Lab10 – Algorithms and Sorting and Time Complexity

Enrique Saracho Felix

100406980

CPSC 1150

28/07/2023

# Exercise 1

## Part A

### Program SortComparisons

**File name:** SortComparisons.java

**Purpose:** Displays the average number of comparisons linear and binary search algorithms make to find an element in different sizes of lists.

**Packages:** java.util.Arrays

**Input:** No input needed.

**Output:** A table of strings with doubles representing the values of the average number of comparisons for each algorithm in various sizes of lists.

**Pseudocode:**

Algorithm *SortComparisons*

START

(**main**)

For size equal to 10, 100, 1,000, 10,000, and 100,000, {

Set list as an array of integers = **genArray**(size)

Sort list

Set avgLnS as double = **averageLinear**(list)

Set avgBS as double = **averageBinary**(list)

Print size, avgLnS, avgBS

}

(**genArray**, parameter: size(integer))

Set arr as an array of integers of size size

For each value in arr

value = random integer in range [0, size)

Return arr

(**linearSearch**, parameters: arr(array of integers), num(integer))

Set comp as integer

For each value in arr {

comp += 1

If value = num

Return comp

}

Return comp

(**binarySearch**, parameters: arr(array of integers), num(integer))

Set start as integer = 0

Set end as integer = last element of arr

return **binarySearch**(arr, num, start, end, 0)

(**binarySearch**, parameters: arr(array of integers), num(integer), start(integer), end(integer), comp(integer))

If start > end

return comp

Set mid as integer = (start + end) / 2

comp += 1

If arr[mid] == num

return comp

else if arr[mid] < num

return **binarySearch**(arr, num, mid + 1, end, comp)

else

Return **binarySearch**(arr, num, start, mid – 1, comp)

(**averageLinear**, parameter: arr(array of integers))

Set sum as integer = 0

For each value in arr {

sum += **linearSearch**(arr, value)

}

Return sum / length of arr (as double)

(**averageBinary**, parameter: arr(array of integers))

Set sum as integer = 0

For each value in arr {

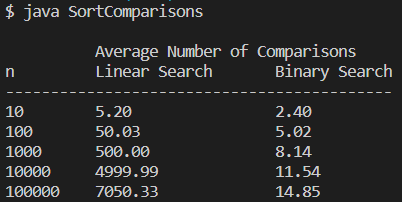
sum += **binarySearch**(arr, value)

}

Return sum / length of arr (as double)

END *SortComparisons*

**Test run(s):**

****

## Part B

The binary search algorithm on average needs less comparisons to find the solution than the linear algorithm.

## Part C

Linear:

Comparisons ≈ n / 2

Binary:

Average ≈ ln(n)

## Part D

If I were to choose a search algorithm between the two I would choose binary, because it’s more efficient.

# Exercise 2