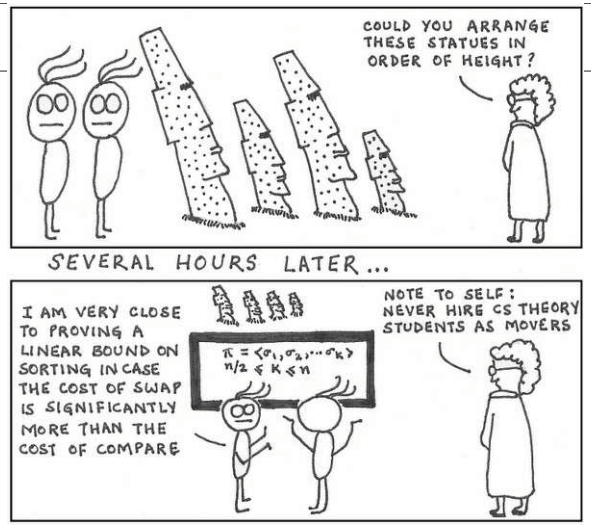

Slides #15
Sections 9.1-9.5

Sorting and Searching

Topics

1) How can we sort data in an array?
   a) Selection Sort
   b) Insertion Sort
2) How can we search for an element in an array?
   a) Linear Search
   b) Binary Search

Sorting

Sorting

- Sorting is..
- Examples:
  - Sorting an array of names into alphabetical order.
  - Sorting an array of stock prices into descending order.
- It's a classic computer science problem:
  - Theoretical analysis possible (later).
  - Many possible sorting algorithms.
  - Generally algorithms evaluated...

Selection sort

- Algorithm Idea:
  - Search list to find the...
    - Exchange element with first item.
  - Search list to find the...
    - Exchange element with second item.
  - ...
  - Repeat until all items are in their place.

Selection sort example

- Sort this list using selection sort:
  8 1 6 9 6 4 2 0

Selection sort

When working with vector v1:
v1.size() is...
(unsigned 0) – 1 = ~4 billion.
Use:
...< static_cast<int>(v1.size()) - 1;...

```
void selectionSort (int data[], int size)
{
    // Work our way through the list
    for (int index = 0; index < size-1; index++) {
        int minIdx = index;
        // Find the next smallest value
        for (int scan = index+1; scan < size; scan++){
            if (data[scan] < data[minIdx]) {
                minIdx = scan;
            }
        }
        // Swap the values
        int temp = data[minIdx];
        data[minIdx] = data[index];
        data[index] = temp;
    }
}
```

```
int main() {
    const int POINTS = 5;
    int sortMe[] = {5, 10, 1, 18, 3};

    selectionSort(sortMe, POINTS);
    ...
}
```


Insertion sort

- Insertion Sort functions by:

- Algorithm description:
  - Skip the 1st element; it's already a sorted sub-list!
  - Take the 2nd element, insert it into the sorted sub-list.
  - Take the 3rd element, insert it into the sorted sub-list.
  - ...
  - Repeat until...
    has been inserted into the sorted sub-list.

Insertion sort example

- Sort this list using insertion sort:
  8 1 6 9 6 4 2 0

Insertion sort

```
void insertionSort (int data[], int size) {
    for (int index = 1; index < size; index++) {
        int key = data[index];
        int position = index;

        //  Shift larger values to the right
        while ( (position > 0)
            && (key < data[position-1]) )
        {
            data[position] = data[position-1];
            position--;
        }
        // Put the key into the hole we made
        data[position] = key;
    }
}
```

```
int main() {
    const int POINTS = 5;
    int sortMe[] = {5, 10, 1, 18, 3};

    insertionSort(sortMe, POINTS);
    ...
}
```


Criteria for selecting a sort algorithm

- Simplicity:
  Simple algorithms are easier to...
- Faster algorithms generally win out for..
  - Ex: all SFU students, all Canadians seniors.
    - # Item Comparisons
    - # Item Swaps
- How much memory is needed for each algorithm?
  - Some sort algorithms use large amounts of memory.

Review

- Which sort algorithm most resembles sorting a hand of cards as you are dealt cards one at a time?
- Draw out sorting the following using selection sort. Show only the swaps, and what is already sorted.
  4 8 1 0 7

Searching

Searching

- Searching involves...
  - Ex: “Find the number 25 in the collection”
  - or sometimes: “Is the number 25 in the collection?”
  - and commonly: “Find Bob's phone number.”
- Definitions:
  - Target element:
  - Search pool:

About searching

- There are many search algorithms.
  - Generally, we want the one which...
- A search can result in:
  - Finding the target element in the search pool (and returning its index), or
  - Proving that the target element is...

Linear search

- Linear search:
  -
    until have found the target element or
    have examined all elements.
- It's “linear” search because:
  - start with the first element and linearly advance to the last element.

Linear search example

- Given the following search pool:
  Val: 8 19 71 5 16 27 38 40 0 56 26 10 24 30
- Use linear search to find the following:
  - 24
  - 8
  - 28
- Count how many comparisons were needed.

Linear search

```
// Find the index of the target element.
//   data:      Elements to search.
//   size:      Number of elements in data[]
//   target:    Value to find.
//   returns:   Index of target; -1 for not found.
int linearSearch (int data[], int size, int target)
{
    // Cycle through all elements
    for (int index = 0; index < size; index ++) {
        // When we find the item, return it's index.
        if (data[index] == target) {
            return index;
        }
    }
    // Item not found:
    return -1;
}
```

```
int main() {
    const int N = 5;
    int myData[] = {5, 10, 1, 18, 3};

    int pos = linearSearch(myData, N, 18);
    cout << "Index " << pos;
    ...
```


Binary search introduction

- Limitation:
  - Binary search works on...
- Idea:
  - Each comparison...
- Similar to how to play "guess the number [1...100]".
  - Guess 50, it's less than that: [1 ... 49]
  - Guess 25, it's more than that: [26 ... 49]
  - Guess 37, it's less than that: [26 ... 36]
  - Guess 31, it's less than that: [26 ... 30]
  - Guess 28, it's more than that: [29 ... 30]
  - Guess 30, it's less that that: Answer is 29!

Binary search description

- Binary search works as follows:
  - Start by looking at the middle element of the set.
    - If it's equal to the target, you are done!
    - If mid-element is less than the target...
    - If mid-element is greater than the target...
  - Repeat the above until:
    - You've found the element; or
    - There are...

Binary search example

Middle Formula:
(min + max) / 2

- Given the following search pool:

Idx:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Val:	0	5	8	10	16	19	24	26	27	30	38	40	56	71

- Use binary search to find the following:
  - 56
  - 0
  - 28
- Count how many comparisons were needed.

Binary search code

```
int binarySearch (int data[], int size, int target)
{
    int min=0, max=size-1, mid=0;
    // Narrow in the [min, max] bounds
    while (min <= max) {
        mid = (min+max) / 2;
        if (data[mid] == target) {
            return mid;
        } else {
            if (target < data[mid]) {
                max = mid-1;
            } else {
                min = mid+1;
            }
        }
    }
    return -1;   // Not found, return -1.
}
```

```
int main() {
    const int N = 5;
    int myData[] = {1, 3, 5, 10, 18};

    int pos = binarSearch(myData, N, 18);
    cout << "Index " << pos;
    ...
```


Linear vs binary search

- Comparisons:
  - requires a sorted list.
  - is slower (on average).
  - is easier to understand, implement and debug.
- Algorithm Selection:
  - If it's easy to keep the data sorted or you'll be searching a lot, use binary search.
  - Otherwise, linear search may be better.

Review

- Fill in the following table for number of comparisons required to find elements in the following list.

2 5 7 8 11

	Linear Search	Binary Search
Find 7		
Find 11		
Find 6		

Summary

- Searching and Sorting are two classic computing science problems.
- Sorting:
  - Selection sort: Finds next smallest item.
  - Insertion sort: Sort next item into existing list.
- Searching:
  - Linear: Look at each element to find item.
  - Binary: Look half way through sorted list to find which half target element could be in.
- Runtime efficiency (time) is how most algorithms are characterized.