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 {

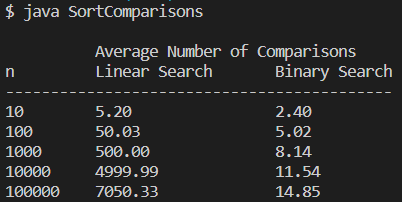
sum += **binarySearch**(arr, value)

}

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

## Program MergeSort

**File name:** MergeSort.java

**Purpose:** Implements the merge sort algorithm.

**Input:** No input needed.

**Output:** In a string message: the elements of an array before the merge sort, and the elements after the merge sort.

**Pseudocode:**

Algorithm *MergeSort*

START

(**main**)

Set list1 as array of integers of size 20

**printArray**(“The array before merge sort: ”, list1)

**mergeSort**(list1)

**printArray**(“The array before merge sort: ”, list1)

(**mergeSort**, parameter: list(array of integers))

If (length of list > 1) {

Set mid as integer = length of list / 2

Set firstHalf as array of integers of size mid

**copyArray**(list, firstHalf, 0, mid)

**mergeSort**(firstHalf)

Set secondHalf as array of integers of size (length of list – mid)

**copyArray**(list, secondHalf, mid, length of list)

**mergeSort**(secondHalf)

**merge**(firstHalf, secondHalf, list)

}

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

Set list as array of integers of size size

For each element in list

element = random integer in range [0, 100)

Return list

(**printArray**, parameters: header(string), arr(array of integers))

Print header

For each element in arr

Print element + space

Print new line

(**copyArray**, parameters: sourceLs(array of integers), destLs(array of integers), start(integer), end(integer))

For each index from start to end(exclusive)

destLs[index - start] = sourceLs[index]

(**merge**, parameters: firstHalf(array of integers), secondHalf(array of integers), list(array of integers))

Set i, j, and k as integers = 0

For each i from 0 to length of list {

If j = length of firstHalf {

list[ i ] = secondHafl[ k ]

k += 1

Continue to next iteration

}

If k = length of secondHalf {

list[ i ] = fistHalf[ j ]

j += 1

Continue to next iteration

}

If firstHalf[ j ] < secondHalf[ k ] {

list[ i ] = firstHalf[ j ]

j++

} Else {

list[ i ] = secondHalf[ k ]

k++

}

}

END *MergeSort*

**Test run(s):**

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated