07 A: Sorting I

CS1102S: Data Structures and Algorithms

Martin Henz

March 3, 2010

Generated on Tuesday 2nd March, 2010, 10:59

1. Introduction
2. Insertion Sort
3. A Lower Bound
4. Shell Sort

1. Introduction
   - Flashback: Priority Queues
   - Comparison-based Sorting
   - A Counter-Example

2. Insertion Sort

3. A Lower Bound

4. Shell Sort

Flashback: Priority Queues

Main idea

Keep elements in complete binary tree with parent element always bigger than child elements

Requirement

Elements are *ordered* (Comparable or through Comparator)

Flashback: Hashing

Main idea

Compute hash value for elements; use hash value as index into array

Requirement

Given mapping of elements to their hash value

Sorting

Input

Unsorted array of elements

Behavior

Rearrange elements of array such that the smallest appears first, followed by the second smallest etc, finally followed by the largest element

Comparison-based Sorting

The only requirement

A comparison function for elements

The only operation

Comparisons are the only operations allowed on elements

Counter-example: Sorting Small Distinct Integers

Input

Array a of N distinct integers from 1 to M

Sorting algorithm

```
int[] helper = new int[M];
for (int i=0; i<N; i++)
   helper[a[i]] = a[i];
int index = 0;
for (int j=0; j<M; j++)
   if (helper[j]!=0)
      a[index++] = helper[j];
```

Counter-example: Sorting Small Distinct Integers

```
int[] helper = new int[M];
for (int i=0; i<N; i++)
   helper[a[i]] = a[i];
int index = 0;
for (int j=0; j<M; j++)
   if (helper[j]!=0)
      a[index++] = helper[j];
```

Analysis

Runtime $O(M + N)$

Counter-example: Sorting Small Distinct Integers

```
int[] helper = new int[M];
for (int i=0; i<N; i++)
   // the following line
   // uses elements as indices!
   helper[a[i]] = a[i];
int index = 0;
for (int j=0; j<M; j++)
   if (helper[j]!=0)
      a[index++] = helper[j];
```

Focus: Comparison-based Sorting

The only operation

Comparisons are the only operations allowed on elements

How to proceed

- Insertion Sort
- A Lower Bound
- Shell Sort
- Heap Sort

Insertion Sort: Idea

Passes

Algorithm proceeds in $N - 1$ *passes*

Invariant

After pass i, the elements in positions 0 to i are sorted.

Consequence of Invariant

After $N - 1$ passes, the elements in positions 0 to $N - 1$ are sorted.
That is the whole array!

How to do a pass?

Pass i

Move element in position i to the left, until it is larger than the element to the left or until it is at the beginning of the array.

Original	34	8	64	51	32	21	Positions Moved
After $p = 1$	8	34	64	51	32	21	1
After $p = 2$	8	34	64	51	32	21	0
After $p = 3$	8	34	51	64	32	21	1
After $p = 4$	8	32	34	51	64	21	3
After $p = 5$	8	21	32	34	51	64	4

Insertion Sort

Original	34	8	64	51	32	21	Positions Moved
After $p = 1$	8	34	64	51	32	21	1
After $p = 2$	8	34	64	51	32	21	0
After $p = 3$	8	34	51	64	32	21	1
After $p = 4$	8	32	34	51	64	21	3
After $p = 5$	8	21	32	34	51	64	4

Use of invariant

After pass i, the elements in positions 0 through i are sorted, provided that before pass i, the elements in positions 0 through $i - 1$ are sorted.

Insertion Sort: Implementation

```
public static <AnyType extends
                  Comparable<? super AnyType>>
void insertionSort(AnyType[ ] a) {
    int j;
    for( int p = 1; p < a.length; p++ ) {
        AnyType tmp = a[ p ];
        for( j = p; j > 0 &&
                tmp.compareTo(a[j - 1]) < 0; j--)
            a[j] = a[j - 1];
        a[j] = tmp;
    }
}
```

1. Introduction
2. Insertion Sort
3. A Lower Bound
   - Inversions
   - Swaps and Inversions
   - Worst Case
   - Average Case
4. Shell Sort

Basic Operation of Insertion Sort

Basic operation

The main operation of insertion sort is the *swapping of two elements*.

How many swaps are needed for sorting?

How many items are "out of place"?

Definition

An *inversion* in an array a is an ordered pair (i, j) such that $i < j$, but $a[i] > a[j]$.

Sorting by Removing Inversions

What does a swap do?

A swap removes exactly one inversion!

Consequence

The number of swaps required to sort an array is exactly the number of inversions in the array.

Worst Case

How many inversions in the worst case?

A list sorted in reverse has the maximal number of inversions

Maximal number of inversions

$$\sum_{i=0}^{N-1} i = N(N-1)/2$$

Average Case

How many inversions in the average case?

Consider the number of inversions in an list L and its reverse L_r.

Consider a pair of elements (x, y)

Either (x, y) is an inversion in L, or in L_r!

Overall

The sum of inversions of L and L_r *together* is $N(N-1)/2$.

Overall average

The overall average of inversions in a given list is $N(N-1)/4$

Runtime of Swapping Sorting Algorithms

Theorem

Any algorithm that sorts its elements by swapping runs in $\Omega(N^2)$.

Theorem

Any algorithm that removes one inversion in each step runs in $\Theta(N^2)$.

Idea

Main idea

Proceed in passes $h_1, h_2, \ldots, h_t$, making sure that after each pass, $a[i] \leq a[i + h_k]$.

Invariant

After pass h_k, elements are still h_{k+1} sorted

Shell Sort: Example using $\{1, 3, 5\}$

Original	81	94	11	96	12	35	17	95	28	58	41	75	15
After 5-sort	35	17	11	28	12	41	75	15	96	58	81	94	95
After 3-sort	28	12	11	35	15	41	58	17	94	75	81	96	95
After 1-sort	11	12	15	17	28	35	41	58	75	81	94	95	96

Shell Sort: Implementation

```
public static <AnyType extends
                Comparable<? super AnyType>>
void shellsort(AnyType [ ] a) {
   int j;
   for(int gap = a.length / 2; gap > 0; gap /= 2)
      for(int i = gap; i < a.length; i++) {
         AnyType tmp = a[ i ];
         for(j = i;
             j >= gap &&
             tmp.compareTo(a[j - gap]) < 0;
             j -= gap)
            a[ j ] = a[ j - gap ];
         a[j] = tmp; } }
```

Shell’s Increments

Start	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
After 8-sort	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
After 4-sort	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
After 2-sort	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
After 1-sort	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Analysis

Shell's Increments

The worst-case running time of Shellsort, using Shell's increments $1, 2, 4, \ldots,$, is $\Theta(N^2)$.

Hibbards's Increments

The worst-case running time of Shellsort, using Hibbard's increments $1, 3, 7, \ldots, 2^k - 1$, is $\Theta(N^{3/2})$.

What Next?

- Tutorial tomorrow:
  - Lab 7: solution and discussion
  - Section 5.4: Hashtables with probing
  - Questions/clarifications on Assignment 3 (due on Friday)
- Friday 5/3: Sorting II (Heapsort, Mergesort)
- Wednesday 10/3: Sorting III (Quicksort)
- Friday 12/3: Midterm 2: Trees, Hashing, Priority Queues, Sorting I + II