Searching and Sorting
(Savitch, Chapter 7.4)

TOPICS

• Algorithms
• Complexity
• Binary Search
• Bubble Sort
• Insertion Sort
• Selection Sort
What is an algorithm?

• A finite set of precise instructions for performing a computation or for solving a problem.

• Read about the history behind this word (http://en.wikipedia.org/wiki/Algorithm).
Examples

• Find the maximum in a finite sequence of integer values
• Locating an element in an ordered list – search problem
  – Linear search
  – Binary search
• Sorting
Characteristics of any algorithm

- Input and Output
- Definiteness: steps defined precisely
- Correctness: does it always work?
- Finiteness: Finite number of steps. Some people call this property: “An algorithm always terminates”
- Effectiveness: Each step must be performed exactly and in a finite amount of time
- Generality: Applicable to all instances of a problem, not just for a specific set of inputs
Linear search

- Find an element (e) in an array (array) of elements
  public static boolean search(int[] array, int element) {
    for (int i = 0; i < array.length; i++) {
      if (array[i] == element)
        return true;
    }
    return false;
  }

- Similar problems:
  – Find the number of duplicate elements in a list.
  – Find the maximum, minimum, median, mode.
  – Palindrome detection.
Complexity

- The complexity (cost) of an algorithm is measured in instructions
  - Time taken to execute the algorithm is too machine dependent
- But the number of instructions depends on the size of the input
- So complexity is measured in instructions per input
Example: Search

• Imagine a method that is given an array of integers and returns true iff at least one of the numbers is even.
  – The method loops through the array
  – For each integer, it uses the % operator to test if the integer is even

• How complex is this method?
public boolean evenp(int array[]) {
    int limit = array.length;
    for(int i=0; i < limit; i++) {
        if (0 == array[i] % 2) {
            return true;
        }
    }
    return false;
}
Complexity Example

• How complex is evenp?
  – Depends on the length of the array
  – Assume array.length == N

• How many instructions per integer?
  – Test if i < limit
  – Apply modulus operator
  – Test if result is zero
  – Increment i
Complexity Example (cont)

- What if the first number is even?
- What if none of the numbers are even?

- We typically analyze the worst-case scenario for its complexity
  - In this case, the worst case is that none of the integers are even
  - Average-case and best-case analyses are possible, but less useful
Complexity of evenp

• So the worst-case complexity of evenp is approximately $4n+2$
  – Initializing limit: 1
  – Initializing i: 1

• Does the +2 matter? $\text{No}$
• Does the 4 matter? $\text{Usually no}$
Complexity of evenp

• Its cost is linear with the size of the array
• Double the array size, double the cost
• A fixed number of operations per input is what matters
Other algorithms of similar complexity

- Find the maximum in an (unsorted) array
- Sum the elements of an array
- Sum the squared elements of an array
- Match two strings
Binary search

• Guess the number game: (from wikipedia)
  – "I'm thinking of an integer between forty and sixty inclusive, and to your guesses I'll respond 'High', 'Low', or 'Yes!' as might be the case."
  – If the numbers are between 1 and 16, how many guesses do you need?

• Use binary search when the array is already sorted
public static boolean binarySearch(int[] array, int element) {
    int first = 0;
    int last = array.length - 1;
    boolean found = false;
    while (!found && first < last) {
        int middle = first + (last - first) / 2;
        if (array[middle] == element) found = true;
        else if (array[middle] < element) first = middle + 1;
        else if (array[middle] > element) last = middle - 1;
    }
    return found;
}
Binary search example

Element: c
Array: a b c d e e f g
Index: 0 1 2 3 4 5 6 7

1. Check the midpoint ‘d’ of array[0] and array[7]: Go low

2. Check the midpoint ‘b’ of array[0] and array[2]: Go high

3. Check the midpoint ‘c’ of array[2] and array[2]: Found it!
Binary Search Complexity

- Each iteration of the algorithm has a constant number of steps
- On each iteration, the data size is cut in half
- How many iterations?
  - Data sizes: n, n/2, n/4, ..., 1
  - Number of iterations: \( \log(n) \)
Sorting

• Sorting is the process of arranging a list of items into a particular order
• There must be some value on which the order is based
• There are many algorithms for sorting a list of items
• These algorithms vary in efficiency, and may depend on original ordering!
Java animations

- [Website](http://www.sorting-algorithms.com/)
- Find out about other sites using Google
- We will examine three specific algorithms:
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
Bubble sort (First pass)

\[ i = 0 \]
Bubble sort (Second pass)

\[ i = 1 \]
Bubble sort

• Complete the rest of the passes as an exercise
public static void sort(int a[]) {
    for (int i = 0; i < a.length - 1; i++) {
        for (int j = 0; j < a.length - 1 - i; j++) {
            if (a[j] > a[j+1]) {
                // swap the elements
                int tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
        }
    }
}
public static void main(String[] args) {
    int[] numbers = {6, 5, 4, 3, 2, 1};
    sort(numbers);
    for (int i=0; i<numbers.length; i++)
        System.out.print(numbers[i] + " ");
    System.out.println();
}
Bubble sort

• The algorithm keeps executing the loops even if the numbers are already sorted.
• Can you improve the algorithm so that it stops in such cases?
Insertion Sort

• The approach of Insertion Sort:
  – pick any item and insert it into a sorted sublist
  – repeat until all items have been inserted

• In more detail:
  – consider the first item to be a sorted sublist (of one item)
  – insert the second item into the sorted sublist
  – shift items as necessary to make room to insert new item
  – insert the third item into the sorted sublist (of two items)
  – shift items as necessary to make room to insert new item
  – repeat until all values are inserted into their proper position
3 is sorted.  
Shift nothing.  
Insert 9.

3 and 9 are sorted.  
Shift 9 to the right.  
Insert 6.

3, 6, 9 are sorted.  
Shift 9, 6, 3, to the right.  
Insert 1.

1, 3, 6, 9 are sorted.  
Shift 9, 6, 3 to the right.  
Insert 2.
Insertion Sort

- Example summary (without animation):

  original: 3 9 6 1 2
  insert 9: 3 9 6 1 2
  insert 6: 3 6 9 1 2
  insert 1: 1 3 6 9 2
  insert 2: 1 2 3 6 9
Developing the code: Insertion sort

• At any point some part of the list is already sorted.
  – In the beginning, the first element by itself can be considered as a (trivial) sorted list.
• Insert one by one, the subsequent elements in this sorted list.
• In other words, at the \( j^{\text{th}} \) step, the \( j^{\text{th}} \) element is inserted into the correct position of the list of previously sorted \( j-1 \) elements.
Insertion sort code
(Lewis & Loftus)

// Sorts the specified array of integers using the insertion sort algorithm.
public static void insertionSort (int[] numbers) {

    for (int index = 1; index < numbers.length; index++) {

        int key = numbers[index];
        int position = index;

        // shift larger values to the right
        while (position > 0 && numbers[position-1] > key) {
            numbers[position] = numbers[position-1];
            position--;
        }

        numbers[position] = key;
    }
}
Selection Sort

• The approach of Selection Sort:
  – select one value and put it in its final place in sorted list
  – repeat for all other values

• In more detail:
  – find the smallest value in the list
  – switch it with the value in the first position
  – find the next smallest value in the list
  – switch it with the value in the second position
  – repeat until all values are placed
Example

Scan right starting at 3.
1 is the smallest.
Exchange 1 and 3.

Scan right starting at 9.
2 is the smallest.
Exchange 9 and 2.

Scan right starting at 6.
3 is the smallest.
Exchange 6 and 3.

Scan right starting at 6.
6 is the smallest.
Exchange 6 and 6.
Selection Sort

• Example summary (without animation):

original:       3  9  6  1  2
smallest is 1:  1  9  6  3  2
smallest is 2:  1  2  6  3  9
smallest is 3:  1  2  3  6  9
smallest is 6:  1  2  3  6  9
Selection sort code
(Lewis & Loftus)

// Sorts the specified array of integers using the selection sort algorithm.
public static void selectionSort (int[] numbers) {

    int min, temp;
    for (int index = 0; index < numbers.length-1; index++) {

        minimum = index;
        for (int scan = index+1; scan < numbers.length; scan++)
            if (numbers[scan] < numbers[minimum])
                minimum = scan;

        // Swap the values
        temp = numbers[minimum ];
        numbers[minimum ] = numbers[index];
        numbers[index] = temp;
    }
}