CSE 373 Lecture 16: Sorting Faster and Faster…

✦ What’s on our plate today?
  ➤ Faster sorting Algorithms:
    ✦ Shellsort
    ✦ Heapsort
    ✦ Mergesort
  
✦ Covered in Chapter 7 of the textbook

Recall from Last Time: Insertion Sort

✦ Main Idea:
  ➤ Start with 1st element, insert 2nd if < 1st after shifting 1st element →
    First 2 are now sorted…
  ➤ Insert 3rd after shifting 1st and/or 2nd as needed → First 3 sorted…
  ➤ Repeat until last element is correctly inserted → All N elements sorted

✦ Example: Sort 19, 5, 2, 1
  ➤ 5, 19, 2, 1 (shifted 19)
  ➤ 2, 5, 19, 1 (shifted 5, 19)
  ➤ 1, 2, 5, 19 (shifted 2, 5, 19)

✦ Running time:
  ➤ Worst case → reverse order input = \( \Theta(N^2) \)
  ➤ Best case → input already sorted = \( O(N) \)
Shellsort: Motivation

✦ **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, 1, 19, 5
2. Do Insertion sort on subsequence of elements spaced apart by 1:
   → 2, 1, 19, 5 → 1, 2, 19, 5 → 1, 2, 19, 5 → 1, 2, 5, 19

✦ **Note:** Fewer number of shifts than plain Insertion sort
→ 4 versus 6 for this example

R. Rao, CSE 373

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, …, or 2 depending on “increment sequence”

✦ **In our example, we used the increment sequence:** N/2, N/4, …, 1 = 2, 1 (for N = 4 elements)
   → This is Shell’s original increment sequence

✦ **Shellsort:** Pick an increment sequence h_k > h_{k-1} > … > h_1
   → Start with k = t
   → Insertion sort all subsequences of elements that are h_k apart so that A[i] ≤ 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)

R. Rao, CSE 373
ShellSort: Nuts and Bolts

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;
        }
}
✦ Note: The two inner for loops correspond almost exactly to the code for Insertion sort!
✦ Running time = ? (What is the worst case?)

ShellSort: Analysis
✦ Simple to code but hard to analyze \( \Rightarrow \) depends on increment sequence
✦ What about the increment sequence N/2, N/4, ..., 2, 1?
  \( \Rightarrow \) Upper bound
    ◆ ShellSort does \( h_k \) insertions sort with \( N/h_k \) elements for \( k = 1 \) to \( t \)
    ◆ Running time = \( O(\Sigma_{k=1\ldots t} h_k (N/h_k)^2) = O(N^2 \Sigma_{k=1\ldots t} 1/h_k) = O(N^2) \)
  \( \Rightarrow \) Lower bound
    ◆ What is the worst case?
Shellsort: Analysis

✦ What about the increment sequence $N/2$, $N/4$, …, 2, 1?
  ✓ Upper bound
    ✷ Shellsort does $h_k$ insertions sort 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?
    ✷ Smallest elements in odd positions, largest in even positions
      ✷ 2, 11, 4, 12, 6, 13, 8, 14
    ✷ None of the passes $N/2$, $N/4$, …, 2 do anything!
    ✷ Last pass ($h_1 = 1$) must shift $N/2$ smallest elements to first half and
      $N/2$ largest elements to second half $\rightarrow$ 4 shifts 1 slot, 6 shifts 2, 8
      shifts 3, … = $1 + 2 + 3 + \ldots$ (N/2 terms)
    ✷ at least $N^2$ steps $= \Omega(N^2)$

Shellsort: Breaking the $O(N^2)$ Barrier

✦ 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 in different passes

✦ Hibbard’s increment sequence: $2^k - 1$, $2^{k-1} - 1$, …, 7, 3, 1
  ✓ Adjacent increments have no common factors
  ✓ Worst case running time of Shellsort 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: Insertion sort good for small input sizes (~20); Shellsort better for moderately large inputs (~10,000)
Hey…How about using Binary Search Trees?

- Can we beat $O(N^{1.5})$ using a BST to sort $N$ elements?

Using Binary Search Trees for Sorting

- Can we beat $O(N^{1.5})$ using a BST to sort $N$ elements?
  - Yes!!
  - Insert each element into an initially empty BST
  - Do an In-Order traversal to get sorted output

- Running time: $N$ Inserts, each takes $O(\log N)$ time, plus $O(N)$ for In-Order traversal = $O(N \log N) = o(N^{1.5})$

- Drawback – Extra Space: Need to allocate space for tree nodes and pointers $\Rightarrow O(N)$ extra space, not in place sorting

- Waitaminute…what if the tree is complete, and we use an array representation – can we sort in place?
  - Recall your favorite data structure with the initials B. H.
Using Binary Heaps for Sorting

✦ Main Idea:
  ◆ Build a max-heap
  ◆ Do N DeleteMax operations and store each Max element in the unused end of array

Build Max-heap

DeleteMax

Largest element in correct place

Heapsort: Analysis

✦ Running time = time to build max-heap + time for N DeleteMax operations = ?
Heapsort: Analysis

- Running time = time to build max-heap + time for $N$
  DeleteMax operations = $O(N) + N O(\log N) = O(N \log N)$
- Can also show that running time is $\Omega(N \log N)$ for some
  inputs, so worst case is $\Theta(N \log N)$
- Average case running time is also $O(N \log N)$ (see text for proof if you are interested)

How about a “Divide and Conquer” strategy?

- Very important strategy in computer science:
  1. Divide problem into smaller parts
  2. Independently solve the parts
  3. Combine these solutions to get overall solution
- Idea: Divide array into two halves, recursively sort left and
  right halves, then merge two halves $\rightarrow$ known as Mergesort
- Example: Mergesort the input array:

```
0 1 2 3 4 5 6 7
8 2 9 4 5 3 1 6
```
Questions to ponder over the Weekend
Is Mergesort an in place sorting algorithm?
What is the running time for Mergesort?
How can I find time to read Chapter 7?
What is the meaning of life? (extra credit)

Have a good weekend!