Data Structures and Algorithms

CS245-2015S-10

Sorting

David Galles

Department of Computer Science
University of San Francisco
10-0: Main Memory Sorting

- All data elements can be stored in memory at the same time
- Data stored in an array, indexed from 0 \ldots n - 1, where \( n \) is the number of elements
- Each element has a key value (accessed with a `key()` method)
- We can compare keys for <, >, =
- For illustration, we will use arrays of integers – though often keys will be strings, other Comparable types
A sorting algorithm is *Stable* if the relative order of duplicates is preserved.

The order of duplicates matters if the keys are duplicated, but the records are not.

---

**A non-Stable sort**
10-2: Insertion Sort

- Separate list into sorted portion, and unsorted portion
- Initially, sorted portion contains first element in the list, unsorted portion is the rest of the list
  - (A list of one element is always sorted)
- Repeatedly insert an element from the unsorted list into the sorted list, until the list is sorted
10-3: $\Theta()$ For Insertion Sort

- Running time $\propto$ # of comparisons
- Worst Case:
For Insertion Sort

- Running time $\propto$ # of comparisons
- Worst Case: Inverse sorted list

# of comparisons:
For Insertion Sort

- Running time $\propto$ # of comparisons
- Worst Case: Inverse sorted list

# of comparisons:

$$\sum_{i=1}^{n-1} i \in \Theta(n^2)$$
For Insertion Sort

- Running time $\propto$ # of comparisons
- Best Case:
For Insertion Sort

- Running time $\propto$ # of comparisons
- Best Case: Sorted List

# of comparisons:
For Insertion Sort

- Running time $\propto$ # of comparisons
- Best Case: Sorted List

# of comparisons:

$$n - 1$$
10-9: Bubble Sort

- Scan list from the last index to index 0, swapping the smallest element to the front of the list
- Scan the list from the last index to index 1, swapping the second smallest element to index 1
- Scan the list from the last index to index 2, swapping the third smallest element to index 2
  ...
- Swap the second largest element into position \((n - 2)\)
10-10: $\Theta()$ for Bubble Sort

- Running time $\propto$ # of comparisons
- Number of Comparisons:
10-11: $\Theta()$ for Bubble Sort

• Running time $\propto$ # of comparisons

• Number of Comparisons:

$$\sum_{i=1}^{n-1} i \in \Theta(n^2)$$
10-12: Selection Sort

- Scan through the list, and find the smallest element
- Swap smallest element into position 0
- Scan through the list, and find the second smallest element
- Swap second smallest element into position 1
  
  ... 

- Scan through the list, and find the second largest element
- Swap smallest largest into position $n - 2$
θ() for Selection Sort

- Running time $\propto$ # of comparisons
- Number of Comparisons:
for Selection Sort

- Running time $\propto$ # of comparisons
- Number of Comparisons:

$$\sum_{i=1}^{n-1} i \in \Theta(n^2)$$
• Insertion sort is fast if a list is “almost sorted”

• How can we use this?
  • Do some work to make the list “almost sorted”
  • Run insertion sort to finish sorting the list

• Only helps if work required to make list “almost sorted” is less than $n^2$
10-16: Shell Sort

- Sort \( \frac{n}{2} \) sublists of length 2, using insertion sort
- Sort \( \frac{n}{4} \) sublists of length 4, using insertion sort
- Sort \( \frac{n}{8} \) sublists of length 8, using insertion sort
  ...
- Sort 2 sublists of length \( \frac{n}{2} \), using insertion sort
- Sort 1 sublist of length \( n \), using insertion sort
10-17: **Shell’s Increments**

- Shell sort runs several insertion sorts, using increments
  - Code on monitor uses “Shell’s Increments”:
    \[ \{n/2, n/4, \ldots 4, 2, 1\} \]

- Problem with Shell’s Increments:
  - Various sorts do not interact much
  - If all large elements are stored in large indices, and small elements are stored in even indices, what happens?
Shell’s Increments: \( \{n/2, n/4, \ldots 4, 2, 1\} \)
- Running time: \( O(n^2) \)

“/3” increments: \( \{n/3, n/9, \ldots , 9, 3, 1\} \)
- Running time: \( O(n^{3/2}) \)

Hibbard’s Increments: \( \{2^k - 1, 2^{k-1} - 1, \ldots 7, 3, 1\} \)
- Running time: \( O(n^{3/2}) \)
10-19: Shell Sort: Best case

- What is the best case running time for Shell Sort (using Shell’s increments)
- When would the best case occur?
What is the best case running time for Shell Sort (using Shell’s increments)

- When would the best case occur?
  - When the list was originally sorted

- How long would each pass through Shell Sort take?
What is the best case running time for Shell Sort (using Shell's increments)?

- When would the best case occur?
  - When the list was originally sorted
- How long would each pass through Shell Sort take?
  - $\Theta(n)$
- How Many Passes?
Shell Sort: Best case

- What is the best case running time for Shell Sort (using Shell’s increments)
  - When would the best case occur?
    - When the list was originally sorted
  - How long would each pass through Shell Sort take?
    - $\Theta(n)$
  - How Many Passes?
    - $\lg n$
  - Total running time?
What is the best case running time for Shell Sort (using Shell’s increments)?

When would the best case occur?
- When the list was originally sorted

How long would each pass through Shell Sort take?
- $\Theta(n)$

How Many Passes?
- $\log n$

Total running time?
- $\Theta(n \log n)$
10-24: Stability

- Is Insertion sort stable?
- Is Bubble Sort stable?
- Is Selection Sort stable?
- Is Shell Sort stable?
10-25: Stability

- Is Insertion sort stable? Yes!
- Is Bubble Sort stable? Yes!
- Is Selection Sort stable? No!
- Is Shell Sort stable? No!