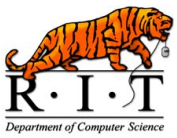

Introduction to Searching

or, Some ways to find the stuff you need...

Searching

- A common and important task, often performed on large sets of data
  - What does this imply for efficiency concerns?
- There are a number of different techniques for searching, including:
  - Serial (also called "linear") searches
  - Binary searches
  - Search by hashing

Looking at some approaches

- Sample problem:
  - We have a set of numbers we want to search for a given value.
  - For now, we'll assume that the information is stored in an array (for ease of coding)
- We'll use this problem as the basis for evaluating some efficiencies

Serial searching

- Algorithm is as follows:
  - Start at the beginning of the data
  - Walk through the data set one element at a time, looking for the specified value
  - The search stops when the value is found, or when we run out of data.

Challenge #1

- Write the code to do a serial search of an array for a value.

```
// Returns the index of the value, or –1 if not found
public int FindValue( int [] data, int value )
{




}
```


Serial search analysis

- What is the worst case time?
- What is the best case time?
- What is the average case time?

Binary searching

- Key assumption:
  - The data set must be in sorted order (typically expected to be smallest to largest)
- Can be done with numbered lists, Strings, Objects, or anything that can be sorted in some order

Binary search algorithm

- If the search value is equal to the value in the middle of the current data set, return it.
- If the search value is greater than the middle element, then search through the upper half of the data set.
- If the search value is less than the middle element, then search through the lower half.

Challenge #2:

- Write the code to do a binary search of an array for a value.

```
// Returns the index of the value, or –1 if not found
public int FindValueInSortedData( int [] data, int value )
{




}
```

- A sample solution to this challenge can be found in the "`BinarySearch.java`" sample file

Binary search analysis

- What is the worst case time?
- What is the best case time?
- What is the average case time?

- What if the data is stored in a linked list, as opposed to an array? Does anything change?

Other search alternatives

- Binary and serial searching are *generic* search algorithms, that will work with basically any type of data
- Other search algorithms exist, which are optimized for specific types of data, or usages
  - Searching for sub-strings within an existing string
  - Searching for patterns
  - etc.

Self-organizing search

- Principle:
  - An algorithm may be more efficient if some items in the list are searched for more frequently than others.
  - By "bubbling" a found item toward the front of the list, future searches for that item will be executed more quickly. If those items are nearer to the front of the list then search will be sped up considerably.
- Assumptions:
  - No set ordering requirement (i.e., any data can be put anywhere, and the collection's ordering may be changed)
  - Moving data around is "cheap"
  - Common characteristic of many data sets is that 80% of all operations are performed on 20% of the items in a data set

- [Outline procedure on the board, using linked lists, and shifting to the beginning of the list]

- Questions:
  - What is the performance of the reorganization of the data in the list?
  - What potential drawbacks are there to implementing this search for an array?
  - Can you see an alternative approach that would work well for arrays over time?