**DATA STRUCTURE AND ALGORITHM**

MACHINE PROBLEM 3: CONTACT FINDER

**Submitted by:**

**VARGAS**, John Lloyd E.

**Submitted to:**

Ma’am **FERNANDO**, Donna Q.

**Table of Contents**

[Title Page………………………………………………………………………………………….1](#Title_page)

[Table of Contents………………………………………………………………………………….2](#toc)

[Output screenshot…………………………………………………………………………..…...…3](#mainmenu)

[Main menu…………………………………………………………………….…………..3](#mainmenu)

[Search by ID (Binary Search)…………………………………………………………......4](#searchbyid)

[Search by Name (Linear Search)……………………………………………………….....5](#searchbyname)

[Show Statistics………………………………………………………………………….....6](#showstats)

[Unique Names and Frequency………………………………………………………….....7](#uniquenames)

[Exit………………………………………………………………………………………...8](#exit)

[Java source codes………………………………………………………………………………….9](#main)

[Main.java…………….........................................................................................................9](#main)

[Statistics.java……………………………………………………………………………………………14](#stats)

**=================**

**Main menu**

**==================**

A screen shot of a computer

AI-generated content may be incorrect.

**====================================**

**Search by ID (Binary Search)**

**===================================**

**A screenshot of a computer

AI-generated content may be incorrect.**

The binary search algorithm is a good choice along with the insertion sort *(sorting algorithm)* when searching for an identification especially *unique* identifiers (UIDs). Binary search strength relies on a sorting algorithm, especially at a large dataset such as these **2 000** IDs. Because of its efficiency, the binary search algorithm can search in a large dataset in only less than 20 comparisons.

**=====================================**

**Search by Name (Linear Search)**

**=====================================**

**A screenshot of a computer

AI-generated content may be incorrect.**

The linear search algorithm is mostly used in a small dataset. However, the linear search algorithm does not require sorting, unlike the binary search. The linear search algorithm is used for searching names, as the names are not sorted out, whereas if the binary search algorithm is used here, it is most likely that it will take longer to determine the key than the linear search algorithm. Although this algorithm is slower, it is reliable when it comes to unsorted datasets.

**=====================================**

**Show Statistics**

**=====================================**

**A screenshot of a computer

AI-generated content may be incorrect.**

**=================================================**

**Unique Names and Frequency (Under statistics menu)**

**=========================A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.=========================**

**=========================**

**[Exit](#exit)**

**=========================**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Main.java**

import *java.io.BufferedReader*;  
import *java.io.FileNotFoundException*;  
import *java.io.FileReader*;  
import *java.util.ArrayList*;  
import *java.util.Scanner*;  
  
public class *Main* {  
 public static int binaryComparisonCount = 0;  
 public static int linearComparisonCount = 0;  
 public static long binarySearchTime = 0;  
 public static long linearSearchTime = 0;  
 public static int lastFoundIndex = -1;  
 public static *String* lastFoundValue = "";  
 private *BufferedReader* br;  
 private *Scanner* in = new *Scanner*(*System*.in);  
 private *ArrayList*<*ArrayList*<*String*>> contactRecords = new *ArrayList*<>();  
 private int binarySearchCount = 0;  
 private int linearSearchCount = 0;  
 private *String* recentKeyValue;  
 private boolean recentKeyResult = false;  
 private boolean isBinarySearchLastUsed = true;  
  
 public *Main*() {  
 try {  
 br = new *BufferedReader*(new *FileReader*("Contact Records.csv"));  
 *addToArray*();  
 } catch (*FileNotFoundException ae*) {  
 *System*.out.*println*("File not found");  
 throw new *RuntimeException*();  
 }  
 }  
  
 public static void *main*(*String*[] *args*) {  
 new *Main*().*menu*();  
 }  
  
 private void *menu*() {  
 while (true) {  
 *System*.out.*println*();  
 *System*.out.*println*("═".*repeat*(50));  
 *System*.out.*println*("📇 CONTACT MANAGEMENT SYSTEM");  
 *System*.out.*println*("═".*repeat*(50));  
 *System*.out.*println*("[1] 🔍 Search by ID (Binary Search)");  
 *System*.out.*println*("[2] 🔎 Search by Name (Linear Search)");  
 *System*.out.*println*("[3] 📊 Show Statistics");  
 *System*.out.*println*("[4] 🚪 Exit");  
 *System*.out.*println*("─".*repeat*(50));  
 *System*.out.*print*("Enter choice >> ");  
 int userChoice = *Integer*.*parseInt*(in.*nextLine*());  
  
 switch (userChoice) {  
 case 1 -> *searchByID*();  
 case 2 -> *searchByName*();  
 case 3 -> {  
 *Statistics* stats = new *Statistics*(contactRecords, linearSearchCount, binarySearchCount,  
 recentKeyValue, recentKeyResult, isBinarySearchLastUsed);  
 stats.*summarizeStats*();  
 stats.*menu*();  
 }  
 case 4 -> {  
 *System*.out.*println*("\n👋 Exiting program... Goodbye!");  
 return;  
 }  
 default -> *System*.out.*println*("⚠️ Invalid option. Try again.");  
 }  
 }  
 }  
  
 private void *searchByID*() {  
 *insertionSort*();  
 *System*.out.*println*("\n═ SEARCH BY ID ═");  
 *System*.out.*print*("Enter ID to find >> ");  
 int key = *Integer*.*parseInt*(in.*nextLine*());  
  
 long start = *System*.*nanoTime*();  
 *ArrayList*<*ArrayList*<*String*>> findId = *binarySearch*(key);  
 long end = *System*.*nanoTime*();  
 binarySearchTime = end - start;  
  
 recentKeyValue = key + "";  
 binarySearchCount++;  
 *System*.out.*println*("─".*repeat*(50));  
 if (findId == null) {  
 *System*.out.*println*("❌ ID " + key + " not found.");  
 recentKeyResult = false;  
 return;  
 }  
 *System*.out.*printf*("%-10s | %-30s%n", "ID", "Name");  
 *System*.out.*println*("─".*repeat*(50));  
 *System*.out.*printf*("%-10s | %-30s%n", findId.*getFirst*().*getFirst*(), findId.*getFirst*().*get*(1));  
 *Statistics* stats = new *Statistics*(contactRecords, linearSearchCount, binarySearchCount,  
 recentKeyValue, recentKeyResult, isBinarySearchLastUsed);  
 stats.*trackMetrics*();  
 recentKeyResult = true;  
 }  
  
 private void *searchByName*() {  
  
 *System*.out.*println*("\n═ SEARCH BY NAME ═");  
 *System*.out.*print*("Enter name to find >> ");  
 *String* name = in.*nextLine*();  
  
 long start = *System*.*nanoTime*();  
 *ArrayList*<*ArrayList*<*String*>> findName = *linearSearch*(name);  
 long end = *System*.*nanoTime*();  
 linearSearchTime = end - start;  
  
 recentKeyValue = name;  
 linearSearchCount++;  
  
 *System*.out.*println*("\n" + "─".*repeat*(50));  
 *System*.out.*println*("🔎 Found " + findName.*size*() + " record(s) for name: " + name);  
 *System*.out.*println*("─".*repeat*(50));  
  
 if (findName.*isEmpty*()) {  
 *System*.out.*println*("❌ No matching name found.");  
 } else {  
 *System*.out.*printf*("%-5s %-10s | %-30s%n", "#", "ID", "Name");  
 *System*.out.*println*("─".*repeat*(50));  
 for (int i = 0; i < findName.*size*(); i++) {  
 *System*.out.*printf*("%-5s %-10s | %-30s%n", (i + 1) + ".", findName.*get*(i).*get*(0), findName.*get*(i).*get*(1));  
 }  
 }  
 *Statistics* stats = new *Statistics*(contactRecords, linearSearchCount, binarySearchCount,  
 recentKeyValue, recentKeyResult, isBinarySearchLastUsed);  
 stats.*trackMetrics*();  
 recentKeyResult = !findName.*isEmpty*();  
 }  
  
 private *ArrayList*<*ArrayList*<*String*>> *binarySearch*(int *key*) {  
 isBinarySearchLastUsed = true;  
 int left = 0;  
 int right = contactRecords.*size*() - 1;  
 *ArrayList*<*ArrayList*<*String*>> result = new *ArrayList*<>();  
 binaryComparisonCount = 0;  
 lastFoundIndex = -1;  
 lastFoundValue = "";  
  
 while (left <= right) {  
 int middle = left + (right - left) / 2;  
 binaryComparisonCount++;  
 int middleId = *Integer*.*parseInt*(contactRecords.*get*(middle).*getFirst*());  
  
 if (middleId == *key*) {  
 lastFoundIndex = middle;  
 lastFoundValue = contactRecords.*get*(middle).*get*(1);  
 result.*add*(contactRecords.*get*(middle));  
 return result;  
 }  
  
 if (middleId < *key*) {  
 left = middle + 1;  
 } else {  
 right = middle - 1;  
 }  
 }  
 return null;  
 }  
  
 private *ArrayList*<*ArrayList*<*String*>> *linearSearch*(*String key*) {  
 isBinarySearchLastUsed = false;  
 *ArrayList*<*ArrayList*<*String*>> names = new *ArrayList*<>();  
 linearComparisonCount = 0;  
 lastFoundIndex = -1;  
 lastFoundValue = "";  
  
 for (int i = 0; i < contactRecords.*size*(); i++) {  
 linearComparisonCount++;  
 *String* fullName = contactRecords.*get*(i).*get*(1);  
 *String* firstName = fullName.*substring*(0, fullName.*indexOf*(' '));  
  
 if (firstName.*equalsIgnoreCase*(*key*) || fullName.*equalsIgnoreCase*(*key*)) {  
 names.*add*(contactRecords.*get*(i));  
 lastFoundIndex = i;  
 lastFoundValue = fullName;  
 }  
 }  
 return names;  
 }  
  
 private void *insertionSort*() {  
 for (int i = 1; i < contactRecords.*size*(); i++) {  
 *ArrayList*<*String*> currentRow = contactRecords.*get*(i);  
 int currentValue = *Integer*.*parseInt*(currentRow.*get*(0));  
 int j = i - 1;  
  
 while (j >= 0 && *Integer*.*parseInt*(contactRecords.*get*(j).*get*(0)) > currentValue) {  
 contactRecords.*set*(j + 1, contactRecords.*get*(j));  
 j--;  
 }  
 contactRecords.*set*(j + 1, currentRow);  
 }  
 }  
  
 private void *addToArray*() {  
 try {  
 *String* line = br.*readLine*();  
 while ((line = br.*readLine*()) != null) {  
 *String*[] tempLine = line.*split*(",");  
 *ArrayList*<*String*> row = new *ArrayList*<>();  
 row.*add*(tempLine[0]);  
 row.*add*(tempLine[1]);  
 contactRecords.*add*(row);  
 }  
 } catch (*Exception e*) {  
 throw new *RuntimeException*(*e*);  
 }  
 }  
}

**Statistics.java**

import *java.util.ArrayList*;  
import *java.util.Scanner*;  
  
public class *Statistics* {  
 private final *Scanner* in = new *Scanner*(*System*.in);  
 private final int totalNumberOfContacts;  
 private final *ArrayList*<*ArrayList*<*String*>> contactRecords;  
 private final int minimumID;  
 private final int maximumID;  
 private final int linearSearchCount;  
 private final int binarySearchCount;  
 private final *String* recentKeyValue;  
 private final boolean recentKeyResult;  
 private boolean isBinarySearchLastUsed;  
 private *ArrayList*<*String*> uniqueNames = new *ArrayList*<>();  
  
 public *Statistics*(*ArrayList*<*ArrayList*<*String*>> *contactRecords*, int *linearSearchCount*, int *binarySearchCount*,  
 *String recentKeyValue*, boolean *recentKeyResult*, boolean *isBinarySearchLastUsed*) {  
 this.contactRecords = *contactRecords*;  
 this.linearSearchCount = *linearSearchCount*;  
 this.binarySearchCount = *binarySearchCount*;  
 this.recentKeyValue = (*recentKeyValue* == null) ? "" : *recentKeyValue*;  
 this.recentKeyResult = *recentKeyResult*;  
 this.isBinarySearchLastUsed = *isBinarySearchLastUsed*;  
 uniqueNames = *compileUniqueFirstNames*();  
 totalNumberOfContacts = *contactRecords*.*size*();  
 minimumID = *getMinimumID*();  
 maximumID = *getMaximumID*();  
 }  
  
 public void *summarizeStats*() {  
 *System*.out.*println*("\n" + "═".*repeat*(50));  
 *System*.out.*println*("📊 CONTACT RECORDS STATISTICS");  
 *System*.out.*println*("═".*repeat*(50));  
 *System*.out.*printf*("Total Contacts : %d%n", totalNumberOfContacts);  
 *System*.out.*printf*("Sorted by ID : %b%n", *isSorted*());  
 *System*.out.*printf*("Minimum ID : %d%n", minimumID);  
 *System*.out.*printf*("Maximum ID : %d%n", maximumID);  
 *System*.out.*printf*("Linear Searches Done : %d%n", linearSearchCount);  
 *System*.out.*printf*("Binary Searches Done : %d%n", binarySearchCount);  
 *System*.out.*printf*("Last Search Result : %s (%s)%n", recentKeyResult, recentKeyValue);  
 *System*.out.*println*("─".*repeat*(50));  
 }  
  
 public void *menu*() {  
 while (true) {  
 *System*.out.*println*("\n[0] 🔙 Go Back");  
 *System*.out.*println*("[1] 🧾 Unique Names & Frequency");  
 *System*.out.*print*("Enter choice >> ");  
 int choice = *Integer*.*parseInt*(in.*nextLine*());  
 switch (choice) {  
 case 0 -> {  
 return;  
 }  
 case 1 -> *displayNameStats*();  
 default -> *System*.out.*println*("⚠️ Invalid option. Try again.");  
 }  
 }  
 }  
  
 public void *trackMetrics*() {  
 *System*.out.*println*("\n" + "═".*repeat*(50));  
 *System*.out.*println*("⚙️ SEARCH METRICS");  
 *System*.out.*println*("═".*repeat*(50));  
 if (isBinarySearchLastUsed) {  
 *System*.out.*println*("Algorithm Used : Binary Search");  
 *System*.out.*println*("Reason : Best for sorted dataset, and is efficient.");  
 *System*.out.*println*("Comparisons Made : " + *Main*.binaryComparisonCount);  
 *System*.out.*println*("Execution Time : " + *Main*.binarySearchTime + " ns");  
 } else {  
 *System*.out.*println*("Algorithm Used : Linear Search");  
 *System*.out.*println*("Reason : Names are unsorted.");  
 *System*.out.*println*("Comparisons Made : " + *Main*.linearComparisonCount);  
 *System*.out.*println*("Execution Time : " + *Main*.linearSearchTime + " ns");  
 }  
 *System*.out.*println*("Found Index : " + (*Main*.lastFoundIndex == -1 ? "N/A" : *Main*.lastFoundIndex));  
 *System*.out.*println*("Found Value : " + (*Main*.lastFoundValue.*isEmpty*() ? "N/A" : *Main*.lastFoundValue));  
 *System*.out.*println*("─".*repeat*(50));  
 }  
  
 private void *displayNameStats*() {  
 *System*.out.*println*("\n" + "═".*repeat*(50));  
 *System*.out.*println*("🧾 UNIQUE NAMES & FREQUENCY");  
 *System*.out.*println*("═".*repeat*(50));  
 *ArrayList*<*String*> uniqueNames = *compileUniqueFirstNames*();  
 *ArrayList*<*Integer*> nameCounts = *countFirstNameFrequency*();  
 *bubbleSort*(uniqueNames, nameCounts);  
 for (int i = 0; i < uniqueNames.*size*(); i++) {  
 *System*.out.*printf*("%-3d %-15s (%dx)%n", i + 1, uniqueNames.*get*(i), nameCounts.*get*(i));  
 }  
 *System*.out.*println*("─".*repeat*(50));  
 }  
  
 private *ArrayList*<*Integer*> *countFirstNameFrequency*() {  
 *ArrayList*<*Integer*> nameCountList = new *ArrayList*<>();  
  
 for (*String* name : uniqueNames) {  
 int count = 0;  
 for (*ArrayList*<*String*> contactRecord : contactRecords) {  
 *String* fullName = contactRecord.*get*(1);  
 int spaceIndex = fullName.*indexOf*(' ');  
 *String* firstName = (spaceIndex != -1) ? fullName.*substring*(0, spaceIndex) : fullName;  
  
 if (firstName.*equals*(name)) {  
 count++;  
 }  
 }  
 nameCountList.*add*(count);  
 }  
  
 return nameCountList;  
 }  
  
 private *ArrayList*<*String*> *compileUniqueFirstNames*() {  
 *ArrayList*<*String*> uniqueNamesList = new *ArrayList*<>();  
  
 for (*ArrayList*<*String*> contactRecord : contactRecords) {  
 *String* fullName = contactRecord.*get*(1);  
 int spaceIndex = fullName.*indexOf*(' ');  
 *String* firstName = (spaceIndex != -1) ? fullName.*substring*(0, spaceIndex) : fullName;  
  
 if (!uniqueNamesList.*contains*(firstName)) {  
 uniqueNamesList.*add*(firstName);  
 }  
 }  
  
 return uniqueNamesList;  
 }  
  
 private boolean *isSorted*() {  
 for (int i = 0; i < contactRecords.*size*() - 1; i++) {  
 if (*Integer*.*parseInt*(contactRecords.*get*(i).*getFirst*()) > *Integer*.*parseInt*(contactRecords.*get*(i + 1).*getFirst*())) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 private int *getMinimumID*() {  
 int minimumID = *Integer*.*parseInt*(contactRecords.*getFirst*().*getFirst*());  
 for (int i = 1; i < contactRecords.*size*(); i++) {  
 int currentIDIndex = *Integer*.*parseInt*(contactRecords.*get*(i).*getFirst*());  
 if (currentIDIndex < minimumID) {  
 minimumID = currentIDIndex;  
 }  
 }  
 return minimumID;  
 }  
  
 private int *getMaximumID*() {  
 int maximumID = *Integer*.*parseInt*(contactRecords.*getFirst*().*getFirst*());  
 for (int i = 1; i < contactRecords.*size*(); i++) {  
 int currentIDIndex = *Integer*.*parseInt*(contactRecords.*get*(i).*getFirst*());  
 if (currentIDIndex > maximumID) {  
 maximumID = currentIDIndex;  
 }  
 }  
 return maximumID;  
 }  
  
 private void *bubbleSort*(*ArrayList*<*String*> *unsortedArray*, *ArrayList*<*Integer*> *counts*) {  
 for (int i = 0; i < *unsortedArray*.*size*() - 1; i++) {  
 for (int j = 0; j < *unsortedArray*.*size*() - 1; j++) {  
 int firstLetter = *unsortedArray*.*get*(j).*charAt*(0);  
 int secondLetter = *unsortedArray*.*get*(j + 1).*charAt*(0);  
  
 if (firstLetter > secondLetter) { *//Swap if out of order  
 String* temp = *unsortedArray*.*get*(j);  
 *unsortedArray*.*set*(j, *unsortedArray*.*get*(j + 1));  
 *unsortedArray*.*set*(j + 1, temp);  
  
 int tempCount = *counts*.*get*(j);  
 *counts*.*set*(j, *counts*.*get*(j + 1));  
 *counts*.*set*(j + 1, tempCount);  
 }  
 }  
 }  
 }  
}