Topics in Computer Science
Searching

Searching an array

Linear Search

A searching algorithm that can locate an entry in a list of elements

An algorithm that can be very easily implemented

Effective and efficient for relatively small lists, but inefficient for large lists with lots of data

Searching an array

Linear Search

Suppose we are looking for `51` in the array below. Let's call it `list`

15
21
22
25
27
29
33
41
42
44
49
51
60

Searching an array

Linear Search

We compare `51` with the first element of `list`

If `51` is not the first element, we proceed to the next element always checking that we have not reached the end of the list

Searching an array

Linear Search

We now compare `51` with the second element of `list`

15
21
22
25
27
29
33
41
42
44
49
51
60

`51` is has not been found. We continue to move through the list until we either find the `51` or we have reached the end of the list.

Searching an array

Linear Search

As we keep comparing...

51 has been located with 12 comparisons.

Exercise 1

a) Do a hand trace of the linear search algorithm as we try to locate the number `39` in each of these arrays:

i)

2
5
7
10
15
22
28
32
37
39
41
45
50

ii)

39
45
50
75
123
140
148
160
225
327
339
400
800

iii)

1
2
3
4
5
6
7
8
9
10
11
12
13

Exercise 2

In your class `SearchAndSort` include a Java method with the signature

```
public static int linearSearch(int[] list, int key)
```

that finds `key` in `list` using a linear search. Test your method on a small array

(Algorithm is on the next slide)

Exercise 2 (Cont'd)

Algorithm

```
set i = index of first element of list

while i is within the bounds of list and key has not been found {
    increase i by 1
}
if i has gone beyond end of list
    key is not in list
else
    i is now pointing to the element we want
```

Exercise 2 - Continued

Write a Java method with the signature

```
public static int linearSearch(String[] list, String key)
```

that finds `key` in `list` using a linear search

Exercise 3

Test your linear search method on the following arrays when looking for `key = 28`

i)

2
5
7
10
15
22
28
32
37
39
41
45
50

ii)

39
45
50
75
123
140
148
160
225
327
339
400
800

iii)

1
2
3
4
5
6
7
8
9
10
11
12
13

Exercise 4

Generate an array of 500 random numbers, then perform a linear search on the array with keys of your choice. Include a variable in your program that counts the number of comparisons

Run a few more tests with arrays of larger sizes to get a sense of the number of comparisons necessary to find a key in the array

Searching

Linear Search Order of Complexity

List size	Comparisons (worst-case)
1	1
2	2
3	3
4	4
5	5
6	6
…	…
n	n

Complexity of Linear Search algorithm is O(n)

Searching an array

Binary Search

A searching algorithm that can locate an entry in an **ordered** list of elements in an extremely efficient way

An example of a large class of algorithms called **divide and conquer** that use strategies to solve a problem by quickly reducing its size

At each stage of the solution, Binary Search cuts the size of the problem in roughly half

Searching an array

Binary Search

To illustrate its extreme efficiency, Binary Search can locate an element in an array of:

16,000,000 elements with 24 comparisons or less

32,000,000 elements with 25 comparisons or less

64,000,000 elements with 26 comparisons or less

1.71×10^{10} elements with 34 comparisons or less

6.022×10^{23} elements with 79 comparisons or less

Searching an array

Binary Search

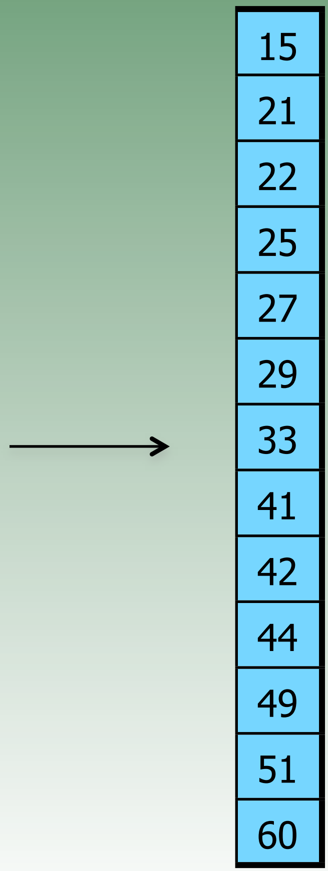
Suppose we are looking for `51` in the list below. Let's call it $\texttt{list}_1$

15
21
22
25
27
29
33
41
42
44
49
51
60

Searching an array

Binary Search

We compare `51` with the middle element of `list`$_1$



Since `51` is larger than `33`, then the element must be in the bottom half of `list`$_1$

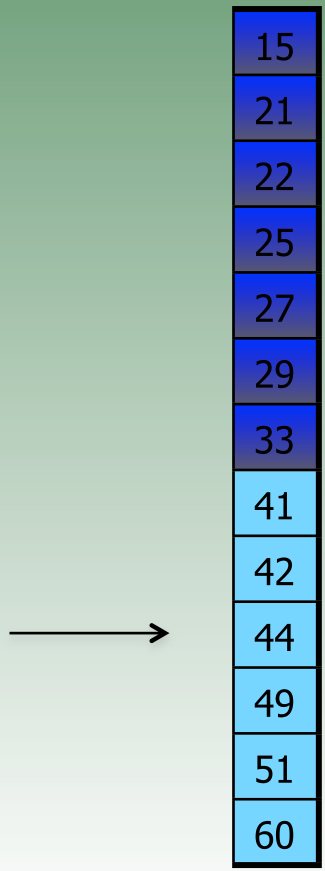
We can discard the entire top half of `list`$_1$ and repeat the process

What remains is `list`$_2$

Searching an array

Binary Search

We now compare 51 with the middle element of $\texttt{list}_2$



Since 51 is larger than 44, then the element must be in the bottom half of $\texttt{list}_2$

We can discard the entire top half of $\texttt{list}_2$ and repeat the process

What remains is $\texttt{list}_3$

Searching an array

Binary Search

We now compare `51` with the middle element of $\texttt{list}_3$

`51` has been located with 3 comparisons

Exercise 5

The previous slides show how the list gets cut in half and becomes smaller and smaller as the algorithm progresses.

Show the same process for each of these arrays when the number we are looking for is `39`

i)

2
5
7
10
15
22
28
32
37
39
41
45
50

ii)

39
45
50
75
123
140
148
160
225
327
339
400
800

iii)

1
2
3
4
5
6
7
8
9
10
11
12
13

Exercise 6

Write a Java method with the signature

```
public static int binarySearch(int[] list, int key)
```

that finds `key` in `list` using a binary search. Test your method on a small array

Algorithm is on the next slide

Exercise 6 (Cont'd)

Algorithm

```
set start = index of first element of list
set end   = index of last element of list
set idx   = middle index between start and end

while key has not been found and start is not larger than end {
    if key is larger than the element that idx points to in list
        set start = the next index after idx
    else
        set end = the previous index before idx

    set idx = middle index between start and end
}
if start has gone beyond end
    element is not in list
else
    idx is now pointing to element in list
```

Exercise 6 - Continued

Write a Java method with the signature

```
public static int binarySearch(String[] list, String key)
```

that finds `key` in `list` using a binary search

Exercise 7

Test your binary search method on the following arrays when looking for `key = 28`

i)

2
5
7
10
15
22
28
32
37
39
41
45
50

ii)

39
45
50
75
123
140
148
160
225
327
339
400
800

iii)

1
2
3
4
5
6
7
8
9
10
11
12
13

Exercise 8

Generate an array of 2000 random numbers, sort the array, then perform a binary search on the array with keys of your choice. Include a variable in your program that counts the number of comparisons

Run a few more tests with arrays of larger sizes to get a sense of the small number of comparisons necessary to find a key in the array

Exercise 9

Do a hand trace of the binary search method. Keep track of all variables and parameters, and how they change as the method executes.

Use the following to do your trace:

Use an array of integers called `primes`, which is sorted ascending, and containing ten numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29

Trace the binary search method with these two calls:

```
System.out.println(binarySearch(primes, 23));
System.out.println(binarySearch(primes, 9));
```

Searching

Binary Search Order of Complexity - Logarithms

In Binary Search, the logarithm in base 2 of a number is needed in order to calculate the maximum number of comparisons that Binary Search performs on a sorted array when trying to find an element

The logarithm in base 2 of n $\log_2(n)$ is the exponent to which 2 needs to be raised in order to obtain n

For example:

$\log_2(64) = 6$ because $2^6 = 64$
$\log_2(512) = 9$ because $2^9 = 512$
$\log_2(1024) = 10$ because $2^{10} = 1024$

Searching

Binary Search Order of Complexity - Logarithms

When the numbers get larger or when they are not exact powers of 2, we need to use a formula and a calculator to compute the logarithm in base 2 of a number:

$$\log_2 n = \frac{\log n}{\log 2}$$

Use a calculator

For example: $$\log_2(128) = \frac{\log(128)}{\log(2)} = 7$$

Use a calculator

$$\log_2(23701) = \frac{\log(23701)}{\log(2)} = 14.53266$$

Searching

Binary Search Order of Complexity

This algorithm requires finding an element in a sorted array by cutting the list in half after each comparison (assume element is in the list)

List size	Comparisons (worst-case)	Comparisons
1	1	1
2	1 to cut list in half + 1 for a list of 1 element	2
4	1 to cut list in half + 2 for a list of 2 elements	3
8	1 to cut list in half + 3 for a list of 4 elements	4
16	1 to cut list in half + 4 for a list of 8 elements	5
32	1 to cut list in half + 5 for a list of 16 elements	6
…	…	…
n	1 to cut list in half + $\log_2(n)$ for a list of (n / 2) elements	$\log_2(n) + 1$

Searching

Binary Search Order of Complexity

In the previous examples, the list length is always a power of two. Most of the time our arrays will not have a length that is a power of two. In those cases we need to floor the logarithm,

Number of comparisons = floor($\log_2$(list size)) + 1

For example, to calculate the maximum number of comparisons needed to find an element in a sorted array of 13,345 elements,

$$\lfloor \log_2(13435) \rfloor + 1 = \left\lfloor \frac{\log 13435}{\log 2} \right\rfloor + 1 = \lfloor 13.7137 \rfloor + 1 = 13 + 1 = 14$$

Therefore, the element can be found in 14 comparisons or less

Searching

Binary Search Order of Complexity

Another example, to calculate the maximum number of comparisons needed to find an element in a sorted array of 425,779 elements,

$$\lfloor \log_2(425779) \rfloor + 1 = \left\lfloor \frac{\log 425779}{\log 2} \right\rfloor + 1 = \lfloor 18.6997 \rfloor + 1 = 18 + 1 = 19$$

The element can be found in 19 comparisons or less

The number of comparisons performed is directly proportional to the logarithm of the number of elements with the base having a negligible effect

Complexity of Binary Search algorithm is O(log(n))

Exercise 10

Complexity of binary search: Write a table that relates the number comparisons with respect to the size of the array. Estimate how many comparisons `binarySearch` would take when searching for an item in a sorted list that contains the item we are looking for, if the list

a) Is empty

b) Contains 1 element

c) Contains 2 elements

d) Contains 4 elements

e) Contains 20 elements

f) Contains 80 elements

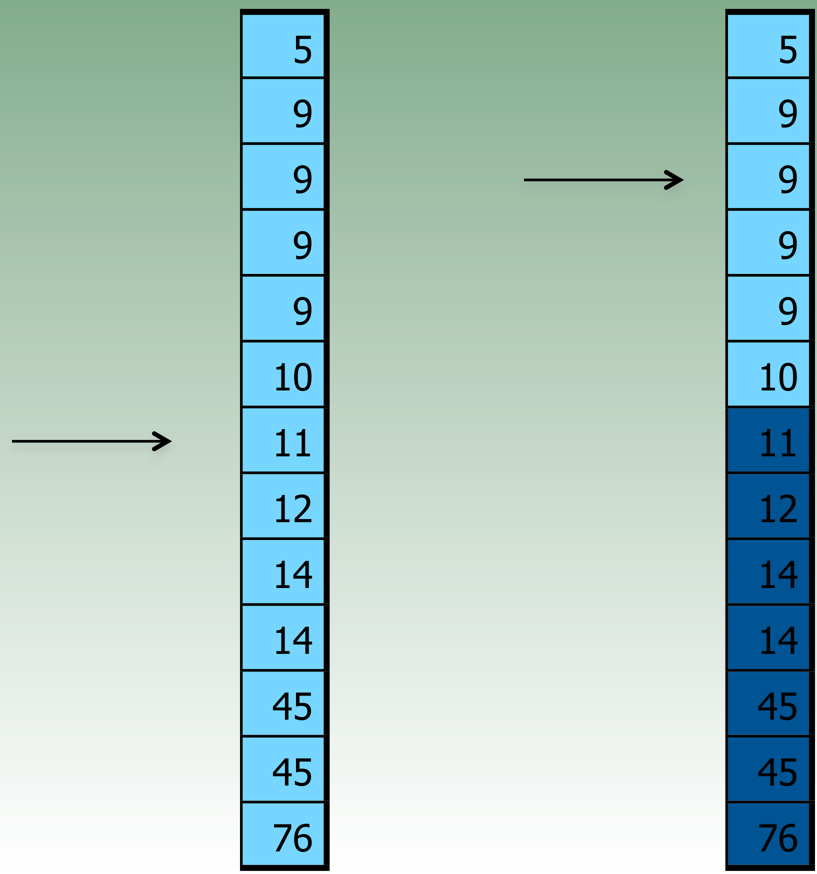
g) Contains 1,000 elements

h) Contains `n` elements

Exercise 11

The binary search algorithm explained so far will work very well for arrays with unique entries. If an array contains repeated entries, the algorithm will return the first match, not the first occurrence.

For example, given the array below, the call `binarySearch(items, 9)` will return `2` since `9` is the first match found at index location `2`. Since binary search does not use a linear strategy, the number located is not necessarily the first occurrence in the array



Next slide

Exercise 11 - Continued

Write a Java program with the signature

```
public static int findFirst(int[] items, int key)
```

that efficiently finds the first occurrence of `key` in the array `items`

The method must contain both a binary search and a linear search to accomplish this task

The method locates an occurrence of `key` in `items` by using a binary search. Then, the method uses a linear search to scan backwards from the found position and locate the first occurrence of `key`

Be careful with out of bounds errors which might happen if the first occurrence happens to be the first element of the array

See next slide for an example

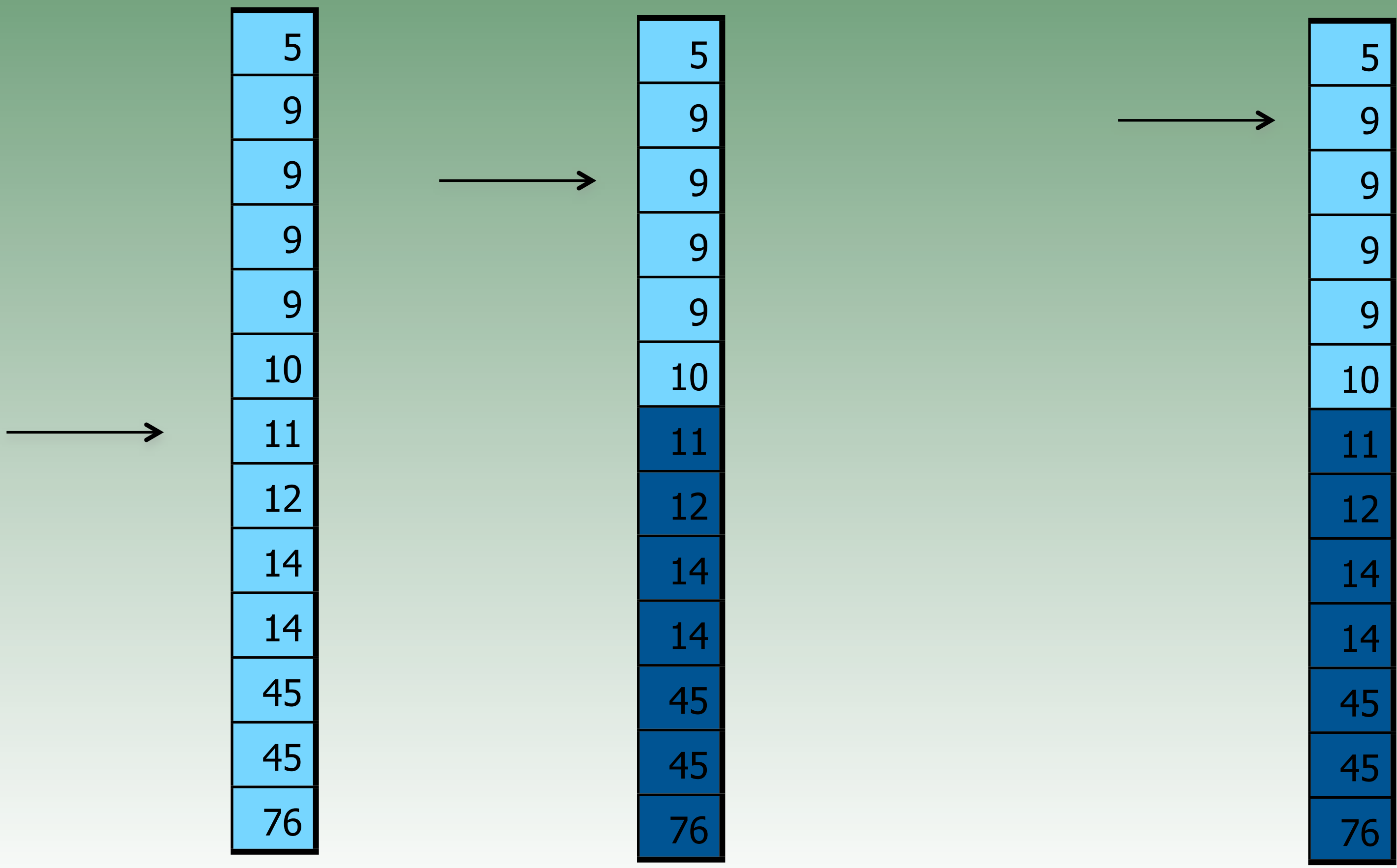
Next slide

Exercise 11 - Continued

The call `findFirst(items, 9)` would cause the following:

binary search is used to find an occurrence of `9`

linear search is then used to scan backwards and find the first occurrence of `9`



Next slide

Exercise 11 - Continued

Use the following to test your method

items	call	returns
{2,2,2,2}	findFirst(items, 2)	0
{2,2,2,2,3}	findFirst(items, 2)	0
{2,2,2,2,3}	findFirst(items, 3)	4
{2,2,2,2,3,3,3,3,3,3}	findFirst(items, 3)	4
{3,4,4,4,5,5,5,5,6,7,8}	findFirst(items, 4)	1
{3,4,4,5,6,7,7,7,7}	findFirst(items, 7)	5
{3,4,4,5,6,7,7,7,7}	findFirst(items, 5)	3
{3,4,4,5,6,7,7,7,7}	findFirst(items, 1)	-1
{3,4,4,5,6,7,7,7,7}	findFirst(items, 10)	-1
{5}	findFirst(items, 5)	0
{5}	findFirst(items, 8)	-1

Exercise 12

Write a Java method with the signature

```
public static int findFirst(String[] items, String key)
```

that finds the index location of the first entry in the sorted array `items` that begins with `key`

For example, if `key` has the value `co`, then a possible first entry might be the word `coal`

Use a combination of binary search and linear search to write this method

Binary search will need to be modified so that the comparisons are made between `key` and a substring of `items[i]`; the substring will be from position `0` to position `key.length() - 1`

Exercise 13

Write a Java method with the signature

```
public static String[] findAll(String[] items, String key)
```

that returns all the entries in the sorted array `items` that start with `key`

For example, in an array of names, if `key` has the value `Ad`, then the method returns all names that start with `Ad` such as Adam, Adams, Adele, Adrian, Adward.

For now, to make the method simpler, you may assume that a maximum of 50 entries will be returned. This way you can declare an array of 50 strings inside the method and return it. If no matching entries are found, the method returns a `null`

Use a combination of binary search and linear search to write this method

Next slide

Exercise 13 - Continued

algorithm

```
declare resultList as array of string without instantiating
let resultList = null

let index = find the first entry that starts with key (binary search)

if index != -1 {
    instantiate resultList as an array of 50 elements

    let i = 0
    while items[index] starts with key {     // linear scan
        resultList[i] = items[index]
        increment i
        increment index
    }
}

return resultList
```

Exercise 14

a) Given a sorted non-empty array of 2 elements or more, when will a linear search perform the same number of comparisons as a binary search?

b) Graph the functions $y = x$ and $y = \log_2 x + 1$ and find the points of intersection. Use these to support your answer of part a)

c) Solve the equation $x = \log_2 x + 1$ to support your answers of a) and b)

Exercise 15

Calculate the following:

a) The maximum number of comparisons needed to locate an element in an array of 36,000 elements. Assume the element is in the array.

b) The largest possible array so that any element can be located with no more than 8 lookups.

c) The smallest array that will require a maximum of 8 comparisons to locate an element in the array.

d) All the possible sizes that an array could be if exactly 10 comparisons are required to locate an element in the array. The answer should be a range such as: "The array could be from 100 to 200 elements in size"

e) The maximum number of comparisons needed to realize that the element is not in an array of size n

Searching Multiple Arrays

Consider the following request:

```
list all the courses that student 733822811 is taking and print
the course codes and course descriptions
```

We need a set up as follows

studentsAndCourses

733822811	CSC148,MAT223,PHY150,CHM292
432011922	MUS305,HIS378,ENG140
732392194	ENG140,PSY100,CHM108
531118220	CSC148,PHY150

courses

CSC148	Computer Science
MAT223	Linear Algebra
PHY150	Theoretical Physics
MUS305	Music Performance III
ENG140	English Literature I
HIS378	19th Century History
PSY100	Introduction to Psychology

Searching Multiple Arrays

To process the request, we need to scan the first table and, as soon as we find a matching student number we take the course numbers and scan the second table. Once the course is found in the second table we can then print the required information. If a course cannot be found in the `courses` table, we print a N/A (not available) message

studentsAndCourses

733822811	CSC148,MAT223,PHY150,CHM292
432011922	MUS305,HIS378,ENG140
732392194	ENG140,PSY100,CHM108
531118220	CSC148,PHY150

courses

CSC148	Computer Science
MAT223	Linear Algebra
PHY150	Theoretical Physics
MUS305	Music Performance III
ENG140	English Literature I
HIS378	19th Century History
PSY100	Introduction to Psychology

Searching Multiple Arrays

The ouput of the given request is

```
733822811
     CSC148 Computer Science
     MAT223 Linear Algebra
     PHY150 Theoretical Physics
     CHM292 N/A
```

studentsAndCourses

733822811	CSC148,MAT223,PHY150,CHM292
432011922	MUS305,HIS378,ENG140
732392194	ENG140,PSY100,CHM108
531118220	CSC148,PHY150

courses

CSC148	Computer Science
MAT223	Linear Algebra
PHY150	Theoretical Physics
MUS305	Music Performance III
ENG140	English Literature I
HIS378	19th Century History
PSY100	Introduction to Psychology

Exercise 16a

Include in your `SearchAndSort` class a Java method with the signature

```
public static void printCourseInfo(String studentNumber)
```

that implements the searching as explained in the previous slides. The method accepts a student number and prints out all the courses that the student is taking along with the course descriptions. Include a short `main()` method that asks for the student number from the user.

The arrays are not sorted so we need to use sequential searches in this method.

Next slide

Exercise 16a (Cont'd)

The `studentsAndCourses` array contains student numbers and the courses that the students are taking. The courses are separated by commas.

The arrays are not passed to the method as parameters. Instead we make them accessible by `static` declaration prior to the method. For example,

```
public static String[][] studentsAndCourses =
    {{"914454747", "CSC148,PHY150"},
     {"875733620", "ENG100,MUS101"}};
```

Here is a sample run,

```
Enter student number: 733822811

733822811
        CSC148 Computer Science
        MAT223 Linear Algebra
        PHY150 Theoretical Physics
        CHM292 N/A
```

Exercise 16b

Modify the method of part 16a so that the user can enter the starting characters of a student number and the method generates a report for all the students whose number starts with the provided information. For example:

```
Enter student number: 73

733822811
       CSC148 Computer Science
       MAT223 Linear Algebra
       PHY150 Theoretical Physics
       CHM292 N/A

732392194
       ENG140 English Literature I
       PSY100 Introduction to Psychology
       CHM108 N/A
```

Exercise 16c

Modify the method of part 16c so that the user can enter an asterisk as input. This means that a full report for all students is to be printed.

```
Enter student number: *

733822811
        CSC148 Computer Science
        MAT223 Linear Algebra
        PHY150 Theoretical Physics
        CHM292 N/A

432011922
        MUS305 Music Performance III
        HIS378 19th Century History
        ENG140 English Literature I

531118220
        CSC148 Computer Science
        PHY150 Theoretical Physics

732392194
        ENG140 English Literature I
        PSY100 Introduction to Psychology
        CHM108 N/A
```

Exercise 16d

Add the following table to your class of the previous exercise

students

733822811	Donald	Philip	Science
432011922	Johnston	Donna	Humanities
732392194	Peters	Susan	Science
531118220	Cook	Ann	Life Sciences

Modify the method so that the student information is also printed. For example

```
Enter student number: 733822811

733822811 Donald, Philip. Program: Science
        CSC148 Computer Science
        MAT223 Linear Algebra
        PHY150 Theoretical Physics
        CHM292 N/A
```

Exercise 16e

Replace the data arrays of your program with the data arrays in the `Student Information` link.

Run your program with 3 test cases and save the results:

1) For a single student
2) For a group of students whose student numbers start with the same set of digits (choose however many digits you want)
3) For a full report

Save each test case in text files under the filenames:

`<your first name>1.data` (for example: `Peter1.data`)

`<your first name>2.data`

`<your first name>3.data`

Searching

Complexity of Multiple Array Sequential Search

We look at arrays where only one item is in each array location. Then, one item can be found in `studentsAndCourses` in O(n) time.

Once the item is found, the corresponding item in `courses` can be found in O(m) time.

studentsAndCourses

733822811	CSC148
432011922	MUS305
732392194	ENG140
531118220	CSC148

courses

CSC148	Computer Science
MAT223	Linear Algebra
PHY150	Theoretical Physics
MUS305	Music Performance III
ENG140	English Literature I
HIS378	19th Century History
PSY100	Introduction to Psychology

Searching

Complexity of Multiple Array Sequential Search

Searching for `732392194` causes one linear scan of the `studentsAndCourses` array and one linear scans of the `course` array

studentsAndCourses

733822811	CSC148
432011922	MUS305
732392194	ENG140
531118220	CSC148

courses

CSC148	Computer Science
MAT223	Linear Algebra
PHY150	Theoretical Physics
MUS305	Music Performance III
ENG140	English Literature I
HIS378	19th Century History
PSY100	Introduction to Psychology

Searching

Complexity of Multiple Array Sequential Search

In this particular case, we perform 3 + 5 comparisons approximately. In the worst-case scenario, both tables are of equal size. The number of comparisons would be n + n

studentsAndCourses

733822811	CSC148
432011922	MUS305
732392194	ENG140
531118220	CSC148

courses

CSC148	Computer Science
MAT223	Linear Algebra
PHY150	Theoretical Physics
MUS305	Music Performance III
ENG140	English Literature I
HIS378	19th Century History
PSY100	Introduction to Psychology

Searching

Complexity of Multiple Array Sequential Search (Two arrays)

List sizes	Comparisons (worst-case)	Comparisons
1	1 + 1	2
2	2 + 2	4
3	3 + 3	6
4	4 + 4	8
5	5 + 5	10
…	…	…
n	n + n	2n

Complexity of Multiple Array Sequential Search with two arrays is O(n)

Exercise 17

Determine the complexity of a multiple array sequential search with three arrays of size n each

List sizes	Comparisons (worst-case)	Comparisons
1		
2		
3		
4		
5		
…		
n		

Exercise 18

Determine the complexity of a multiple array sequential search with n arrays of size n each

List sizes	Comparisons (worst-case)	Comparisons
1		
2		
3		
4		
5		
...		
n		

Exercise 19

Determine the complexity of generating a report for all the students, their courses and all other information that could be linked through n tables. Use the `studentsAndCourses` array that contains only one course per student.

Exercise 20

Make a small modification to your `printCourseInfo()` method so that no printing happens. Instead have a variable count the number of times that printing would happen. Use a global counting variable.

Then, write a `main()` method that makes many, many requests to the `printCourseInfo()` method so that there is a significant waiting time (about 4 seconds or more) before the program is finished (use dummy loops if you have to).

Print the count at the very, very end of the program. We are trying to get a sense of the time it takes to perform many requests when we use linear search.

Exercise 21

Perform a sort on the arrays of exercises 16a-b, (16a-e TOPS) then take the resulting arrays and replace the original ones with the sorted ones.

For `studentsAndCourses` array, sort by student number
For `courses` array, sort by course code
For `students` array, sort by student number

We will need the sorted arrays for binary search. We do not want to sort every time we run a binary search. Therefore the arrays <u>must be sorted</u> prior to using binary search as we will do in the next set of exercises.

Exercise 22

<under construction>

Exercise 23

Run the program of exercise 21 with a binary search and compare the times of both programs. Run a few comparisons with different amount of requests and check that the complexity of binary search is O(log n) and the complexity of linear search is O(n)

Selected Solutions

Solution to Exercise 6

```
public static int binarySearch (int[] list, int key) {
    int bottom = 0;
    int top = list.length - 1;
    int index = (bottom + top) / 2;

    while (list [index] != key && bottom <= top) {
        if (key > list[index])
            bottom = index + 1;
        else
            top = index - 1;

        index = (bottom + top) / 2;
    }

    if (bottom > top)
        return -1;
    else
        return index;
}
```

Solution to Exercise 12

```
public static int findFirst(String[] items, String key) {
    int index = binarySearchStartsWith(items, key);
    if (index != -1) {
        while (index >= 0 && items[index].startsWith(key)) { // short evaluation
            index--;
        }
        index++;   // index had gone one element beyond; now points to correct one
    }
    return index;
}
```

See next slide for code of method `binarySearchStartsWith()`

Solution to Exercise 12 - Continued

```
private static int binarySearchStartsWith (String[] list, String key) {
    int bottom = 0;
    int top = list.length - 1;
    int index = (bottom + top) / 2;

    while (!list[index].startsWith(key) && bottom <= top) {
        if (key.compareTo(list[index]) > 0)
            bottom = index + 1;
        else
            top = index - 1;

        index = (bottom + top) / 2;
    }

    if (bottom > top)
        return -1;
    else
        return index;
}
```

Solution to Exercise 13

```
public static String[] findAll(String[] items, String key) {
    final int RESULT_LIST_SIZE = 50;
    String[] resultList = null;

    int index = findFirst(items, key);
    if (index != -1) {
        resultList = new String[RESULT_LIST_SIZE];

        // initialize all elements of resultList with null for possible later use
        for (int i = 0; i < resultList.length; i++) {
            resultList[i] = null;
        }

        // populate resultList with all items that start with key
        for (int i = 0;
             i < RESULT_LIST_SIZE &&
             index < items.length &&
             items[index].startsWith(key); i++, index++) {
            resultList[i] = items[index];
        }
    }
    return resultList;
}
```