CSE 326 Lecture 14: Sorting

Today’s Topics:
✦ Elementary Sorting Algorithms:
  ➤ Bubble Sort
  ➤ Selection Sort
  ➤ Insertion Sort
  ➤ Shellsort

✦ Covered in Chapter 7 of the textbook

Sorting: Definitions

✦ Input: You are given an array A of data records, each with a key (which could be an integer, character, string, etc.).
  ➤ There is an ordering on the set of possible keys
  ➤ You can compare any two keys using <, >, ==

✦ For simplicity, we will assume that A[i] contains only one element – the key

✦ Sorting Problem: Given an array A, output A such that:
  For any i and j, if i < j then A[i] ≤ A[j]

✦ Internal sorting: all data in main memory
✦ External sorting: data on disk
Why Sort?

✦ Sorting algorithms are among the most frequently used algorithms in computer science
  ➤ Crucial for efficient retrieval and processing of large volumes of data. E.g. Database systems
✦ Allows binary search of an N-element array in \(O(\log N)\) time
✦ Allows \(O(1)\) time access to \(k\)th largest element in the array for any \(k\)
✦ Allows easy detection of any duplicates

Sorting: Things to Think about…

✦ **Space**: Does the sorting algorithm require extra memory to sort the collection of items?
  ➤ Do you need to copy and temporarily store some subset of the keys/data records?
  ➤ An algorithm which requires \(O(1)\) extra space is known as an **in place** sorting algorithm
Sorting: More Things to Think about…

- **Stability**: Does it rearrange the order of input data records which have the same key value (duplicates)?
  - E.g. Given: Phone book sorted by name. Now sort by county – is the list still sorted by name within each county?
  - Extremely important property for databases
  - A **stable sorting algorithm** is one which does not rearrange the order of duplicate keys

---

Sorting 101: Bubble Sort

- **Idea**: “Bubble” larger elements to end of array by comparing elements i and i+1, and swapping if A[i] > A[i+1]
  - Repeat from first to end of unsorted part

- **Example**: Sort the following input sequence:
  - 21, 33, 7, 25
Sorting 101: Bubblesort

/* Bubble sort pseudocode for integers
 * A is an array containing N integers */

for(int i=0; i<N; i++) {
    /* From start to the end of unsorted part */
    for(int j=1; j<(N-i); j++) {
        /* If adjacent items out of order, swap */
    }
}

✦ Stable? In place? Running time = ?

Sorting 102: Selection Sort

✦ Bubblesort is stable and in place, but O(N^2) – can we do better by moving items more than 1 slot per step?

✦ Idea: Scan array and select smallest key, swap with A[1]; scan remaining keys, select smallest and swap with A[2]; repeat until last element is reached.

✦ Example: Sort the following input sequence:

21, 33, 7, 25

✦ Is selection sort stable (suppose you had another 33 instead of 7)? In place?
✦ Running time = ?
Sorting 102: Selection Sort

✦ Bubblesort is $O(N^2)$ – can we do better by moving items more than 1 slot per step?
✦ Example: Sort the following input sequence:
  $\Rightarrow$ 21, 33, 7, 25
✦ NOT STABLE. In place (extra space = 1 temp variable).
✦ Running time = $N$ steps with $N-1$, …, 1 comparisons
  $= N-1 + \ldots + 1 = O(N^2)$

Sorting 103: Insertion Sort

✦ What if first $k$ elements of array are already sorted?
  $\Rightarrow$ E.g. 4, 7, 12, 5, 19, 16
✦ Idea: Can insert next element into proper position and get $k+1$ sorted elements, insert next and get $k+2$ sorted etc.
  $\Rightarrow$ 4, 5, 7, 12, 19, 16
  $\Rightarrow$ 4, 5, 7, 12, 16, 19  Done!
  $\Rightarrow$ Overall, N-1 passes needed
  $\Rightarrow$ Similar to card sorting…
  $\Rightarrow$ Start with empty hand
  $\Rightarrow$ Keep inserting…

Shift right
Sorting 103: Insertion Sort
/* Insertion sort pseudocode for integers
 * A is an array containing N integers */

int j, P, Tmp;
for(P = 1; P < N; P++ ) {
    Tmp = A[ P ];
    for(j = P; j > 0 && A[ j - 1 ] > Tmp; j-- )
}

✦ Is Insertion sort in place? Stable?
✦ Running time = ?

R. Rao, CSE 326

Sorting 103: Insertion Sort

int j, P, Tmp;
for(P = 1; P < N; P++ ) {
    Tmp = A[ P ];
    for(j = P; j > 0 && A[ j - 1 ] > Tmp; j-- )
}

✦ Insertion sort: In place (O(1) space for Tmp) and stable
✦ Running time: Worst case is reverse order input = Θ(N²)
  ⇒ Best case is input already sorted = O(N).
Lower Bound on Simple Sorting Algorithms

- An inversion is a pair of elements in wrong order
  \[ i < j \text{ but } A[i] > A[j] \]
- Our simple sorting algorithms so far swap adjacent elements (explicitly or implicitly): swapping removes 1 inversion
  \[ \text{Running time proportional to no. of inversions in array} \]
- Given N distinct keys, total of \( N(N-1)/2 \) possible inversions.
  \[ \text{Average list contains: } N(N-1)/4 \text{ inversions} \]
  \[ \text{Average running time of Insertion sort is } \Theta(N^2) \]
- Any sorting algorithm that swaps adjacent elements requires \( \Omega(N^2) \) time: Each swap removes only one inversion

Shellsort: Breaking the Quadratic Barrier

- **Main Insight:** Insertion sort runs fast on nearly sorted sequences do several passes of Insertion sort on different subsequences of elements
- Example: Sort 19, 5, 2, 1
  1. Do Insertion sort on subsequences of elements spaced apart by 2: 1st and 3rd, 2nd and 4th
     \[ 19, 5, 2, 1 \]
  2. Do Insertion sort on subsequence of elements spaced apart by 1:
     \[ 2, 1, 19, 5 \]
- Note: Fewer number of shifts than plain Insertion sort
  \[ 4 \text{ versus } 6 \text{ for this example} \]
Shellsort: Overview

- Named after Donald Shell – first algorithm to achieve $o(N^2)$
  - Running time is $O(N^x)$ where $x = 3/2, 5/4, 4/3, \ldots$, or 2 depending on “increment sequence”
- In our example, we used the increment sequence: $N/2, N/4, \ldots, 1 = 2, 1$ (for $N = 4$ elements)
  - This is Shell’s original increment sequence
- Shellsort: Pick an increment sequence $h_t > h_{t-1} > \ldots > h_1$
  - Start with $k = t$
  - Insertion sort all subsequences of elements that are $h_k$ apart so that $A[i] \leq A[i+h_k]$ for all $i$ (known as an $h_k$-sort)
  - Go to next smaller increment $h_{k-1}$ and repeat until $k = 1$ (note: $h_1 = 1$)

Shellsort: An Example (a pathetic one)

- Example: Shell’s original sequence: $h_t = N/2$ and $h_k = h_{k+1}/2$
  - Sort 21, 33, 7, 25
  - Try it! (What is the increment sequence?)
Shellsort: An Example

✦ Example: Shell’s original sequence: \( h_i = N/2 \) and \( h_k = h_{k+1}/2 \)

\( 21, 33, 7, 25 \) (\( N = 4 \), increment sequence = 2, 1)

\( 7, 25, 21, 33 \) (after 2-sort)

\( 7, 21, 25, 33 \) (after 1-sort)

Shellsort: The Nuts and Bolts

```c
/* Shell sort pseudocode for integers
 A is an array containing N integers */
int i, j, Increment, Tmp;
for( Increment = N/2; Increment > 0; Increment /= 2 )
  for( i = Increment; i < N; i++ ) {
    Tmp = A[ i ];
    for( j = i; j >= Increment &&
         A[ j - Increment ] > Tmp ; j -= Increment )
    A[ j ] = Tmp;
  }
✦ Note: The two inner for loops correspond almost exactly to the code for Insertion sort!
✦ Running time = ? (What is the worst case?)
```
Shellsort: Run Time Analysis

✦ Simple to code but hard to analyze
  ➔ Run time depends on increment sequence

✦ What about the increment sequence $h_k = N/2, N/4, \ldots, 2, 1$?
  ➔ Upper bound
  ♦ Shellsort does $h_k$ insertion sorts with $N/h_k$ elements for $k = 1$ to $t$
  ♦ Running time = $O(\sum_{k=1}^{t} h_k (N/h_k)^2)$
    = $O(N^2 \sum_{k=1}^{t} 1/h_k)$ = $O(N^2)$

➔ Lower bound
  ♦ What is the worst case?

Shellsort: Run Time Analysis

✦ What about the increment sequence $N/2, N/4, \ldots, 2, 1$?
  ➔ Lower bound
  ♦ What is the worst case?
  ♦ Suppose smallest elements in odd positions, largest in even positions in sorted order:
    2, 11, 4, 12, 6, 13, 8, 14
  ♦ None of the passes $N/2, N/4, \ldots$ do anything!
  ♦ Last pass ($h_t = 1$) must shift $N/2$ smallest elements to first half and $N/2$ largest elements to second half
  ♦ 4 shifts 1 slot, 6 shifts 2, 8 shifts 3, \ldots = $1 + 2 + 3 + \ldots$ ($N/2$ terms)
  ♦ Run time = At least $N^2$ steps = $\Omega(N^2)$
Shell sort: Can we do better?

- The reason we got $\Omega(N^2)$ was because of increment sequence
  - Adjacent increments have common factors (e.g. 8, 4, 2, 1)
  - We keep comparing same elements over and over again
  - Need to increment so that different elements are compared in different passes
- Is there a better increment sequence than N/2, N/4, ..., 2, 1?

Shell sort: How to Break the $O(N^2)$ Barrier

- Hibbard’s increment sequence: $2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1$
  - Adjacent increments have no common factors
  - Worst case running time of Shell sort with Hibbard’s increments = $\Theta(N^{1.5})$ (Theorem 7.4 in text)
  - Average case running time for Hibbard’s = $O(N^{1.25})$ in simulations but nobody has been able to prove it! (next homework assignment?)
- Final thoughts on the “Simple Sorts” discussed today:
  - Insertion sort good for small input sizes (~20)
  - Shell sort better for moderately large inputs (~10,000)
After Midterm: The crème de la crème of Sorts: 
Heapsort, Mergesort, and Quicksort 

Next Class: Midterm Review 

To Do: 

Midterm on Wed Feb 12: Read Chapters 1 through 6 
HW #3 due: Thu Feb 13