HST 952

Computing for Biomedical Scientists

Lecture 9

Outline

- Searching and sorting
- Programming examples

Searching and sorting

- To search a collection of numbers, characters, strings or any other typed that can be ordered
  - if the collection is not sorted, we may have to examine each member of the collection in turn until we find the item we are seeking (a *linear* or *sequential search* strategy)
  - if the collection is sorted we may be able to speed up or otherwise improve the efficiency of our search by taking advantage of this fact

Searching and sorting

- We will look briefly at some searching methods
  - linear/sequential search
  - binary search
- Since the process of searching a collection of numbers, etc. can be improved if we know that the items in the collection are already sorted we will look at some sorting methods:
  - selection sort
  - merge sort
- We will use arrays and vectors to illustrate the searching and sorting methods

Linear/sequential search

- If we have a completely filled array
  - start at the beginning of the array and proceed in sequence until either the value is found or the end of the array is reached
  - or, start at the end of the array and proceed backwards in sequence until the value being sought is found or the beginning of the array is reached
- If the array is only partially filled,
  - keep track of the number of valid entries in the array
  - forward search stops when the last meaningful value has been checked

Linear/sequential search

- Linear search is not the most efficient search method
- However, it works and is easy to program

```
public int itemInArray(char item, char[] arr, int validEntries)
    {
        int index = -1;

        for (int i = 0; i < validEntries; i++) {
            if (item == arr[i])
                index = i;
        }
        return index;
    }
```

Linear/sequential search

- In the worst possible case (when the element we are seeking is in the last position in the array or is not in the array at all), the total number of array item comparisons performed in a linear search equals the total length of the array being searched
- For an array with 2048 elements this means 2048 comparisons in the worst case

Binary search

- More efficient search method than linear search
- Works for arrays/collections that are already sorted (is essentially the strategy that humans use for searching a phone book or dictionary)
  - this strategy is often called a *divide-and-conquer* strategy
- Strategy for searching for a name in one section of a phone book is the same as initial strategy for searching for the name in the entire phone book
- This implies that we can solve the binary search problem using recursion

Binary search

```
Search:
    middle page = (first page + last page)/2
    Open the phone book to middle page;
    If (name is on middle page)
        then done;  //this is the base case
    else if (name is alphabetically before middle page)
        last page = middle page //redefine search area to front half
        Search //recursive call with reduced number of pages
    else //name must be after middle page
        first page = middle page //redefine search area to back half
        Search //recursive call with reduced number of pages
```

Binary search

- In the worst possible case (e.g., when the element we are seeking is not in the array at all), the total number of array item comparisons performed in a binary search is significantly less than for linear search
- For an array with 2048 elements a binary search would perform 11 comparisons in the worst case (compared to 2048 for linear search!)

Binary search

- Programming example

Selection sort

To sort an array, A of integers in non-decreasing order:

- search the array for the smallest number and record its index
- swap (interchange) the smallest number with the first element of the array
  - the sorted part of the array is now the first element: A[0]
  - the unsorted part of the array is the remaining elements: A[1]- A[length-1]
- search A[1]- A[length-1] for the next smallest element and record that element's index
- swap the next smallest element with the second element of the array
- repeat the search and swap until all elements have been placed
  - each iteration of the search/swap process increases the length of the sorted part of the array by one, and reduces the unsorted part of the array by one

Selection sort

- One of the easiest sorting methods to understand and code
- However it is not the most efficient
  - in the worst case (we are sorting in non-decreasing/increasing order: e.g.,1-1-2-3-4 but the array is already sorted in decreasing order: e.g., 4-3-2-1-1) for an array of length 1000, we perform roughly 1,000,000 comparisons
  - we will look at merge sort next which is more efficient than selection sort

Selection sort

- Board example
- Programming example

Merge sort

- Uses a *divide-and-conquer* strategy for sorting
- The idea is to sort a set of n numbers, etc. by splitting them into two sets of roughly equal size (n/2)
- We then sort each of the half-sized sets of numbers separately
- To complete the sorting of the original set of n numbers, we *merge* the two sorted, half-sized sets
- Note that we can apply the same split-and-merge approach when we sort the two half-sized sets
  - this means we can use a recursive approach for merge sorting

Merge sort

- Merging means that we produce from the two sorted sets a single sorted set containing all the elements in the two sets and no others
- We can use two vectors to hold these two sets
- A simple algorithm for merging the two sorted sets is:
  - compare the first elements of the two vectors holding the two sets of numbers we wish to merge
  - store the smaller of the two into a new vector that will hold the combined set of numbers (can break ties arbitrarily)
  - remove the stored element from its original vector so that this vector has a new first element
  - repeat this process until there are no elements left in the two original vectors

Merge sort

- More efficient than selection sort:
  - In the worst case we want to merge-sort two sets that are already sorted in decreasing order: e.g.,

    4-3-2-1-1 and 6-5-2-1-0

    into one set with elements in non-decreasing/ increasing order: i.e.,

    0-1-1-1-2-2-3-4-5-6
  - For a vector/array of length 1000, we would need to perform roughly 10,000 comparisons (as opposed to 1,000,000 for selection sort)

Merge sort

- Board example
- Programming example

Other sorting algorithms

There are other sorting algorithms that we won't examine in detail:

- Quicksort
- Insertion sort
- Heapsort
- Bubblesort
- Bogosort (only ever used to illustrate the worst/ most inefficient method possible for sorting)

Summary

- Linear search is most appropriate for searching through a collection of unsorted items
- It is not very efficient, but is easy to program
- Binary search is a more efficient search method than linear search
- It works for arrays/collections that are already sorted (is essentially the strategy that humans use for searching a phone book or dictionary)
  - this strategy is often called a *divide-and-conquer* strategy
- Strategy for searching for a name in one section of a phone book is the same as initial strategy for searching for the name in the entire phone book
- This implies that we can solve the binary search problem using recursion

Summary

- Selection sort is one of the easiest sorting methods to understand and code
- Interchanges smallest number in array with first location in array
- It is not the most efficient sorting method
- Merge sort uses a divide-and-conquer strategy for sorting
- More efficient than selection sort

Read

Foundations of Computer Science by Aho and Ullman

- Chapter 2
- Chapter 3

Tip

- Start Homework 4 early!