Tutorial 02

1.Using an appropriate example, explain how main searching algorithms can be performed.

2.Compare (look for similarities) and contrast (look for differences) linear search and algorithms.

3.Write a function using pseudo or source codes for searching an integer variable called an item using linear search in an array called an unordered array.

4.Write the C program for Binary search.

5.Briefly explain bubble sort, selection sort, and insertion sort.

Answer

1. There are three main searching algorithms: linear search, binary search, and hash search.

Linear search is the simplest searching algorithm. It starts at the beginning of the data structure and compares each element to the target value. If the target value is found, the algorithm returns the index of the element. If the target value is not found, the algorithm returns -1.

For example, let's say we have an array of numbers [1, 2, 3, 4, 5] and we want to find the number 3. The linear search algorithm would start at the beginning of the array and compare the first element, 1, to the target value, 3. Since 1 is not equal to 3, the algorithm would move on to the next element, 2. Again, 2 is not equal to 3, so the algorithm would move on to the next element, 3. Since 3 is equal to the target value, the algorithm would return the index of the element, which is 2.

Binary search is a more efficient searching algorithm than linear search. It works by repeatedly dividing the data structure in half and searching the half that is more likely to contain the target value.

For example, let's say we have the same array of numbers [1, 2, 3, 4, 5] and we want to find the number 3. The binary search algorithm would first divide the array in half, which gives us [1, 2] and [3, 4, 5]. Since the target value, 3, is greater than the middle element of the first half, [1, 2], the algorithm would search the second half, [3, 4, 5]. The algorithm would then divide the second half in half, which gives us [3] and [4, 5]. Since the target value, 3, is equal to the middle element of the second half, [3], the algorithm would return the index of the element, which is 0.

Hash search is a very efficient searching algorithm that works by using a hash function to convert the target value into a unique index. The data structure is then searched using the index.

For example, let's say we have an array of numbers [1, 2, 3, 4, 5] and we want to find the number 3. The hash function would convert the number 3 into the index 2. The data structure would then be searched using the index 2, and the element at index 2 would be returned, which is the number 3.

The best searching algorithm to use depends on the data structure and the size of the data. Linear search is the simplest and most efficient algorithm for small data structures. Binary search is more efficient than linear search for large data structures that are sorted. Hash search is the most efficient algorithm for large data structures that are not sorted.

1. Similarities:

Both linear search and binary search are searching for algorithms.

Both algorithms can be used to find an element in a data structure.

Both algorithms can be used to find the index of an element in a data structure.

Differences:

Linear search works by comparing each element in the data structure to the target value, one by one.

Binary search works by dividing the data structure in half and searching the half that is more likely to contain the target value.

Linear search is a simpler algorithm than binary search.

Binary search is more efficient than linear search for large data structures that are sorted.

Here is an example of how linear search and binary search would be used to find the number 3 in the array [1, 2, 3, 4, 5]:

#Linear search:

The linear search algorithm would start at the beginning of the array and compare the first element, 1, to the target value, 3. Since 1 is not equal to 3, the algorithm would move on to the next element, 2.

The algorithm would then compare the second element, 2, to the target value, 3. Since 2 is not equal to 3, the algorithm would move on to the next element, 3.

Since 3 is equal to the target value, the algorithm would return the index of the element, which is 2.

Binary search:

The binary search algorithm would first divide the array in half, which gives us [1, 2] and [3, 4, 5]. Since the target value, 3, is greater than the middle element of the first half, [1, 2], the algorithm would search the second half, [3, 4, 5].

The algorithm would then divide the second half in half, which gives us [3] and [4, 5]. Since the target value, 3, is equal to the middle element of the second half, [3], the algorithm would return the index of the element, which is 0.

As you can see, binary search is more efficient than linear search for large data structures that are sorted. This is because binary search only needs to compare the target value to the middle element of the data structure at each step. Linear search, on the other hand, needs to compare the target value to each element in the data structure at each step.

1. pseudocode for a function that searches for an integer variable called item using linear search in an unordered array called unordered\_array:

function linear\_search(unordered\_array, item)

for i = 0 to length(unordered\_array) - 1

if unordered\_array[i] == item

return i

end for

return -1

end function

Here is the source code for the same function in Python:

def linear\_search(unordered\_array, item):

for i in range(len(unordered\_array)):

if unordered\_array[i] == item:

return i

return -1

This function works by first looping through the unordered array, one element at a time. At each step, the function compares the current element to the target value. If the current element is equal to the target value, the function returns the index of the element. If the current element is not equal to the target value, the function continues looping. If the function reaches the end of the array without finding the target value, it returns -1.



#include <stdio.h>

int binary\_search(int \*array, int size, int value) {

int low = 0;

int high = size - 1;

int mid;

while (low <= high) {

mid = (low + high) / 2;

if (array[mid] == value) {

return mid;

} else if (array[mid] < value) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

int main() {

int array[] = {1, 3, 5, 7, 9};

int size = sizeof(array) / sizeof(array[0]);

int value;

printf("Enter the value to search: ");

scanf("%d", &value);

int index = binary\_search(array, size, value);

if (index == -1) {

printf("The value %d is not found in the array.\n", value);

} else {

printf("The value %d is found at index %d.\n", value, index);

}

return 0;

}

This program works by first initializing the low and high variables to the beginning and end of the array, respectively. The program then enters a loop that continues as long as low is less than or equal to high. At each step of the loop, the program calculates the middle index of the array, mid, and compares the value at that index to the target value. If the value at the middle index is equal to the target value, the program returns the index. If the value at the middle index is less than the target value, the program sets low to the middle index + 1. If the value at the middle index is greater than the target value, the program sets high to the middle index - 1. If the program reaches the end of the loop without finding the target value, it returns -1.



Bubble sort is a simple sorting algorithm that works by repeatedly comparing adjacent elements in an array and swapping them if they are in the wrong order. The algorithm starts at the beginning of the array and compares the first two elements. If the first element is greater than the second element, the two elements are swapped. The algorithm then moves on to the next two elements and repeats the process. This continues until the end of the array is reached.

Selection sort is another simple sorting algorithm that works by repeatedly finding the smallest element in an array and swapping it with the first element. The algorithm starts at the beginning of the array and compares the first element to the rest of the elements in the array. If the first element is greater than any of the other elements, the algorithm swaps the first element with the smallest element. The algorithm then moves on to the next element and repeats the process. This continues until the end of the array is reached.

Insertion sort is a sorting algorithm that works by repeatedly inserting elements into a sorted subarray. The algorithm starts at the beginning of the array and compares the first element to the rest of the elements in the array. If the first element is smaller than any of the other elements, the algorithm inserts the first element into the sorted subarray. The algorithm then moves on to the next element and repeats the process. This continues until the end of the array is reached.

All three of these sorting algorithms are simple to implement and can be used to sort small arrays. However, they are not very efficient for sorting large arrays. There are more efficient sorting algorithms that can be used for large arrays.