CSE 373 Lecture 15: 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 memory, External ➔ 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 kth 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
✦ Stability: Does it rearrange the order of input data records which have the same key value (duplicates)?
  ➤ E.g. 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 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 = ?

```c
/* Bubble sort for integers */
#define SWAP(a,b) { int t; t=a; a=b; b=t; }
void bubble( int A[], int n ) {
    int i, j;
    for(i=0;i<n;i++) /* n passes thru the array */
        for(j=1;j<(n-i);j++) {
            /* If adjacent items out of order, swap */
        }
}
```

✦ Stable? 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?
✦ 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
  🤗 NOT STABLE. In place (extra space = 1 temp variable).
  🤗 Running time = N steps with N-1, …, 1 comparisons
  = N-1 + … + 1 = O(N^2)
What if first \( k \) elements of array are already sorted?

- 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.

\[
\begin{align*}
4, 5, 7, 12, 19, 16 \\
4, 5, 7, 12, 16, 19 \\
4, 5, 7, 12, 16, 19, \text{ Done!}
\end{align*}
\]

- Overall, \( N-1 \) passes needed
- Similar to card sorting...
- Start with empty hand
- Keep inserting...

\[
\text{Overall, } N-1 \text{ passes needed}
\]

\[
\text{Similarto cardsort...}
\]

\[
\text{Start with empty hand}
\]

\[
\text{Keep inserting...}
\]

**Insertion sort**

- In place (O(1) space for Tmp) and stable
- Running time: Worst case \( \Theta(N^2) \)

- Running time = ?

---

**Lower Bound on Simple Sorting Algorithms**

- An inversion is a pair of elements in wrong order
  \( i < j \) but \( A[i] > A[j] \)
- Our simple sorting algorithms so far swap adjacent elements (explicitly or implicitly) \( \rightarrow \) removes 1 inversion
  \( \Theta \) Their running time is proportional to number of inversions in array
- Given \( N \) distinct keys, total of \( N(N-1)/2 \) possible inversions.
  Average list will contain half this number of inversions \( \Theta(N(N-1)/4) \)
- Average running time of Insertion sort is \( \Theta(N^2) \)
- Any sorting algorithm that swaps adjacent elements requires \( \Omega(N^2) \) time \( \rightarrow \) each swap removes only one inversion
Shellsort: Breaking the Quadratic Barrier

- Named after Donald Shell – first algorithm to achieve \(o(N^2)\)
  - Running time is \(O(N^x)\) where \(x = \frac{3}{2}, \frac{5}{4}, \frac{4}{3}, \ldots, \text{ or } 2\) depending on “increment sequence”
- Idea: Use an increment sequence \(h_1 < h_2 < \ldots < h_k\)
  - Start with \(k = 1\)
  - 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 \(k = 1\)
- Example: Shell’s original sequence: \(h_k = N/2\) and \(h_k = h_{k+1}/2\)
  - Sort 21, 33, 7, 25
  - Try it! (What is the increment sequence?)

Shellsort

```c
void Shellsort( ElementType A[], int N ){
    int i, j, Increment; ElementType Tmp;
    for( Increment = N/2; Increment > 0; Increment /= 2 )
        for( i = Increment; i < N; i++ ) {
            Tmp = A[i];
            for( j = i; j >= Increment; j -= Increment )
                if( Tmp < A[j - Increment] )
                else
                    break;
            A[j] = Tmp;
        }
}
```

- Running time = ? (What is the worst case?)

Shellsort: Breaking the Quadratic Barrier

- Named after Donald Shell – first algorithm to achieve \(o(N^2)\)
  - Running time is \(O(N^x)\) where \(x = \frac{3}{2}, \frac{5}{4}, \frac{4}{3}, \ldots, \text{ or } 2\) depending on “increment sequence”
- Idea: Use an increment sequence \(h_1 < h_2 < \ldots < h_k\)
  - Start with \(k = 1\)
  - 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 \(k = 1\)
- Example: Shell’s original sequence: \(h_k = N/2\) and \(h_k = h_{k+1}/2\)
  - Sort 21, 33, 7, 25 (\(N = 4\), increment sequence = 2, 1)
  - 7, 25, 21, 33 (after 2-sort)
  - 7, 21, 25, 33 (after 1-sort)

Answer and further analysis in next class…
Also in the next class, the crème de la crème:
Heapsort, Mergesort, and Quicksort
To Do:
If you can’t wait, read chapter 7
If you can, read chapter 7 anyway…