val == key:
8             return i
9     return -1
10
11
12 def binary_search_recursive(arr, low, high, key):
13     if low > high:
14         return -1
15     mid = (low + high) // 2
16     if arr[mid] == key:
17         return mid
18     elif arr[mid] > key:
19         return binary_search_recursive(arr, low, mid - 1, key)
20     else:
21         return binary_search_recursive(arr, mid + 1, high, key)
22
23
24 def compare_execution_times():
25     sizes = [1000, 5000, 10000]
26     print(f'{"Array Size":<12}{"Linear Search (sec)":<22}{"Binary Search (sec)":<22}')
27     print("-" * 56)
28
```

```

29-     for n in sizes:
30-
31-         arr = sorted(random.sample(range(1, n * 10), n))
32-         key = random.choice(arr)
33-
34-
35-         start = time.time()
36-         linear_search(arr, key)
37-         linear_time = time.time() - start
38-
39-
40-         start = time.time()
41-         binary_search_recursive(arr, 0, len(arr) - 1, key)
42-         binary_time = time.time() - start
43-
44-         print(f"{n:<12}{linear_time:<22.8f}{binary_time:<22.8f}")
45-
46- def main():
47-     print("Execution Time Comparison of Linear and Binary Search\n")
48-     compare_execution_times()
49-
50-
51- if __name__ == "__main__":
52-     main()
53-

```

OUTPUT:

Output

Clear

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.

```
main.py  [Icons] [Share] [Run]
1
2- def binary_search_descending(arr, low, high, key):
3-     while low <= high:
4-         mid = (low + high) // 2
5-         if arr[mid] == key:
6-             return mid
7-         elif arr[mid] < key:
8-             high = mid - 1
9-         else:
10-            low = mid + 1
11-     return -1
12
13
14 |
15- def main():
16-     n = int(input("Enter size of array: "))
17-     arr = list(map(int, input(f"Enter {n} elements (in descending order): ").split()))
18-     key = int(input("Enter element to search: "))
19
20-     print("\nArray:", arr)
21-     result = binary_search_descending(arr, 0, len(arr) - 1, key)
22
23-     if result != -1:
24-         print(f"Element {key} found at index {result}")
25-     else:
26-         print(f"Element {key} not found in array")
27
28
29- if __name__ == "__main__":
30-     main()
31
```


OUTPUT:

```
Output Clear
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:

```
main.py  [Icons] [Share] [Run]

1 def binary_search(arr, key):
2     low = 0
3     high = len(arr) - 1
4
5     while low <= high:
6         mid = (low + high) // 2
7
8         if arr[mid] == key:
9             return mid
10        elif arr[mid] > key:
11            high = mid - 1
12        else:
13            low = mid + 1
14
15    return -1
16
17 def main():
18     n = int(input("Enter size of array: "))
19     arr = list(map(int, input(f"Enter {n} sorted elements: ").split()))
20     key = int(input("Enter element to search: "))
21
22     result = binary_search(arr, key)
23
24     if result != -1:
25         print(f"Element {key} found at index {result}")
26     else:
27         print(f"Element {key} not found in the array.")
28
29
30 if __name__ == "__main__":
31     main()
32
```

Output:

```
Output [Clear]

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 ===
```

