Bilal Ahmed Khalil **44926**

Bs-Cy (5-1)

Assignment: 4

Instructor: Muhammad Usman Sharif

Task:

Consider two algorithms, Algorithm A and Algorithm B, designed to solve the same problem. Your task is to design a hybrid algorithm that combines the best aspects of both Algorithm A and Algorithm B to achieve improved performance.

Searching for an element in a sorted array.

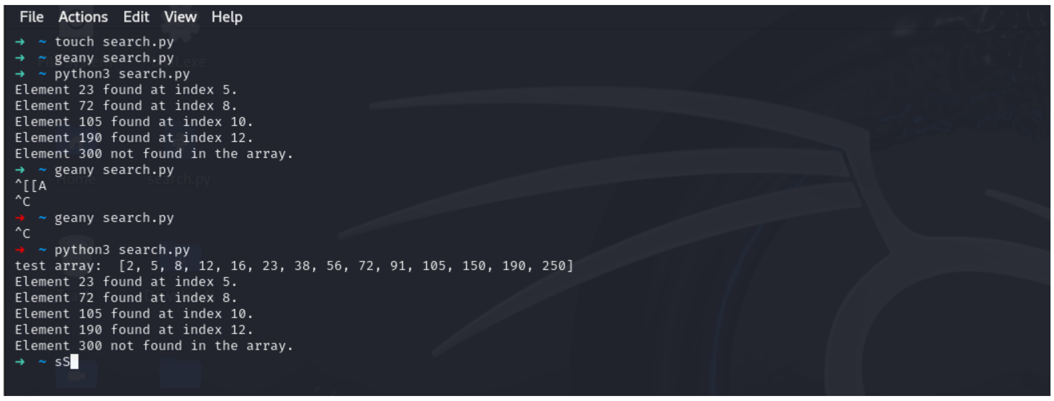
Algorithms Combined:

- Algorithm A: Binary Search
- Algorithm B: Jump Search

Code:

```
import math


def hybrid_search(arr, target):
    n = len(arr)


    # Step 1: Jump Search Phase
    jump = int(math.sqrt(n))  # Determine the optimal jump size (sqrt of array length)
    prev = 0  # Starting index of the current block


    # Perform jumps to find the block where the target may be present
    while prev < n and arr[min(jump, n) - 1] < target:
        prev = jump
        jump += int(math.sqrt(n))
        if prev >= n:
            return -1  # Target not found, we are past the array bounds

    # Step 2: Binary Search Phase within the identified block
    left = prev
    right = min(jump, n) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid  # Target found
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1  # Target not found

# Example Array to Test the Hybrid Search Algorithm
test_array = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91, 105, 150, 190, 250]

# Testing the Hybrid Search with different targets
targets = [23, 72, 105, 190, 300]  # The last target (300) is not in the array

for target in targets:
    result = hybrid_search(test_array, target)
    if result != -1:
        print(f"Element {target} found at index {result}.")
    else:
        print(f"Element {target} not found in the array.")
```

POC:



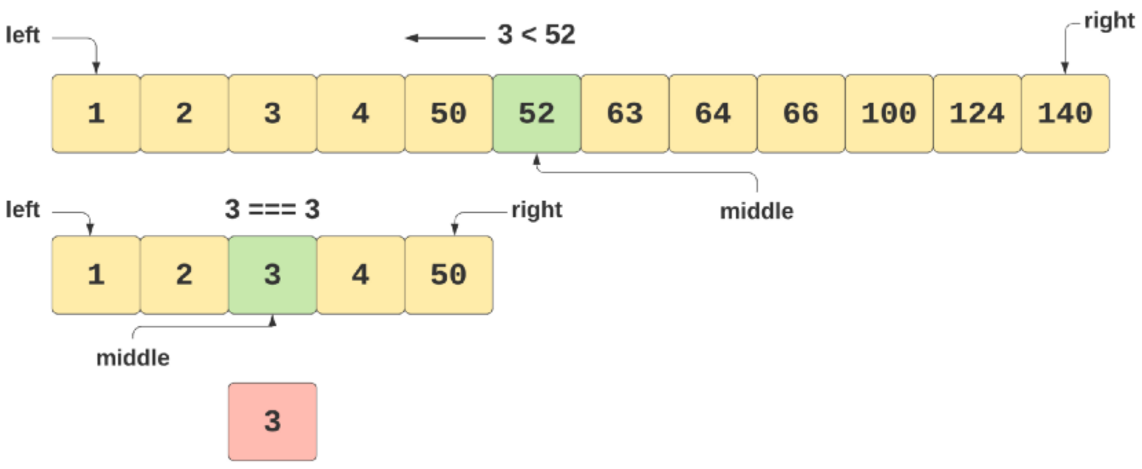
```
➜  ~ touch search.py
➜  ~ geany search.py
➜  ~ python3 search.py
Element 23 found at index 5.
Element 72 found at index 8.
Element 105 found at index 10.
Element 190 found at index 12.
Element 300 not found in the array.
➜  ~ geany search.py
^[[A
^C
➜  ~ geany search.py
^C
➜  ~ python3 search.py
test array:  [2, 5, 8, 12, 16, 23, 38, 56, 72, 91, 105, 150, 190, 250]
Element 23 found at index 5.
Element 72 found at index 8.
Element 105 found at index 10.
Element 190 found at index 12.
Element 300 not found in the array.
➜  ~ sS
```

Working:

To understand hybrid algo working we should first understand working of Binay and Jump search.

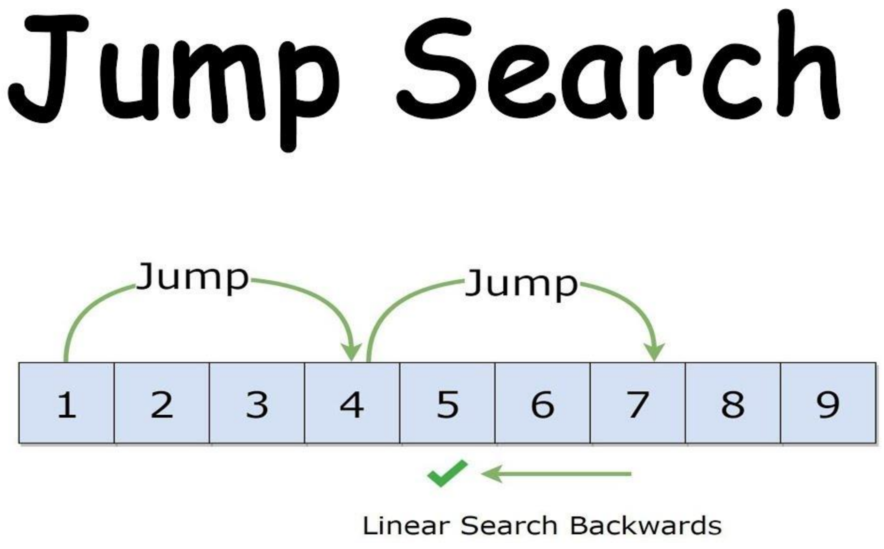
Binary Search

- **How it works**:
  - Binary Search is an efficient algorithm for finding an element in a sorted array. It works by repeatedly dividing the search interval in half.
  - Start with the middle element of the array:
    - If it matches the target, return the index.
    - If the target is smaller, repeat the process on the left subarray.
    - If the target is larger, repeat the process on the right subarray.
- **Time Complexity**: O (log n)
- **Space Complexity**: O (1)



Jump Search

- **How it works**:
    - Jump Search is designed for sorted arrays, where you "jump" ahead by a fixed number of elements (typically the square root of the array size, n\sqrt{n}n) instead of checking each element sequentially.
    - Once a jump goes past the target element, a linear search is performed in the previous block.
- **Time Complexity**: O(√n)
- **Space Complexity**: O(1)
- **When it's useful**: Jump Search is beneficial for searching in large datasets stored on media where sequential access is faster than random access (like tapes or SSDs).

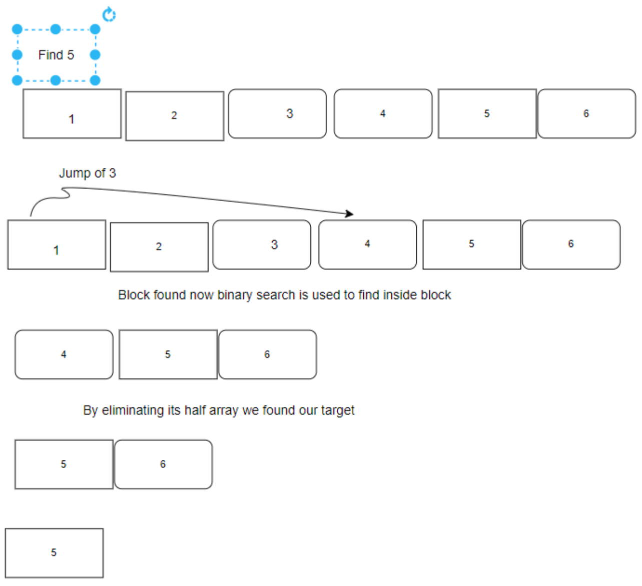


Need of algo:

- **Binary Search** is very efficient with O(log n) complexity but can be suboptimal for very large datasets stored in slower, non-RAM storage, where repeated midpoint accesses may introduce latency.
- **Jump Search** can quickly skip through large portions of the dataset but is inefficient when the target is located near the middle or end of the array due to its O(√n) complexity.
- **Hybrid Search** combines the fast-skipping capability of Jump Search with the precision of Binary Search to optimize the search process.

Hybrid algo Working:

1. **Jump Search Phase**:
    - Use Jump Search to quickly narrow down the search interval.
    - The jump size is chosen as n\sqrt{n}n because it balances the cost of jumping and the cost of the subsequent linear search.
    - This phase identifies the block (of size n\sqrt{n}n) where the target element might be located.
2. **Binary Search Phase**:
    - Once the correct block is identified, switch to Binary Search within that smaller block.
    - This phase ensures fast convergence to the target element within the reduced search space.


Find 5
1
2
3
4
5
6
Jump of 3
1
2
3
4
5
6
Block found now binary search is used to find inside block
4
5
6
By eliminating its half array we found our target
5
6
5

Performance Analysis

- **Time Complexity**:
    - Jump Phase: O(√n)
    - Binary Search Phase: O(log√n)
    - **Overall**: O(√n + log √n) which simplifies to O(√n)
- **Space Complexity**: O(1)Osince the algorithm uses a fixed amount of extra space.

Improved things:

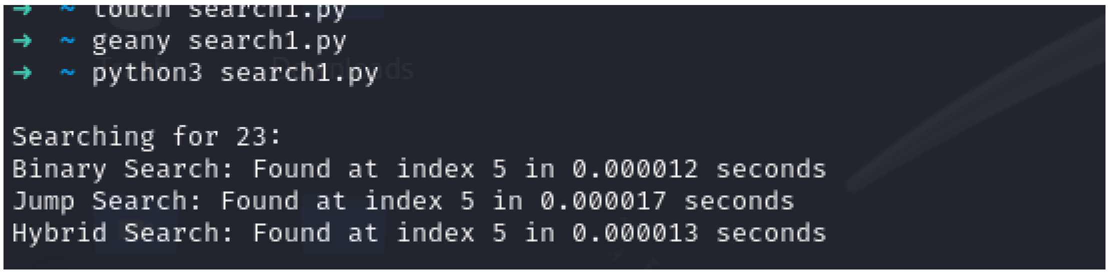
- For large datasets, the initial Jump Search quickly reduces the search space, allowing Binary Search to efficiently finish within a smaller segment.
- This can be particularly useful for data stored in slower, sequential-access storage systems where Binary Search alone might incur a high overhead due to non-sequential access.

  Searching time code:

```
import math

import time


# Binary Search Implementation

def binary_search(arr, target):

    left, right = 0, len(arr) - 1

    while left <= right:

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

        if arr[mid] == target:

            return mid

        elif arr[mid] < target:

            left = mid + 1

        else:

            right = mid - 1

    return -1


# Jump Search Implementation

def jump_search(arr, target):

    n = len(arr)

    jump = int(math.sqrt(n))

    prev = 0


    # Finding the block where the element may be present

    while arr[min(jump, n) - 1] < target:

        prev = jump
        jump += int(math.sqrt(n))
        if prev >= n:
            return -1

    # Linear search within the identified block
    for i in range(prev, min(jump, n)):
        if arr[i] == target:
            return i
    return -1

# Hybrid Search Implementation
def hybrid_search(arr, target):
    n = len(arr)
    jump = int(math.sqrt(n))
    prev = 0

    # Step 1: Jump Search Phase
    while prev < n and arr[min(jump, n) - 1] < target:
        prev = jump
        jump += int(math.sqrt(n))
        if prev >= n:
            return -1

    # Step 2: Binary Search Phase within the identified block
    left = prev
    right = min(jump, n) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

# Function to measure the execution time of different search algorithms
def measure_search_time(search_func, arr, target):
    start_time = time.time()
    result = search_func(arr, target)
    end_time = time.time()
    elapsed_time = end_time - start_time
    return result, elapsed_time

# Example Array
test_array = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91, 105, 150, 190, 250, 300, 350, 400, 450,
500]
targets = [23]

# Testing and Timing the Searches
for target in targets:
    print(f"\nSearching for {target}:")

    # Measure Binary Search Time
    result, time_taken = measure_search_time(binary_search, test_array, target)
    if result != -1:
        print(f"Binary Search: Found at index {result} in {time_taken:.6f} seconds")
    else:
        print(f"Binary Search: Not found in {time_taken:.6f} seconds")
```

```
# Measure Jump Search Time
result, time_taken = measure_search_time(jump_search, test_array, target)
if result != -1:
  print(f"Jump Search: Found at index {result} in {time_taken:.6f} seconds")
else:
  print(f"Jump Search: Not found in {time_taken:.6f} seconds")


# Measure Hybrid Search Time
result, time_taken = measure_search_time(hybrid_search, test_array, target)
if result != -1:
  print(f"Hybrid Search: Found at index {result} in {time_taken:.6f} seconds")
else:
  print(f"Hybrid Search: Not found in {time_taken:.6f} seconds")
```

POC:



```
➜  ~ touch search1.py
➜  ~ geany search1.py
➜  ~ python3 search1.py

Searching for 23:
Binary Search: Found at index 5 in 0.000012 seconds
Jump Search: Found at index 5 in 0.000017 seconds
Hybrid Search: Found at index 5 in 0.000013 seconds
```

Github: https://github.com/44926/Hybrid-algo-using-jump-and-binary-search.git