Sorting Algorithms Part 1

Anton Gerdelan <gerdela@scss.tcd.ie>

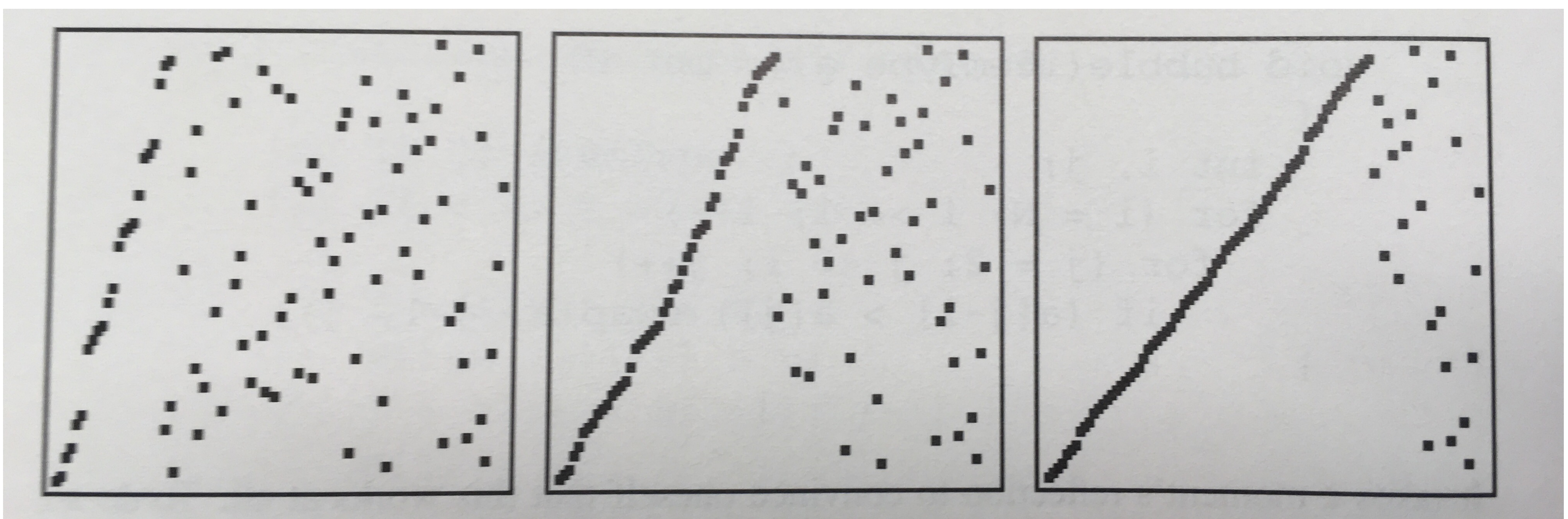

Figure 8.4 Insertion sorting a random permutation.

Value on y axis, index on x axis. From Sedgewick "Algorithms in C++"

Overview and Rules

- We talk about sorting **files** of **records**.
- Records may be quite complex, so sort actually operates on simpler **keys**.
- <u>Sorts re-arrange sets of keys</u>
    - e.g. numerical or alphabetical order.
- This gives us the elegant representation:

  input **instance** S = {9, 2, 3, 8}

  sorting algorithm goes to work in here

  output **permutation** S' = {2, 3, 8, 9}
- We can .: use arrays of integers to represent keys in our demos.
- Can swap sorting functions to compare on same arrays.
- Consider an **indirect sort** to avoid shuffling v. <u>large</u> data (sort array of pointers instead).

Stability

- A sorting algorithm is **stable** if it preserves original order of equal-value keys

- e.g. records show student names originally in alphabetical order. but keys are based on grades.

- Elementary sorts are usually stable.

- Sophisticated sorts are usually unstable.

- Can modify key. Turns out to be quite tricky or expensive.

Elementary Sorts

- We will start with simple 'elementary sorts'
  - sometimes these are the best choice for bigger problems
  - start with simple problems to learn general ideas
- Try to build functions that implement all of these: good practise
- Build a table comparing time and space complexity
- Know special pros and cons of each beyond complexity

Elementary Sorts

- Properties
  - Usually take N^2 steps to sort N items .: bad on big data sets
  - Simple to implement
  - May actually be quicker on mostly-sorted data
- Selection sort
- Insertion sort
- Bubble sort
- Shell sort - *can be a great choice on random data*

Selection Sort Algorithm

- "*select the smallest remaining element*"
- Given set ***S*** of ***n*** values
- **Loop** over array from left to right (`array`[0] to `array`[n-2])
    - **Loop** over rest of array (**[current_index] to [n-1]**) looking for smallest value
        - **Swap** `array[current_index]` and `array[min_index]`
- Blackboard walk-through here
- Live coding demo here…

Selection Sort

- Ease of implementation?
- Time complexity avg., best, worst O(…)?
- Space complexity base O(…)? and *auxiliary* (amount of extra space used whilst working).
- Why is this *stable*? hint: one operation
- Bad for?
- Good for?
  - consider that every item is moved max once.

Aside on Unix-style programs

- Why are we talking about sorting records within files?
- It usually **is** a file with a series of lines as records
    - text list of strings
    - csv or spreadsheet — maybe grades are the key
- If your print the output to the console e.g. printf
- redirect printed output to a new file
    - `./mysort input.csv > output.csv`
- can also chain wee programs together with pipes |
    - `./mysoft input.csv | ./mycsv2json > output.json`

Insertion Sort Algorithm

- "*insert new card into already sorted hand of cards*"
- start at left of array and loop once to right
  - check new card to the right of current card
  - if it's smaller than current card swap them over
  - this should loop all the way back to the left e.g.

    S = {2,3,4,**5**,1}
  - imagine our current card is the 5
- Idea how to code this?

```
for ( int pass = 0; pass < length - 1; pass++ ) {
  for ( int i = pass + 1; i > 0; i-- ) {
    if ( array[i - 1] > array[i] ) {
      int temp = array[i];
      array[i] = array[i - 1];
      array[i - 1] = temp;
    } else {
      break;
    }
  }
}
```

- Loop over array length -1
- Nested loop goes **backwards** from next element to [1]
- Compare current and next elements
- If next is smaller, do a swap and continue left
- Otherwise break and continue main loop right

Insertion Sort

- Time and space complexity?
- Other advantages
    - one at a time input
    - implementation simple

Bubble Sort Algorithm

- sorted = false
- while (sorted == false)
    - sorted = true
    - loop over data
        - if (next < current)
            - swap(current, next)
            - sorted = false

Bubble Sort

- Time and space complexity? Worst, average, **best**?
- Advantages:
    - Code is simple
    - Can stop early if numbers already sorted
        - No other sorting algorithm does this
        - Can do one run to check before calling complex sorting algorithm
    - "Stable"
- *Sedgewick has a different algorithm called Bubble Sort
- Computer scientists have very negative things to say about Bubble Sort's worst case performance vs Insertion Sort.

Summary - Elementary Sorting Algorithms

- Very simple to implement. Also interchangeable. Some useful properties.
- O(n^2) worst case time
  - May not play well with cache - try them with a timer
- O(1) auxiliary memory (1 variable for swapping)
- Stable