Binary search algorithm of e-education based on modern technologies

Chingiz Zaidov

Abstract
In computer science, a binary search algorithm is an algorithm designed to solve a search problem. Algorithms often evaluated by computational complexity or maximum theoretical runtime. In this article, we will talk about the basics of binary search, its working principle, advantages, shortcomings, comparison with other algorithms and areas of application.
Key words: binary, algorithm, e-education, technologies
A search is a method of searching for a specific element in a group of elements. It needs a unique identity associated with the item you want. When the unique identity of the desired element is found, the index of that element is returned. The index indicates the location or address in the list elements of the array where that particular element is found. If the requested information is found, the specific information is returned. Otherwise, it returned a null value.
There are several categories of search algorithms, such as:
Linear search
Binary search
Interpolation search
Hash table lookup
A binary search algorithm is an approach to finding the position of a particular value in a sorted array. A binary search line is the fastest and most efficient search method used to locate an element in an array.
Binary search algorithm. Binary search can only be applied to sorted arrays. For a binary search, the data must first be sorted and, if the data is in numeric form, either in ascending order or in alphabetical order (A to Z) if it is in string format. To perform a binary search on an array of data in unsorted form, it must first be sorted using an ascending sort process and then sorted using a binary search.
Binary search is a simple yet powerful data structure used for various purposes in computer science. Binary search works on sorted arrays. A binary search begins by comparing the element in the middle of the array to the target value. If the target value corresponds to an element, its position in the array is returned. If the target value is less than the element, the search continues in the lower half of the array. If the target value is greater than the element, the search continues in the upper half of the array. By doing this, the algorithm eliminates the half where the target value cannot be in each iteration. Binary Search is defined as a search algorithm that uses a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the array sorted data and reduce the time complexity to O(log N).

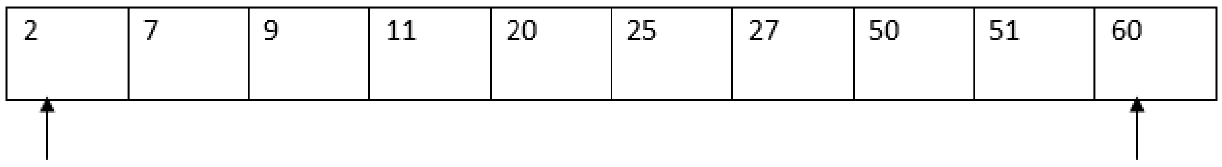
The working principle of binary search

This working principle is based on the rule of ***"Divide and Conquer"***. The divide-and-conquer method works by repeatedly dividing a problem into similar types of subproblems.
Let us understand the working principle of binary search with the help of an example. A prerequisite for binary search is to sort the array. If the data is not sorted, we cannot apply a binary search on that array. So let's understand the concept of binary search.
First, read the search item from the user as input.

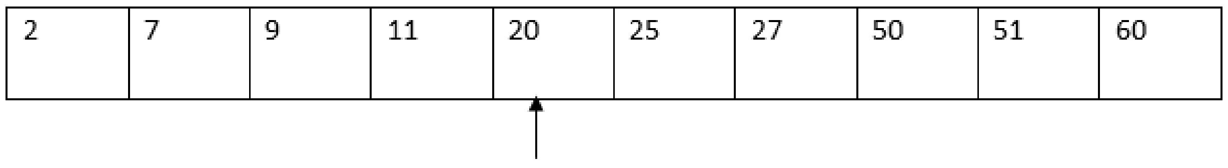
a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*j*	*l*

This array has a total of 10 elements from 0 to 9 and is arranged in ascending order.
The next step is to search for the middle element in the sorted list of arrays. Now we will find the middle element of the series.
Because we want to find the middle position of the array to split it into two halves. For this purpose, let's assume that the left value is 'start' and the right value is 'end'. In the array. So the formula will be:
Middle = [(start + end)/2]. Then a comparison will be made between the middle element in the sorted list and the search elements. Suppose if 'x' is the lookup element in the array, then there will be three cases.
Case 1:
If both elements are matched/found, then display "Given element found" and exit the funct

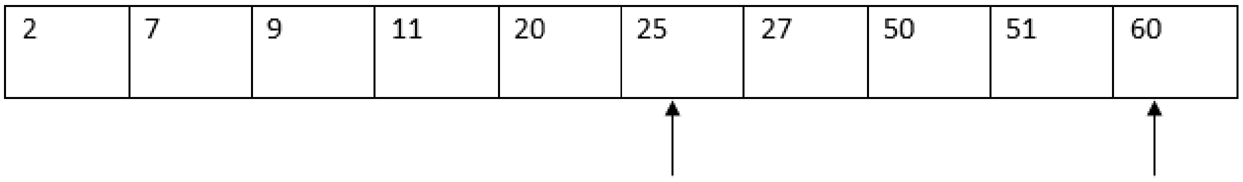
X==array [middle]
Case 2:
If the desired search element is smaller than the middle element, repeat the above procedure for the left sublist of the middle element.
X=< array [middle]
Case 3:
If the desired search element is larger than the middle element, repeat the above process for the right sub-list of the middle element.
X>=array [middle]
Further, a similar process will be repeated until the desired search item is reached. If the item also does not match the search item, display "Item not found in list".
Let's look at another example to understand the concept of binary search. We have taken the following array with elements from 0 to 9 as shown below:



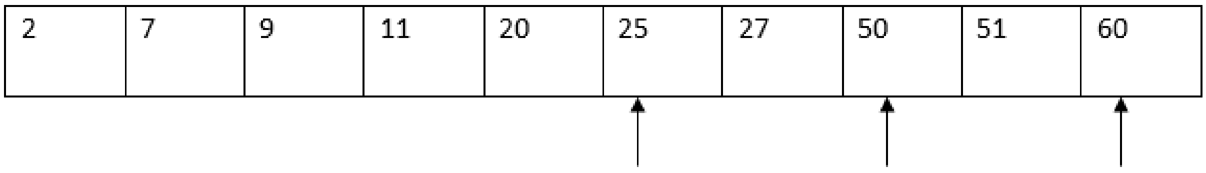
Suppose we have to search for 25. So the algorithm will check if there are 25 in this array. If it is in the array, then it should return the index at which it was found. If it is not in the array, then it should return -1. So, the first thing in a binary search algorithm is to define the range of elements to search. The leftmost position in the array is the starting index and the rightmost position is the ending index. The next step is to find the average index. The index must be an integer. So we'll get a floor value of 4.5. In this way, the average index will be 4.
Middle=[(starting index+ending index)/2]=[(0+9)/2]=4.5=4



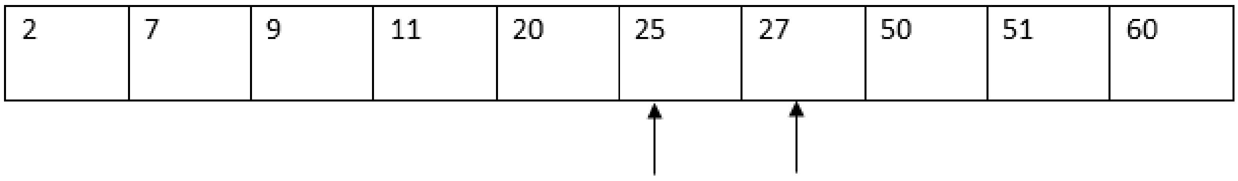
The next step is to find the location of the searched element in the array associated with the middle. Now the average element is 20. Search item 25 is greater than 20. So it will be available somewhere after the 20th. Thus, we are only interested in the elements to the right of the middle index. If A[Mid]< searches for an element, search to the right of the middle element. Now the search will start like this, the start index will be equal to Mid+1, and the end will remain the same.



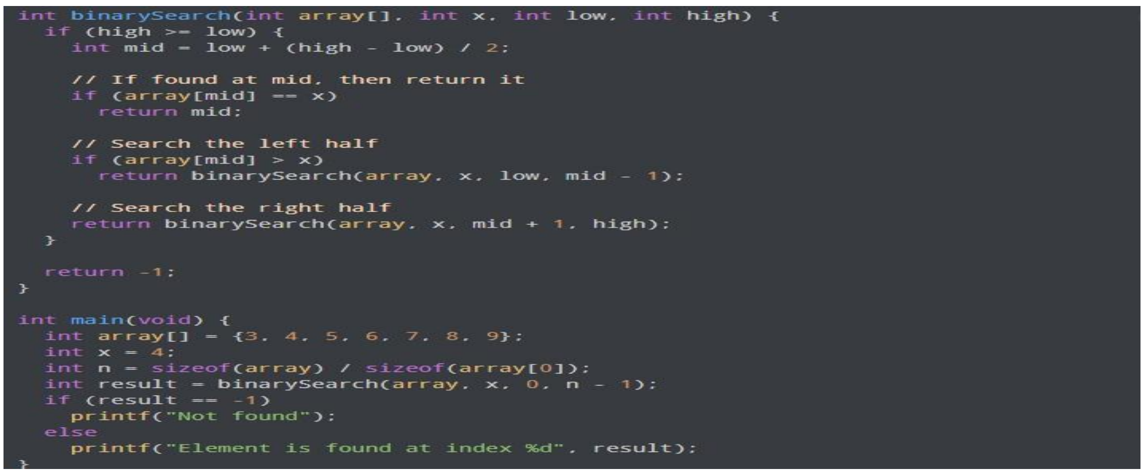
Now we need to find the mean index again by following the above procedure.
Middle=[(starting index+ending index)/2]=[(5+9)/2]=7



The search item to be rechecked is for medium. In this case, the middle element is 50 at index position 7. So the search element 25 will come either to the left or to the right of 50. Since 25 is less than 50, we will limit our search to the left of 50. If there is a search element A[Mid]> then it will be to the left of the search. Middle element. So now the start will stay the same and update with the last Mid-1. Now the array will look like this.



Middle=[(starting index+ending index)/2]=[(5+6)/2]=5.5=5
We are looking for 25 in the array. The middle element is also 25 at index position 5. So we found our search item 25. And the condition A[Mid]= the search element is completely filled.



```
int binarySearch(int array[], int x, int low, int high) {
  if (high >= low) {
    int mid = low + (high - low) / 2;

    // If found at mid, then return it
    if (array[mid] == x)
      return mid;

    // Search the left half
    if (array[mid] > x)
      return binarySearch(array, x, low, mid - 1);

    // Search the right half
    return binarySearch(array, x, mid + 1, high);
  }

  return -1;
}

int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int x = 4;
  int n = sizeof(array) / sizeof(array[0]);
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
}
```

Since the time complexity when searching a large dataset is O(log n), this means that it is much faster than linear search. When the dataset is sorted. When data is stored in contiguous memory. The data has no complex structure or relationships. This would be the best case if the algorithm performed the minimum number of comparisons. This would be the worst case if the algorithm performed the maximum number of comparisons. The time required to complete the maximum number of comparisons will be less than the time taken to complete the minimum number of comparisons. The best case can occur when the search element is equal to the first middle element of the array. The worst case will occur when the search element is at the beginning of the array list or at the end of the array. In binary search:

• Worst case is "O(1)".
• Best case is "O(log n)".
• The average case is "O(log n)".
• Binary search can be used as a building block for more complex algorithms used in machine learning, such as neural network training algorithms or finding optimal hyperparameters for a model.
• Widely used in Competitive Programming.
• Can be used for searching in computer graphics. Binary search can be used as a building block for more complex algorithms used in computer graphics, such as ray tracing or texture mapping algorithms.
• Can be used for database search. Binary search can be used to efficiently search a database of records, such as a customer database or product catalog.

Disadvantages of binary search:

• We require the array to be sorted. If the array is not sorted, we need to sort it before performing the search. This adds an additional O(N * logn) time complexity for the sorting step, which makes using binary search trivial.
• Binary search requires storing the searched data structure in contiguous memory locations. This can be a problem if the data structure is too large to fit in memory or is stored on external storage such as a hard drive or cloud.
• Binary search requires that the elements of an array are comparable, that is, they must be sortable. This can be a problem if the elements of the array are not arranged in a natural order, or if the ordering is not well defined.
• Binary search may be less efficient than other algorithms such as hash tables for searching very large data sets that cannot fit in memory.
• Binary search is faster than linear search, especially for large arrays. As the size of the array increases, the time taken to perform a linear search increases linearly, while the time required to perform a binary search increases logarithmically.

Conclusion

Binary search is more efficient than other search algorithms of similar time complexity, such as interpolation search or exponential search. Binary search is relatively simple to implement and easy to understand, making it a good choice for many applications. Binary search is well suited for searching large datasets stored on external storage, such as a hard drive or cloud. As a building block for more complex algorithms such as those used in binary search, computer graphics, and machine learning.

References:

[1] https://www.programiz.com/dsa/binary-search
[2] https://habr.com/ru/articles/563652/
[3] https://www.freecodecamp.org/news/binary-search-in-c-algorithm-example/
[4] https://www.geeksforgeeks.org/binary-search/