**Group Project**

**TEAM: Andrew Saad, and Jonathan Linder**

Step 1: Select any three sorting algorithm and two searching algorithms

SORT 1

**package** finalProject;

**public** **class** SelectionSort

{

**public** **static** **void** sort(**int** arr[])

{

**int** n = arr.length;

// One by one move boundary of unsorted subarray

**for** (**int** i = 0; i < n-1; i++)

{

// Find the minimum element in unsorted array

**int** min\_idx = i;

**for** (**int** j = i+1; j < n; j++)

**if** (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first

// element

**int** temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

}

SORT 2

**package** finalProject;

//Java program for implementation of Insertion Sort

**public** **class** InsertionSort

{

/\*Function to sort array using insertion sort\*/

**public** **static** **void** sort(**int** arr[])

{

**int** n = arr.length;

**for** (**int** i=1; i<n; ++i)

{

**int** key = arr[i];

**int** j = i-1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

**while** (j>=0 && arr[j] > key)

{

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = key;

}

}

} /\* This code is contributed by Rajat Mishra. \*/

SORT 3

**package** finalProject;

/\* Java program for Merge Sort \*/

**public** **class** MergeSort

{

**static** **int** *count* = 0;

**private** **static** **void** merge(**int**[] first, **int**[] second, **int**[] a)

{

**int** iFirst = 0; // Next element to consider in the first array

**int** iSecond = 0; // Next element to consider in the second array

**int** j = 0; // Next open position in a

// As long as neither iFirst nor iSecond is past the end, move

// the smaller element into a

**while** (iFirst < first.length && iSecond < second.length)

{

**if** (first[iFirst] < second[iSecond])

{

a[j] = first[iFirst];

iFirst++;

} **else**

{

a[j] = second[iSecond];

iSecond++;

}

j++;

}

// Note that only one of the two loops below copies entries

// Copy any remaining entries of the first array

**while** (iFirst < first.length)

{

a[j] = first[iFirst];

iFirst++;

j++;

*count*++;

}

// Copy any remaining entries of the second half

**while** (iSecond < second.length)

{

a[j] = second[iSecond];

iSecond++;

j++;

*count*++;

}

}

**public** **static** **void** sort(**int**[] list)

{

**int** r = list.length;

**if** (list.length <= 1)

{

**return**;

}

**int**[] left = **new** **int**[(r / 2)];

**int**[] right = **new** **int**[r - r / 2];

**for** (**int** i = 0; i < left.length; i++)

{

left[i] = list[i];

}

**for** (**int** i = 0; i < right.length; i++)

{

right[i] = list[left.length + i];

}

*sort*(left);

*sort*(right);

*merge*(left, right, list);

}

}

SEARCH 1

package finalProject;

import java.util.Iterator;

import java.util.List;

public class LinearSearch

{

// This function returns index of element x in arr[]

public static int search(int[] list, int x)

{

int position = -1;

for (int i = 0; i < list.length; i++)

{

++position;

if (list[i] == x)

return position;

}

// return -1 if the element is not found

return -1;

}

}

SEARCH 2:

Please refer to Oracle’s documentation here: <https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#binarySearch(int[],%20int)>

SEARCH 1:

**package** finalProject;

//Java implementation of recursive Binary Search

**public** **class** BinarySearch

{

// Returns index of x if it is present in arr[l..r], else

// return -1

**public** **static** **int** binarySearch(**int** arr[], **int** l, **int** r, **int** x)

{

**if** (r>=l)

{

**int** mid = l + (r - l)/2;

// If the element is present at the middle itself

**if** (arr[mid] == x)

**return** mid;

// If element is smaller than mid, then it can only

// be present in left subarray

**if** (arr[mid] > x)

**return** *binarySearch*(arr, l, mid-1, x);

// Else the element can only be present in right

// subarray

**return** *binarySearch*(arr, mid+1, r, x);

}

// We reach here when element is not present in array

**return** -1;

}

}

/\* This code is contributed by Rajat Mishra \*/

Step 2: Understand the logic of all the algorithms

Step 3: Create java program and convert your algorithm into the source code

Step 4: Create a random function that generates input number for the same program

**int** arraySizes = 0;

**int** counterArray = 0;

Scanner in = **new** Scanner(System.***in***);

//modify number of entries to sort and search through below

System.***out***.println("How many random numbers? \n(Warning over 200,000 may take very long)");

**while** (counterArray == 0)

{

arraySizes = (**int**) in.nextInt();

counterArray++;

}

**long** timeOfAction = 0;

**long** timeOfSort = 0;

**long** timeB4 = 0;

**long** timeAfter = 0;

**int** position = -1;

**int**[] list = **new** **int**[arraySizes];

**int**[] list1 = list;

**int**[] list2 = list;

**int**[] list3 = list;

**int**[] list4 = list;

Step 5: You need to collect the output

System.***out***.println("Building data..." + "\n");

**for**(**int** i = 0; i < list.length; i++)

{

list[i] = r.nextInt((maxRange - minRange) + 1) + minRange;

list1[i] = list[i];

list2[i] = list[i];

list3[i] = list[i];

list4[i] = list[i];

}

Step 6: Create start transaction and end transaction point in your source code

**long** timeOfAction = 0;

**long** timeOfSort = 0;

**long** timeB4 = 0;

**long** timeAfter = 0;

Step 7: Calculate the time in milliseconds for each java program

System.***out***.println("\nInsertion Sort....");

timeB4 = System.*currentTimeMillis*();

InsertionSort.*sort*(list2);

timeAfter = System.*currentTimeMillis*();

timeOfInsertion = timeAfter - timeB4;

System.***out***.println("Insertion Sort took: " + timeOfInsertion + "ms");

System.***out***.println("\nSelection Sort....");

timeB4 = System.*currentTimeMillis*();

SelectionSort.*sort*(list);

timeAfter = System.*currentTimeMillis*();

timeOfSelection = timeAfter - timeB4;

System.***out***.println("Selection Sort took: " + timeOfSelection + "ms");

System.***out***.println("\nMerge Sort....");

timeB4 = System.*currentTimeMillis*();

MergeSort.*sort*(list1);

timeAfter = System.*currentTimeMillis*();

timeOfMerge = timeAfter - timeB4;

System.***out***.println("Merge Sort took: " + timeOfMerge + "ms");

System.***out***.println("\nLinear Search....");

timeB4 = System.*currentTimeMillis*();

position = LinearSearch.*search*(list3, searchNum);

timeAfter = System.*currentTimeMillis*();

System.***out***.println("Found at position " + position);

timeOfLinear = timeAfter - timeB4;

System.***out***.println("Linear search took: " + timeOfLinear + "ms");

System.***out***.println("\nArrays.Binary Search....");

timeB4 = System.*currentTimeMillis*();

Arrays.*sort*(list3);

timeAfter = System.*currentTimeMillis*();

timeOfSort = timeAfter - timeB4;

timeB4 = System.*currentTimeMillis*();

position = Arrays.*binarySearch*(list3, searchNum);

timeAfter = System.*currentTimeMillis*();

**if** (position < 0)

{

position = -1;

}

System.***out***.println("Found at position " + position);

timeOfBinary = timeAfter - timeB4;

System.***out***.println("Binary search took: " + timeOfBinary + "ms" +

"\n+ " + timeOfSort + "ms for Arrays.sort() to prepare data for search.\nTotal time of binary: "

+ (timeOfSort + timeOfBinary) + "ms");

System.***out***.println("\nBinary Search....");

timeB4 = System.*currentTimeMillis*();

Arrays.*sort*(list4);

timeAfter = System.*currentTimeMillis*();

timeOfSort = timeAfter - timeB4;

timeB4 = System.*currentTimeMillis*();

position = BinarySearch.*binarySearch*(list4, 0, list4.length, searchNum);

timeAfter = System.*currentTimeMillis*();

**if** (position < 0)

{

position = -1;

}

System.***out***.println("Found at position " + position);

timeOfAction = timeAfter - timeB4;

System.***out***.println("Binary search took: " + timeOfAction + "ms" +

"\n+ " + timeOfSort + "ms for Arrays.sort() to prepare data for search.\nTotal time of binary: "

+ (timeOfSort + timeOfAction) + "ms");

Step 8: Compare the performance of each algorithm.

FULL MAIN CODE BELOW:

package finalProject;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

public class MainTester

{

@SuppressWarnings("resource")

public static void main(String[] args) throws Exception

{

int arraySizes = 0;

int counterArray = 0;

Scanner in = new Scanner(System.in);

//modify number of entries to sort and search through below

System.out.println("How many random numbers? \n(Warning over 200,000 may take very long)");

while (counterArray == 0)

{

arraySizes = (int) in.nextInt();

counterArray++;

}

long timeOfAction = 0;

long timeOfSort = 0;

long timeB4 = 0;

long timeAfter = 0;

int position = -1;

int[] list = new int[arraySizes];

int[] list1 = list;

int[] list2 = list;

int[] list3 = list;

int[] list4 = list;

//modify the integer to search for below

int searchNum = 0;

int counterSearch = 0;

System.out.println("What integer would you like to search for?");

while (counterSearch == 0)

{

searchNum = (int) in.nextInt();

counterSearch++;

}

//modify range of integers below

int minRange = 0;

int counterMin = 0;

System.out.println("What is the minimum range of the data?");

while (counterMin == 0)

{

minRange = (int) in.nextInt();

counterMin++;

}

int maxRange = 0;

System.out.println("What is the maximum range of the data?");

int counterMax = 0;

while (counterMax == 0)

{

maxRange = (int) in.nextInt();

counterMax++;

}

Random r = new Random();

System.out.println("Building data...");

for(int i = 0; i < list.length; i++)

{

list[i] = r.nextInt((maxRange - minRange) + 1) + minRange;

list1[i] = list[i];

list2[i] = list[i];

list3[i] = list[i];

list4[i] = list[i];

}

System.out.println("\nInsertion Sort....");

timeB4 = System.currentTimeMillis();

InsertionSort.sort(list2);

timeAfter = System.currentTimeMillis();

timeOfAction = timeAfter - timeB4;

System.out.println("Insertion Sort took: " + timeOfAction + "ms");

System.out.println("\nSelection Sort....");

timeB4 = System.currentTimeMillis();

SelectionSort.sort(list);

timeAfter = System.currentTimeMillis();

timeOfAction = timeAfter - timeB4;

System.out.println("Selection Sort took: " + timeOfAction + "ms");

System.out.println("\nMerge Sort....");

timeB4 = System.currentTimeMillis();

MergeSort.sort(list1);

timeAfter = System.currentTimeMillis();

timeOfAction = timeAfter - timeB4;

System.out.println("Merge Sort took: " + timeOfAction + "ms");

System.out.println("\nLinear Search....");

timeB4 = System.currentTimeMillis();

position = LinearSearch.search(list3, searchNum);

timeAfter = System.currentTimeMillis();

System.out.println("Found at position " + position);

timeOfAction = timeAfter - timeB4;

System.out.println("Linear search took: " + timeOfAction + "ms");

System.out.println("\nArrays.Binary Search....");

timeB4 = System.currentTimeMillis();

Arrays.sort(list3);

timeAfter = System.currentTimeMillis();

timeOfSort = timeAfter - timeB4;

timeB4 = System.currentTimeMillis();

position = Arrays.binarySearch(list3, searchNum);

timeAfter = System.currentTimeMillis();

if (position < 0)

{

position = -1;

}

System.out.println("Found at position " + position);

timeOfAction = timeAfter - timeB4;

System.out.println("Binary search took: " + timeOfAction + "ms" +

"\n+ " + timeOfSort + "ms for Arrays.sort() to prepare data for search.\nTotal time of binary: "

+ (timeOfSort + timeOfAction) + "ms");

System.out.println("\nBinary Search....");

timeB4 = System.currentTimeMillis();

Arrays.sort(list4);

timeAfter = System.currentTimeMillis();

timeOfSort = timeAfter - timeB4;

timeB4 = System.currentTimeMillis();

position = BinarySearch.binarySearch(list4, 0, list4.length, searchNum);

timeAfter = System.currentTimeMillis();

if (position < 0)

{

position = -1;

}

System.out.println("Found at position " + position);

timeOfAction = timeAfter - timeB4;

System.out.println("Binary search took: " + timeOfAction + "ms" +

"\n+ " + timeOfSort + "ms for Arrays.sort() to prepare data for search.\nTotal time of binary: "

+ (timeOfSort + timeOfAction) + "ms");

}

}