Shell Sort

CSE 373 - Data Structures
May 8, 2002

Swapping adjacent elements

• An "average" list will contain half the max number of inversions = \( \frac{(n-1)n}{4} \)
  
  So the average running time of Insertion sort is \( \Theta(N^2) \)

• Any sorting algorithm that only swaps adjacent elements requires \( \Omega(N^2) \) time because each swap removes only one inversion

The search for speed

• If we are going to do better than \( O(N^2) \), we are going to have to fix more than one inversion at a time

• How can we fix more than one inversion?
  
  Move the elements further with each swap

Readings and References

• Reading
  
  › Sections 7.4, *Data Structures and Algorithm Analysis in C*, Weiss

• Other References
Shell sort: Better than Quadratic

- Named after Donald Shell – inventor of the 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”
- Shell sort uses repeated insertion sorts on selected subarrays of the larger array being sorted
- Multiple passes with changing subarrays

Subarrays (or subsequences)

- Remember that in the discussion of binary heaps I showed how we could sort a path through the tree
- Similarly, we can sort a subarray contained in a larger array

Subarrays: increment = 5

Subarrays: increment = 2
Shell Sort: diminishing increments

- Uses an *increment sequence* $h_1 < h_2 < \ldots < h_t$
  - Start sorting with the largest increment $h_t$
  - Sort all subarrays 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
- Stop sorting after $h_1 (=1)$
- Choice of increments is important
  - and hard to analyze

Shell Sort

```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;
        }
    }
}
```

Note: the actual sorting is done by insertion sort: "copy down and insert the value in the right place" on each subarray in the innermost loop and $A[j] = Tmp$

Shellsort: Basic Insight

- Insertion sort runs fast on nearly sorted sequences
  - immediate termination when proper spot is found
- do several passes of Insertion sort on different subsequences of elements
- note that the subsequences stay sorted from pass to pass
Example

- Sort 19, 5, 2, 1 with increment sequence 1,2
  - Insertion sort on subsequences of elements spaced apart by 2: 1st and 3rd, 2nd and 4th
    \[ 19, 5, 2, 1 \Rightarrow 2, 1, 19, 5 \]
  - Do Insertion sort on subsequence of elements spaced apart by 1:
    \[ 2, 1, 19, 5 \Rightarrow 1, 2, 19, 5 \Rightarrow 1, 2, 19, 5 \Rightarrow 1, 2, 5, 19 \]
- Fewer shifts than plain Insertion sort
  - 4 versus 6 for this example

Some increment sequences

- Some increments that have been studied
  - Shell's increments
    \[ h_i = \frac{N}{2} \quad h_k = \frac{h_{k+1}}{2} \]
    - bad choice since the subarrays can coincide and so you end up re-sorting something that is already sorted, and not mixing other elements that need it
  - Hibbard's increments
    - relatively prime values: 1, 3, 7, 15, 2k-1
  - Sedgewick
    \[ \{ 1, 5, 19, 41, 109, \ldots \} = 9 \cdot 4^i - 9 \cdot 2^i + 1 \text{ or } 4^i - 3 \cdot 2^i + 1 \]

Example using Shell's Increments

- Example: Shell’s original sequence: \( h_t = N/2 \) and \( h_k = h_{k+1}/2 \)
  - Sort 21, 33, 7, 25 (N = 4, increment sequence = 2, 1)
    - 21, 33, 7, 25 (after 2-sort)
    - 7, 21, 25, 33 (after 1-sort)

Shellsort: Run time

```c
void Shellsort( ElementType A[], int N ){
  // some code...
}
```
Shell sort: Shell's Increments

- Algorithm is simple to code but hard to analyze
  - depends on increment sequence
- Shell's increment sequence 1, 2, 4, ..., N/4, N/2
  - What is the Upper bound?
  - Shellsort does h_k insertion sorts with N/h_k elements for k = 1 to t
  - Running time = \( O(\sum_{k=1}^{t} \frac{N}{h_k}^2) = O(N^2 \sum_{k=1}^{t} \frac{1}{h_k}) = O(N^2) \)

Shell's Increments

- What is the lower bound?
  - Worst case is: 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
  - at least \( N^2 \) steps = \( \Omega(N^2) \)

Shell's Increments: \( \Omega(N^2) \)

- 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 increments such that different elements are in different passes
Hibbard's Increments

- 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 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!

General performance

- Insertion sort good for small input sizes
  - $\sim 20$
  - Often incorporated in other procedures where the list to be sorted is short and is likely to be sorted already

- Shellsort better for moderately large inputs
  - $\sim 10,000$