

Chapter 14

Introduction to Algorithms

14.1 Basics

- An algorithm is a recipe on how to solve certain given problems
 - Independent from programming language
- Frequently used algorithms
 - Sorting an array
 - Searching for a value

14.2 Sorting

14.2.1 How to sort an array?

- Usually helpful to figure out how a problem is solved in real life.
 - How do you sort a deck of cards?
 - Many different algorithms.
 - The simplest is bubble sort
 - More efficient possibilities are
 - * Insertion sort
 - * Selection sort
 - * Quicksort
 - * Heapsort

14.2.2 Bubble Sort

- Simplest sorting algorithm (but not the most efficient one)
- Compare each element of an array with its neighbor.
 - If they are in the wrong order, exchange them.
 - Do this until there are no such neighbors.

14.2.3 Possible implementation

- Implementation of `swap` is left as an exercise.

```
void bubblesort(int len, int array[]) {  
    int sorted = 0;
```

```

while(!sorted) {
    sorted = 1; // assume all are in order
    for(int i = 0; i < len - 1; ++i) {
        if(array[i] > array[i + 1]) {
            // wrong order, swap values.
            swap(&array[i], &array[i+1]);
            sorted = 0; // not in order...
        }
    }
}

```

14.3 Searching

14.3.1 Sequential Search

- Searching in an unsorted array done sequentially
 - Look at all elements until you found the correct one or reach the end.
 - Inefficient but simple

14.3.2 Example

```

// returns index of "findme" or len if it does not exist.
int seqsearch(int findme, int array[], int len) {
    for(int i = 0; i < len; ++i) {
        if(array[i] == findme) {
            return i; // found it.
        }
    }

    return len;
}

```

14.3.3 Binary Search

- If elements are sorted much faster approach can be used
 - Looking up elements in a dictionary
- Binary search
 - If there are no elements to investigate, fail.
 - Pick element in the middle
 - Compare target value and element in the middle
 - * If same as target value, success.
 - * If smaller repeat on the left-hand side
 - * If larger repeat on the right-hand side

14.3.4 Implementation

- Implementation usually recursive
 - Returns index of the target value (-1 if non-existent)

```
int binsearch(int target, int first, int last, int array[]) {  
    if(first > last) return -1; // not found  
  
    int m = (first + last) / 2;  
    if(target == array[m]) return m; // success  
    else if(target < array[m]) return binsearch(target, first, m - 1);  
    else return binsearch(target, m + 1, last); // (target > array[m])  
}
```

