CS122 Algorithms and Data Structures

MW 11:00 am - 12:15 pm, MSEC 101

Instructor: Xiao Qin

Lecture 19: Sorting (1)

1

Introduction

- Common problem: sort a list of values, starting from lowest to highest.
  - Telephone directory
  - Words of dictionary in alphabetical order
  - Students names listed alphabetically
- Choose a criteria which is used to order data
- Given a list of records that have *keys*, we use these keys to define an ordering of the items in the list.

2

Elementary Sorting Algorithms

- We are given *n* records to sort.
- There are a number of simple sorting algorithms whose worst and average case performance is quadratic $O(n^2)$:
  - Insertion sort
  - Selection sort
  - Bubble sort

3

The Insertion Sort Algorithm

- Given an array of integers
- The Insertion Sort algorithm views the array as having a sorted side and an unsorted side.
- The sorted side starts with just the first element, which is not necessarily the smallest element.
- The sorted side grows by taking the front element from the unsorted side.

4

The Insertion Sort Algorithm (cont.)

- The sorted side grows by taking the front element from the unsorted side.
- Inserting it in the place that keeps the sorted side arranged from small to large.
- In some cases there is no need to move the new inserted item.

5

How to Insert an Element

- Copy the new element to a separate location.
- Shift elements in the sorted side, creating an open space for the new element.
- Continue shifting elements until you reach the location for the new element.
- Copy the new element back into the array, at the correct location.

6

The Insertion Sort Algorithm

```
InsertionSort(data[ ], n)
  for i = 1 to n-1
      take next key from unsorted part of array
      insert in appropriate location in sorted part of array:
        for j = i down to 0,
            shift sorted elements to the right if data > data[i]
      increase size of sorted array by 1
```


Insertion Sort Time Analysis

- In O-notation, what is:
  - Worst case running time for *n* items?
  - Average case running time for *n* items?
- Steps of algorithm:

```
for i = 1 to n-1   Outer loop: O(n)
    take next key from unsorted part of array
    insert in appropriate location in sorted part of array:
        for j = i down to 0,   Inner loop: O(n)
              shift sorted elements to the right if key > key[i]
    increase size of sorted array by 1
```


The Selection Sort Algorithm

- Start by finding the smallest element.
- Swap the smallest entry with the first element.
- Part of the array is sorted.
- Find the smallest element in the unsorted side.
- Swap with the front of the unsorted side.
- The size of the sorted side is increased by one element.
- Continue until the unsorted side has just one number. Why?

The Selection Sort Algorithm (cont.)

Basic Idea: Repeatedly select the smallest element, and move this element to the front of the unsorted side.

```
Selectsort(data[], n)
   for i = 1 to n-1
      find smallest key in unsorted part of array;
      swap smallest item to front of unsorted array;
      decrease size of unsorted array by 1;
```


Selection Time Sort Analysis

- In O-notation, what is:
  - Worst case running time for *n* items?
  - Average case running time for *n* items?
- Steps of algorithm:

```
for i = 1 to n-1   O(n)
    find smallest key in unsorted part of array   O(n)
    swap smallest item to front of unsorted array
    decrease size of unsorted array by 1
```

- Selection sort analysis: $O(n^2)$

The Bubble Sort Algorithm

- Scan the array from right to left.
- Look at pairs of elements (adjacent elements) in the array, and swaps their order if needed.
- Repeatedly scan the array from right to left elements until you reach the location for the new element.
- Continue scanning until done

The Bubble Sort Algorithm (cont.)

```
BubbleSort(data[ ], n)
   for i = 0 to n-1
      for j =0 to n-1
           if data[j] > data[j+1] then swap
      if no elements swapped in this pass through
      array, done.
      otherwise, continue
```

13

Bubble Sort Time Analysis

Steps of algorithm:

```
for i = 0 to n-1                              O(n)
   for j =0 to n-1                            O(n)
       if data[j] > data[j+1] then swap
  if no elements swapped in this pass through array,
  done.
  otherwise, continue
```

14

Conclusions

- Selection Sort, Insertion Sort, and Bubble Sort all have a worst-case time of $O(n^2)$, making them impractical for large arrays.
- But they are easy to program, easy to debug.
- Insertion Sort also has good performance when the array is nearly sorted to begin with.
- But more sophisticated sorting algorithms are needed when good performance is needed in all cases for large arrays.
- Next time: Quick Sort, Merge Sort, and Radix Sort.

15