

# CSC 222: Computer Programming II

Spring 2005

## Searching and efficiency

- sequential search
- big-Oh, rate-of-growth
- binary search
- example: spell checker

1

## Searching a list

suppose you have a list, and want to find a particular item, e.g.,

- lookup a word in a dictionary
- find a number in the phone book
- locate a student's exam from a pile

searching is a common task in computing

- searching a database
- checking a login password
- lookup the value assigned to a variable in memory

if the items in the list are unordered (e.g., added at random)

- desired item is equally likely to be at any point in the list
- need to systematically search through the list, check each entry until found

→ sequential search

2

## Sequential search

sequential search traverses the list from beginning to end

- check each entry in the list
- if matches the desired entry, then FOUND (return its index)
- if traverse entire list and no match, then NOT FOUND (return -1)

recall: the `ArrayList` class has an `indexOf` method

```
/**
 * Performs sequential search on the array field named items
 * @param desired item to be searched for
 * @returns index where desired first occurs, -1 if not found
 */
public int indexOf(Object desired)
{
    for(int k=0; k < items.length; k++) {
        if (desired.equals(items[k])) {
            return k;
        }
    }
    return -1;
}
```

3

## How efficient is sequential search?

for this algorithm, the dominant factor in execution time is checking an item

- the number of checks will determine efficiency

in the worst case:

- the item you are looking for is in the last position of the list (or not found)
- requires traversing and checking every item in the list
  
- if 100 or 1,000 entries → NO BIG DEAL
- if 10,000 or 100,000 entries → NOTICEABLE

in the average case?

in the best case?

4

## Big-Oh notation

to represent an algorithm's performance in relation to the size of the problem, computer scientists use *Big-Oh* notation

an algorithm is  $O(N)$  if the number of operations required to solve a problem is proportional to the size of the problem

sequential search on a list of  $N$  items requires *roughly*  $N$  checks (+ other constants)  
→  $O(N)$

for an  $O(N)$  algorithm, doubling the size of the problem requires double the amount of work (in the worst case)

- if it takes 1 second to search a list of 1,000 items, then  
it takes 2 seconds to search a list of 2,000 items  
it takes 4 seconds to search a list of 4,000 items  
it takes 8 seconds to search a list of 8,000 items  
...

5

## Searching an ordered list

when the list is unordered, can't do any better than sequential search

- but, if the list is ordered, a better alternative exists

e.g., when looking up a word in the dictionary or name in the phone book

- can take ordering knowledge into account
- pick a spot – if too far in the list, then go backward; if not far enough, go forward

binary search algorithm

- check midpoint of the list
- if desired item is found there, then DONE
- if the item at midpoint comes after the desired item in the ordering scheme, then repeat the process on the left half
- if the item at midpoint comes before the desired item in the ordering scheme, then repeat the process on the right half

6

## Binary search

the Collections utility class contains a `binarySearch` method

- takes a List of Comparable items and the desired item  
(List is an interface that specifies basic list operations, ArrayList implements List)
- returns index of desired item if found,  $-(\text{insertion\_point}) - 1$  if not found

```
/**
 * Performs binary search on a sorted list.
 * @param items sorted list of Comparable items
 * @param desired item to be searched for
 * @returns index where desired first occurs,  $-(\text{insertion point})-1$  if not found
 */
public static int binarySearch(List<Comparable> items, Comparable desired)
{
    int left = 0; // initialize range where desired could be
    int right = items.length-1;

    while (left <= right) {
        int mid = (left+right)/2; // get midpoint value and compare
        int comparison = desired.compareTo(items[mid]);

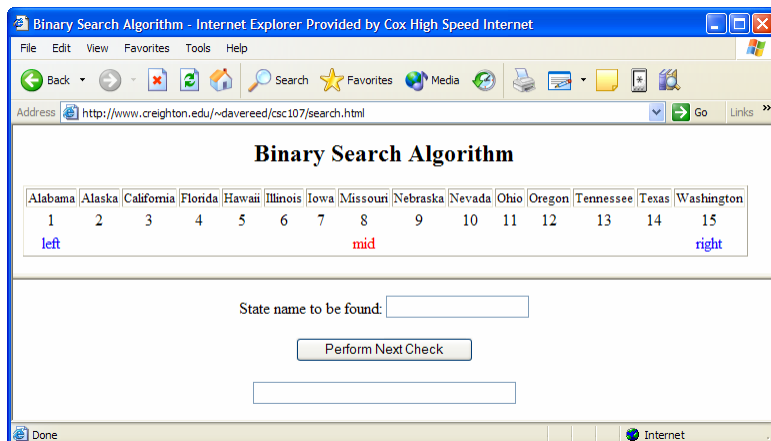
        if (comparison == 0) { // if desired at midpoint, then DONE
            return mid;
        }
        else if (comparison < 0) { // if less than midpoint, focus on left half
            right = mid-1;
        }
        else { // otherwise, focus on right half
            left = mid + 1;
        }
    }
    return /* CLASS EXERCISE */ ; // if reduce to empty range, NOT FOUND
}
```

7

## Visualizing binary search

note: each check reduces the range in which the item can be found by half

- see [www.creighton.edu/~davereed/csc107/search.html](http://www.creighton.edu/~davereed/csc107/search.html) for demo



8

## How efficient is binary search?

again, the dominant factor in execution time is checking an item

- the number of checks will determine efficiency

in the worst case:

- the item you are looking for is in the first or last position of the list (or not found)

start with N items in list

after 1<sup>st</sup> check, reduced to N/2 items to search

after 2<sup>nd</sup> check, reduced to N/4 items to search

after 3<sup>rd</sup> check, reduced to N/8 items to search

...

after  $\log_2 N$  checks, reduced to 1 item to search

in the average case?

in the best case?

9

## Big-Oh notation

an algorithm is  $O(\log N)$  if the number of operations required to solve a problem is proportional to the logarithm of the size of the problem

binary search on a list of N items requires *roughly*  $\log_2 N$  checks (+ other constants)

→  $O(\log N)$

for an  $O(\log N)$  algorithm, doubling the size of the problem adds only a constant amount of work

- if it takes 1 second to search a list of 1,000 items, then  
searching a list of 2,000 items will take time to check midpoint + 1 second  
searching a list of 4,000 items will take time for 2 checks + 1 second  
searching a list of 8,000 items will take time for 3 checks + 1 second

...

10

## Comparison: searching a phone book

Number of entries in phone book	Number of checks performed by sequential search	Number of checks performed by binary search
100	100	7
200	200	8
400	400	9
800	800	10
1,600	1,600	11
...	...	...
10,000	10,000	14
20,000	20,000	15
40,000	40,000	16
...	...	...
1,000,000	1,000,000	20

to search a phone book of the United States (~280 million) using binary search?

to search a phone book of the world (6 billion) using binary search?

11

## Searching example

for small  $N$ , the difference between  $O(N)$  and  $O(\log N)$  may be negligible

consider the following large-scale application: a spell checker

- want to read in and store a dictionary of words
- then, process a text file one word at a time
  - if word is not found in dictionary, report as misspelled
- since the dictionary file is large (117,777 words), the difference will be clear

we can define a `Dictionary` class to store & access words

- constructor, contains, size, displayAll, ...
- note: we can build a dictionary on top of an `ArrayList` of `Strings`  
`ArrayList` provides all the basic operations we require  
encapsulating into a `Dictionary` class allows us to hide later modifications

12

## Dictionary class

implement the Dictionary class with the specified methods

- for now, utilize sequential search (e.g., contains)

download [dictionary.txt](#) from the class directory

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class Dictionary
{
    private ArrayList<String> dict;

    public Dictionary()
    {
        // CONSTRUCTS EMPTY DICTIONARY
    }

    public Dictionary(String dictFile)
    {
        // CONSTRUCT DICTIONARY WITH WORDS FROM dictFile
    }

    public boolean contains(String word)
    {
        // RETURNS TRUE IF word IS STORED IN DICTIONARY
    }

    public int size()
    {
        // RETURNS NUMBER OF WORDS IN DICTIONARY
    }

    public void display()
    {
        // DISPLAYS ALL WORDS IN DICTIONARY, ONE PER LINE
    }
}
```

13

## Spell checker

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import javax.swing.JOptionPane;

public class SpellChecker
{
    public static void main(String[] args)
    {
        String dictFile = JOptionPane.showInputDialog("Dictionary file:");
        Dictionary dict = new Dictionary(dictFile);

        String textFile = JOptionPane.showInputDialog("Text file:");
        try {
            System.out.println("CHECKING...");
            Scanner infile = new Scanner(new File(textFile));
            while (infile.hasNext()) {
                String word = strip(infile.next());
                if (word.length() > 0 && !dict.contains(word)) {
                    System.out.println(word);
                }
            }
            infile.close();
            System.out.println("...DONE");
        } catch (FileNotFoundException exception) {
            System.out.println("NO SUCH FILE FOUND");
        }
    }

    private static String strip(String word)
    {
        // CLASS EXERCISE
    }
}
```

- stores words from a dictionary file
- processes a user-specified file, displaying each misspelled word
- note: need to strip non-letters and capitalization from file words before checking

14

## In-class exercise

copy the following files

- [www.creighton.edu/~davereed/csc222/Code/SpellChecker.java](http://www.creighton.edu/~davereed/csc222/Code/SpellChecker.java)
- [www.creighton.edu/~davereed/csc222/Code/gettysburg.txt](http://www.creighton.edu/~davereed/csc222/Code/gettysburg.txt)
- [www.creighton.edu/~davereed/csc222/Code/poe.txt](http://www.creighton.edu/~davereed/csc222/Code/poe.txt)

add to the Dictionary project and spell check the Gettysburg address

- is the delay noticeable?

how about for Poe's *The Cask of Amontillado*?

now download

[www.creighton.edu/~davereed/csc222/largeDictionary.txt](http://www.creighton.edu/~davereed/csc222/largeDictionary.txt)

and spell check the documents with that

- `largeDictionary.txt` is more than twice as big (264,063 words)
- does it take more than twice as long?

15

## Dictionary w/ binary search

since the dictionaries are in alphabetical order, we can use binary search

- modify the `Dictionary` class to import the `Collections` utility class

```
import java.util.Collections;
```

- modify the `contains` method so that it uses `Collections.binarySearch`

```
return (Collections.binarySearch(dict, word) >= 0);
```

now spell check the documents using binary search

- how much faster is it compared to sequential search
- does `largeDictionary.txt` take more than twice as long?

16