**Student Name:** Federico Watkins

**CSIS 1800:** Introduction to Computer Science and Information Systems

**Chapter number:** 7Problem Solving and Algorithms

**Assignment number:** 7

In all problems briefly explain the process/algorithm of searching and demonstrate all intermediate steps:

1. Demonstrate how many comparisons will be needed using a sequential search to find the value 43 in the following array?

A sequential search will compare the values at each index in ascending order to the search value. Here it finds that 14 ≠ 43.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 17 | 29 | 38 | 43 | 55 | 57 | 61 | 64 | 65 | 77 | 79 | 81 |

17 ≠ 43

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 17 | 29 | 38 | 43 | 55 | 57 | 61 | 64 | 65 | 77 | 79 | 81 |

29 ≠ 43

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 17 | 29 | 38 | 43 | 55 | 57 | 61 | 64 | 65 | 77 | 79 | 81 |

38 ≠ 43

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 17 | 29 | 38 | 43 | 55 | 57 | 61 | 64 | 65 | 77 | 79 | 81 |

43 = 43 at index[4]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 17 | 29 | 38 | 43 | 55 | 57 | 61 | 64 | 65 | 77 | 79 | 81 |

Using a sequential search to find the value 43 will require 5 comparisons

(Computer Science Illuminated, Nell Dale and John Lewis, p. 212-215)

1. Demonstrate how many comparisons would be needed using a binary search to find the value **55** in the following array?

One of the required pre-conditions of a binary search is to have a sorted array. Hence, once sorted the value 55 will be contained inside index [6]

Array:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 27 | 29 | 38 | 42 | 55 | 53 | 61 | 64 | 69 | 77 | 79 | 84 |

Array sorted:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 27 | 29 | 38 | 42 | 53 | 55 | 61 | 64 | 69 | 77 | 79 | 84 |

Binary Search assigns a low and high index then uses those values to calculate the middle index. It compares the search value to value within index[middle] returning such index if they are equal. If they are not equal, it determines if the value at index[middle] is less-than or greater-than the search value. If the value at index[middle] is less-than the search value, it re-assign index[high] to be the index one place before index[middle]. If the value at index[middle] is greater-than the search value, it re-assign index[low] to be the index one place after index[middle].

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| 14 | 27 | 29 | 38 | 42 | 53 | 55 | 61 | 64 | 69 | 77 | 79 | 84 |
| low |  |  |  |  |  | middle |  |  |  |  |  | high |

After sorting the array as required by the pre-condition of a binary search, using a binary search to find the value 55 will require 1 comparisons.

(Computer Science Illuminated, Nell Dale and John Lewis, p. 215-218)

1. Demonstrate all intermediate steps of the selection sort algorithm that has executed 3 times on this array:

Selection sort will take the value of the first index and assign it as the current lowest value. It then iterates through the array comparing every other value to the current lowest-value. If a lower value is not found it cement the current lowest as being in-order. If it does find a lower value than that of the current lowest it will swap them.

***1 Time:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 19 | 22 | 11 | 24 | 7 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 22 | 11 | 24 | 7 | 19 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 22 | 11 | 24 | 7 | 19 |

***2 Times:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 22 | 11 | 24 | 7 | 19 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 7 | 11 | 24 | 22 | 19 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 7 | 11 | 24 | 22 | 19 |

***3 Times:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 7 | 11 | 24 | 22 | 19 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 7 | 11 | 24 | 22 | 19 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 5 | 7 | 11 | 24 | 22 | 19 |

(Computer Science Illuminated, Nell Dale and John Lewis, p. 218-221)

1. Demonstrate all intermediate steps of the of the bubble sort algorithm that has executed 2 times on this array:

Bubble Sort will compare the values within the first index to that of the next index, swapping the larger value to the larger index. After doing this with the first index it does the same with the next index, thus comparing index[1] to index[2] then index[2] to index[3] and so on until the entire array is in order. Hence, Bubble Sort has the behavior that, after each iteration, the largest unordered-value will always be in-order.

***1 Time:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 15 | 13 | 77 | 44 | 22 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 77 | 44 | 22 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 77 | 44 | 22 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 44 | 77 | 22 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 44 | 22 | 77 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 44 | 22 | 5 | 77 |

***2 Times:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 44 | 22 | 5 | 77 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 44 | 22 | 5 | 77 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 44 | 22 | 5 | 77 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 22 | 44 | 5 | 77 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 13 | 15 | 22 | 5 | 44 | 77 |

(Computer Science Illuminated, Nell Dale and John Lewis, p. 221-223)

1. Demonstrate all intermediate steps of the selection sort algorithm that has executed 3 times on this array:

Selection sort will take the value of the first index and assign it as the current lowest value. It then iterates through the array comparing every other value to the current lowest-value. If a lower value is not found it cement the current lowest as being in-order. If it does find a lower value than that of the current lowest it will swap them.

***1 Time:***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 3 | 11 | 11 | 1 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 11 | 11 | 3 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 11 | 11 | 3 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

***2 Times:***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 11 | 11 | 3 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 3 | 11 | 11 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 3 | 11 | 11 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

***3 Times:***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 3 | 11 | 11 | 9 | 7 | 5 | 17 | 8 | 14 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 3 | 5 | 11 | 9 | 7 | 11 | 17 | 8 | 14 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
| 1 | 3 | 5 | 11 | 9 | 7 | 11 | 17 | 8 | 14 | 9 |

(Computer Science Illuminated, Nell Dale and John Lewis, p. 218-221)

1. Show the state of the following array after the first recursive call is made in the Quicksort algorithm, using the value at index 0 as the split value. Explain your results!

Using the value at index[0] as the split value, the left pointer is set to the value at index[1] and the right pointer is set to the value at index[5].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 2 | 17 | 33 | 9 | 1 | 11 |

Because the value at index[1] is greater than the (split) value at index[0], the left pointer stops at index[1]. And because the value at index[5] is greater than the (split) value at index[0], the right pointer moves over to index[4] where it stops because the value there is less than the split.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 2 | 17 | 33 | 9 | 1 | 11 |

The values at the left and right pointers are swapped

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 2 | 1 | 33 | 9 | 17 | 11 |

The left pointer continues to index[2] where it stops because the value there is greater than the split. The right pointer continues to index[3] where it finds that its values is greater than the split so it moves to index[2] and sees that it has already been compared to the split by the left pointer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 2 | 1 | 33 | 9 | 17 | 11 |

Since the pointers have completed their iterations the value of split is swapped with the value of the index[] one spot before the left pointer (index[2]). Values at index[0] and index[1] are cemented.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 2 | 33 | 9 | 17 | 11 |

(Computer Science Illuminated, Nell Dale and John Lewis, p. 228-229)

1. How many comparisons does it take using the binary search to find the value **16** in the following array? Sort the array prior of applying the algorithm.

Binary Search assigns a low and high index then uses those values to calculate the middle index. It compares the search value to value within index[middle] returning such index if they are equal. If they are not equal, it determines if the value at index[middle] is less-than or greater-than the search value. If the value at index[middle] is less-than the search value, it re-assign index[high] to be the index one place before index[middle]. If the value at index[middle] is greater-than the search value, it re-assign index[low] to be the index one place after index[middle].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 11 | 12 | 17 | 23 | 97 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 11 | 12 | 17 | 23 | 97 |
| low |  | middle |  |  | high |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 11 | 12 | 17 | 23 | 97 |
| low |  | middle |  |  | high |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 11 | 12 | 17 | 23 | 97 |
| low |  | middle |  |  | high |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 11 | 12 | 17 | 23 | 97 |
| low |  | middle |  |  | high |

Using the binary search to find the value 16 in the array takes 3 comparison which shows that the value 16 is not found within the array.

(Computer Science Illuminated, Nell Dale and John Lewis, p. 215-218)

1. Demonstrate all intermediate steps of the binary search to find the value 17. Sort the array prior of applying the algorithm.

Binary Search assigns a low and high index then uses those values to calculate the middle index. It compares the search value to value within index[middle] returning such index if they are equal. If they are not equal, it determines if the value at index[middle] is less-than or greater-than the search value. If the value at index[middle] is less-than the search value, it re-assign index[high] to be the index one place before index[middle]. If the value at index[middle] is greater-than the search value, it re-assign index[low] to be the index one place after index[middle].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 21 | 1 | 9 | 23 | 17 | 11 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 9 | 11 | 17 | 21 | 23 |
| low |  | middle |  |  | high |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 9 | 11 | 17 | 21 | 23 |
|  |  |  | low | middle | high |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 9 | 11 | 17 | 21 | 23 |
|  |  |  | low-mid-high |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] |
| 1 | 9 | 11 | 17 | 21 | 23 |
|  |  |  | low-mid-high |  |  |

Using the binary search to find the value 17 in the array takes 3 comparison.

(Computer Science Illuminated, Nell Dale and John Lewis, p. 215-218)