

Searching & Sorting Data

Contents

1	Overview of Searching & Sorting Data	2
2	Searching Data	7
2.1	Linear Search	8
2.2	Other Search Methods	9
3	Sorting Data	16

1 Overview of Searching & Sorting Data

- As we saw last week, arrays and vectors are common *data structures*, used to store sequences of information
- These data structures are used *heavily* in application
- There are also some common operations frequently done on these data structures. . .

- The first is *searching*...that is, if we have an array (or vector) of values, we'd like to search through the array and see if a specific value exists in the array
- Usually when we do this, we're also interested in finding out *where* in the array the element is located, if it is, in fact, in the array

- The second is *sorting*... that is, if we have an array of something like grades, we'd like to be able to sort them into increasing order

- These two problems, searching and sorting, are some of the most well documented and researched problems in computer science
- Each has a wealth of information online and in text books, which you will read through thoroughly next quarter and in data structures

- Now, we simply touch on them, and see how/why the two are closely related and often studied together

2 Searching Data

- To search a given array or vector, we'd like a function something like

```
bool searchList(vector<type> list, type element, int &location)
```

which scans the list provided and returns `true` if the value of `element` exists in the given list, and `false` otherwise.

- Further, if `element` is in the list, `location` is set to the location of the element in the list.

- There are many methods to search a given list, but two are the most common...

2.1 Linear Search

- The simplest method of searching a list is simply trying each element, in order, and seeing if that element of the list is equal to the element in question
- This is called *linear search*
- *examples...*
- Note that the book provides a different form, which is perfectly acceptable also

- Note that for a list of size n , linear search requires...
 - 1 step in *best case*
 - n steps in the *worst case*
 - $\frac{n}{2}$ steps in the *average case*

2.2 Other Search Methods

- Linear search certainly seems easy to implement and understand
- But consider the following...

- When you search through a phone book for someone's name, do you start at the very first name, and reach *each* and *every* name listed until you find the one you want?
- Probably not (well, I hope not)
- What can you say about what you do?

- Would that same approach work for searching arrays/vectors?
- Is there anything you have to assert about the array/vector you would like to employ this method to?

- When searching a phone book, we can skip half the names in the phone book with each guess...
- But the phone book is also *sorted*, listed the names in *ascending* order
- If we could assert that same thing about the array/vector we want to sort, we could employ this same search method, called *binary search*

- Note that for a list of size n , binary search requires...
 - 1 step in *best case*
 - $\log n$ steps in the *worst case* (Amazing!)
- That's because with each "guess", we eliminate *half* of the remaining values to be checked

- But this prompts us to ask the following question. . .
- If the list we want to search *isn't* sorted, what do we do?

- We sort it!

3 Sorting Data

- When sorting an array/vector, we'd like some function such as

```
void sort(vector<int> &list);
```

which takes a vector `list` and sorts it, *in-place*

- We could, of course, have the `sort` function return a copy of the vector passed which is sorted...
- Usually, however, we want to sort the data structure in-place

- One easy to understand method of sorting is called the *selection sort*
- It's not terribly efficient, but it is rather easy to follow

- *examples...*
- Note that §7.8 discusses insertion sort extensively