Question

1. What is Binary Search.

2. What are the advantages and disadvantages of Binary Search.

3. Write the Algorithm of Binary Search.

4. Write a simple C program for Binary Search.

Answers

1. Binary search is a search algorithm that works by repeatedly dividing the search space in half. The algorithm starts by comparing the target value to the middle element of the search space. If the target value is equal to the middle element, then the algorithm returns the index of the middle element. If the target value is less than the middle element, then the algorithm recursively searches the left half of the search space. Otherwise, the algorithm recursively searches the right half of the search space.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Efficiency: Binary Search has a time complexity of O(log n), which makes it highly efficient for searching large datasets. | * Sorted Data Requirement: Binary Search requires the data to be sorted before the search can be performed. If the data is unsorted or frequently changing, sorting overhead may outweigh the benefits of Binary Search. |
| * Versatility: It can be applied to a wide range of data structures, including sorted arrays, linked lists, and binary search trees. | * Limited Applicability: Binary Search is not suitable for unsorted or dynamically changing data. Maintaining a sorted order and updating it whenever changes occur can be costly. |
| * Effective for Sorted Data: Binary Search works best on sorted data, making it suitable for scenarios where the data remains sorted or can be easily sorted. | * Memory Overhead: Binary Search does not lend itself well to scenarios where memory utilization is a concern, as it requires random access to elements. |
| * Reduced Search Space: With each comparison, the search space is halved, leading to faster search times compared to linear search algorithms. |  |



def binary\_search(array, target):

low = 0

high = len(array) - 1

while low <= high:

mid = (low + high) // 2

if array[mid] == target:

return mid

elif array[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1



#include <stdio.h>

int binarySearch(int arr[], int n, int target)

{

int start = 0;

int end = n - 1;

while (start <= end)

{

int mid = start + (end - start) / 2;

if (arr[mid] == target)

return mid;

if (arr[mid] < target)

start = mid + 1;

else

end = mid - 1;

}

return -1; // Target not found

}

int main()

{

int arr[] = {2, 4, 6, 8, 10, 12, 14};

int n = sizeof(arr) / sizeof(arr[0]);

int target = 10;

int result = binarySearch(arr, n, target);

if (result == -1)

printf("Target element not found\n");

else

printf("Target element found at index %d\n", result);

return 0;

}