Introduction to algorithms

Dr Yuhua Li
School of Computer Science & Informatics

(Content adapted from slides by Dr Louise Knight)

Outline of lecture

- Overview of this part of the course
- What is an algorithm and how to express it?
- Sorting algorithms
- Selection sort
- Testing algorithms
- Timing algorithms
- Analysis of selection sort
- Comparing algorithms

Overview of this part of the course

- Algorithms
  - Introduction to algorithms
  - Big O notation, bubble sort and insertion sort
  - Merge sort
  - Goal-directed searching
  - Branch-and-bound for knapsack problem
- Data Structures
  - LinkedList
  - Stacks
  - Queues
- Module Delivery
  - TWO lectures each week:
    - Tuesdays @13:10 ~ 14:00 in N/4.07
    - Thursdays @16:10 ~ 17:00 in S/1.32
  - Exercise Classes [PGR support]
    - Tuesday 15:10 to 16:00 in C/2.04-2.05 [GROUP A]
    - Tuesday 16:10 to 17:00 in C/2.04-2.05 [GROUP B]
- Coursework hand-out week 8 (50% of module mark)

Overview of this part of the course

- What is an algorithm?
- Sorting algorithms
- Searching algorithms
- Time complexity
- What factors can influence this?
- How can we measure it?

What is an algorithm?

- An algorithm is a finite set of instructions for accomplishing a particular task
- An algorithm may have 0+ inputs
- An algorithm must produce at least 1 output
- Examples:
  - Recipes for preparing food
  - Instructions for building something

Data Structures versus Algorithms

- Data structures = how to store/represent information e.g. array
- Algorithms = manipulating data to “do something useful” e.g. pseudocode to calculate average in week 1

But…

- Data structures need operations performed on them…
- …and algorithms need to operate on data

What is an algorithm and how to express it?

Expressing an algorithm

- Natural languages (such as English) have difficulty in clearly expressing complex algorithms
- Programming languages like Java are precise but you get bogged down with the syntax
- Pseudocode is a good compromise, allowing us to express algorithms clearly without being bothered about the precise details of the programming language
- Can resemble different programming languages. Can also develop your own, as long as it's clear

What is an algorithm and how to express it?

Sorting algorithms

Aim: to sort an array of *n* integers into ascending (non-descending) order

Examples:

[1, 2, 3, 4, 5] -> [1, 2, 3, 4, 5]

[3, 4, 5, 2, 4, 5] -> [2, 3, 4, 4, 5, 5]

[5, 4, 3, 2, 1] -> [1, 2, 3, 4, 5]

Selection sort

1. Look through the array to find the smallest item
2. Exchange this item with the first item in the array. This moves the smallest item to the start of the array as required
3. Look through the array to find the second-smallest item. Exchange this with the second item in the array. We start examining the array at the second item since we know the first item in the array is the smallest
4. Repeat the same basic steps for the third, fourth, fifth, etc. smallest item in the array. In step *i* of the algorithm we find the i^{th} smallest item and move it to the i^{th} location in the array

Selection sort example

Input = [10, 4, 14, -3, 12, 6]

1. Examine items starting from the 1st position (index 0), find smallest item, and exchange with the 1st item:

[-3, 4, 14, 10, 12, 6]

2. Examine items starting from the 2nd position (index 1), find second smallest item, and exchange with the 2nd item:

[-3, 4, 14, 10, 12, 6]

3. Examine items starting from the 3rd position (index 2), find third smallest item, and exchange with the 3rd item:

[-3, 4, 6, 10, 12, 14]

Selection sort example

4. Examine items starting from the 4th position (index 3), find fourth smallest item, and exchange with the 4th item:

[-3, 4, 6, 10, 12, 14]

5. Examine items starting from the 5th position (index 4), find fifth smallest item, and exchange with the 5th item:

[-3, 4, 6, 10, 12, 14]

How many steps to sort an array? This depends on size of array…

5 steps here – last step compares last 2 items and puts them in correct order (n – 1)

Selection sort pseudocode outline

Algorithm selectionSort(A,n):
  Input: An array A storing n integers.
  Output: Array A sorted in non-descending order.
  for $i = 0$ **to** $n - 2$ **do**
    (look through the array starting from index i
      and find the smallest item)
    (exchange the found item with the item at index i)

Selection sort detailed pseudocode

Algorithm selectionSort(A,n):
Input: An array A storing n integers.
Output: Array A sorted in non-descending order.
for $i = 0$ **to** $n - 2$ **do**
$min \leftarrow i$
// find the smallest element in the rest of the array
for $j \leftarrow (i + 1)$ **to** $n - 1$ **do**
if $A[j] < A[min]$ **then** $min \leftarrow j$
// place the i^{th} smallest element in place
if $i \neq min$ **then**
$temp \leftarrow A[i]$
$A[i] \leftarrow A[min]$
$A[min] \leftarrow temp$

How do we know our algorithm works?

"It works" means two things:

- It sorts an array of numbers correctly
- It follows the right algorithm to sort the numbers

- We can test the program to show this by printing out intermediate values (e.g. the array after each step)
- We can test the pseudocode by performing a dry run on paper

Testing the program

- Observe the array each time a swap is carried out
- You should run the test on several small typical datasets

You may want the program to be robust and work with extreme or erroneous inputs, such as:

- An array with no numbers in it
- An array with one number in it
- An array that is already sorted
- An array that is in reverse sorted order
- An array containing some negative numbers

Measuring the running time of an algorithm

Two ways of doing this:

- Examine the pseudocode or program
- Run the program and time it

Counting steps of selection sort

```
Algorithm selectionSort(A,n):
    Input: An array A storing n integers.
    Output: Array A sorted in non-descending order.
    for i = 0 to n − 2 do
        min ← i
        // find the smallest element in the rest of the array
        for j ← (i + 1) to n − 1 do
            if A[j] < A[min] then min ← j
        // place the i^th smallest element in place
        if i ≠ min then
            temp ← A[i]
            A[i] ← A[min]
            A[min] ← temp
```

n – 1 iterations

For first iteration of outer loop, n – 1
For second iteration, n – 2
…
For second-to-last, 2
For last, 1

Sum of arithmetic series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$ (Don't need to know where this comes from.)

Last term we want to count is n – 1, not n, so this becomes

$(n – 1)((n – 1) + 1)/2$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Which is

$n(n – 1)/2$

Total comparisons = $(n – 1) + n(n – 1)/2 = n^2/2 + n/2 - 1$

Analysis of selection sort

- Total comparisons = $(n - 1) + n(n - 1)/2 = n^2/2 + n/2 - 1$
- This equation has the form

$T(n) = Cn^2 + Bn + A$

where $T(n)$ is the runtime

- As n gets bigger and bigger, can ignore smaller terms and coefficient C and just say $T(n)$ is proportional to n^2

Comparing algorithms

Can roughly compare different algorithms by:

- Assuming time taken is directly proportional to number of instructions carried out
- Looking at how number of instructions varies with number of inputs, *n*

Summary

- Overview of this part of the course
  - Coursework hand-out week 8
- What is an algorithm and how to express it?
  - Instructions to accomplish a task
  - Expressed using pseudocode
- Sorting algorithms – sort array of *n* integers into ascending order
- Selection sort – in step *i*, find i^{th} smallest item and put in i^{th} location in array

Summary

- Testing algorithms
  - Sorts array into ascending order
  - Using the right algorithm
- Timing algorithms
  - Count steps in pseudocode
  - Time program
- Analysis of selection sort
  - Total comparisons = $n^2/2 + n/2 - 1$
  - T(n) proportional to n^2
- Comparing algorithms – using number of steps (more detail in next lecture)