Sorting

Ph.D. Truong Dinh Huy

Sorting: The Problem Space

General problem

Given a set of N *orderable* items, put them *in order*

Without (significant) loss of generality, assume:

- Items are integers
- Ordering is $\leq$

Chapter 7 of textbook 1

In-place Sorting

Sorting algorithms may be performed *in-place*, that is, with the allocation of at most $\Theta(1)$ additional memory (*e.g.*, fixed number of local variables)

Other sorting algorithms require the allocation of second array of equal size

- Requires $\Theta(n)$ additional memory

We will prefer in-place sorting algorithms

Run-time

The run time of the sorting algorithms we will look at fall into one of three categories:

$$\Theta(n) \qquad \Theta(n \ln(n)) \qquad \mathbf{O}(n^2)$$

We will examine best-, average- and worst-case scenarios for each algorithm

The run-time may change significantly based on the scenario

Run-time

We will review the more traditional $\mathbf{O}(n^2)$ sorting algorithms:

- Insertion sort

Some of the faster $\mathbf{\Theta}(n \log n)$ sorting algorithms:

- Heap sort, Merge sort, and Quick sort

And linear-time sorting algorithms

- Bucket sort, Radix Sort
- We must make assumptions about the data

Insertion Sort

Initially P = 1

Let the first P elements be sorted.

(3)Then the P+1 th element is inserted properly in the list so that now P+1 elements are sorted.

Now increment P and go to step (3)

P+1th element is inserted as follows:

Store the P+1 th element first as some temporary variable, temp;

If Pth element greater than temp, then P+1th element is set equal to the Pth one,

If P-1 th element greater than temp, then Pth element is set equal to P-1th one…..

Continue like this till some kth element is less than or equal to temp, or you reach the first position.

Let this stop at kth position (k can be 1). Set the k+1th element equal to temp.

Extended Example

Need to sort:

34 8 64 51 32 21

P = 1;

Looking at first element only, and we do not change.

P = 2;

Temp = 8;

34 > Temp, so second element is set to 34.

We have reached the end of the list. We stop there. Thus, first position is set equal to Temp;

After second pass;

8 34 64 51 32 21

Now, the first two elements are sorted.

Set P = 3.

Temp = 64, 34 < 64, so stop at 3rd position and set 3rd position = 64

After third pass: 8 34 64 51 32 21

P = 4, Temp = 51, 51 < 64, so we have 8 34 64 64 32 21,

34 < 51, so stop at 2nd position, set 3rd position = Temp,

List is 8 34 51 64 32 21

Now P = 5, Temp = 32, 32 < 64, so 8 34 51 64 64 21, 32 < 51, so 8 34 51 51 64 21, next 32 < 34 so, 8 34 34, 51 64 21, next 32 > 8, so we stop at first position and set second position = 32, we have

8 32 34 51 64 21,

Now P = 6,

We have 8 21 32 34 51 64

Pseudo Code

Assume that the list is stored in an array, A (can do with a linked list as well)

```
Insertion Sort(A[],int N)
{
    for (P = 1; P < N; P++)
        {
            Temp = A[P];
            for (j = P; j > 0 and A[j-1] > Temp; j--)  A[j] = A[j-1];
            A[j] = Temp;
        }    }
```

Time complexity: O(n^2)

Quiz

Sort the sequence 3, 1, 4, 1, 5, 9, 2, 6, 5 using insertion sort

Inversions

Consider the following three lists:

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

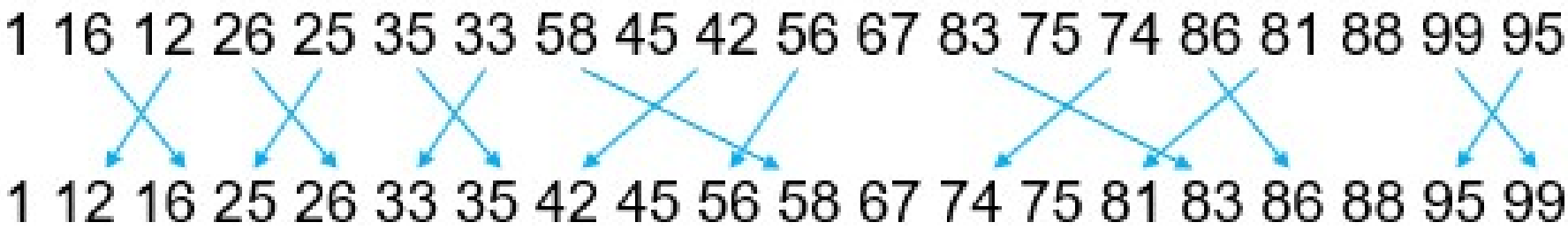
1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

To what degree are these three lists unsorted?

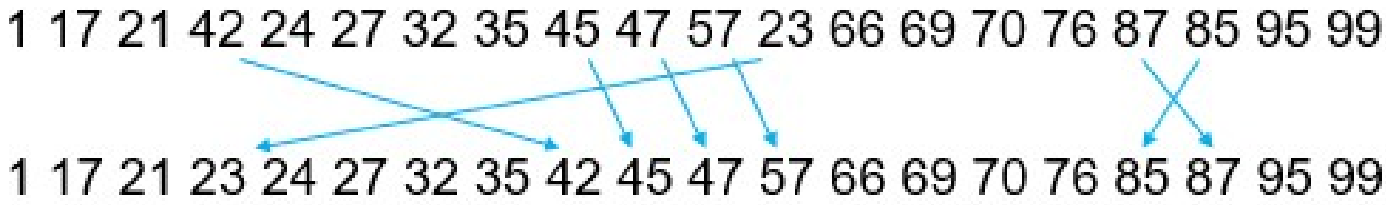
Inversions

The first list requires only a few exchanges to make it sorted



Inversions

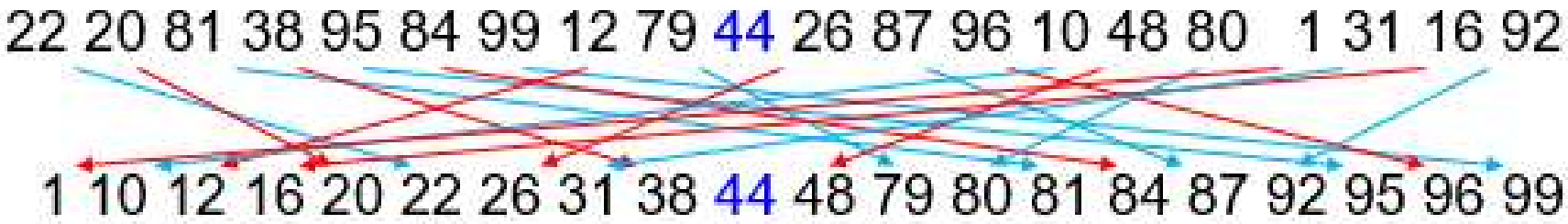
The second list has two entries significantly out of order



however, most entries (13) are in place

Inversions

The third list would, by any reasonable definition, be significantly unsorted



Inversions

Given any list of n numbers, there are

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

pairs of numbers

For example, the list (1, 3, 5, 4, 2, 6) contains the following 15 pairs:

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)	
		(3, 5)	(3, 4)	(3, 2)	(3, 6)
		(5, 4)	(5, 2)	(5, 6)	
			(4, 2)	(4, 6)	
				(2, 6)	

Inversions

You may note that 11 of these pairs of numbers are in order:

(1, 3)	**(1, 5)**	**(1, 4)**	**(1, 2)**	**(1, 6)**	
		(3, 5)	**(3, 4)**	(3, 2)	**(3, 6)**
		(5, 4)	(5, 2)	**(5, 6)**	
			(4, 2)	**(4, 6)**	
				(2, 6)	

Inversions

The remaining four pairs are *reversed*, or *inverted*

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)
		(3, 5)	(3, 4)	**(3, 2)**	(3, 6)
		(5, 4)	**(5, 2)**	(5, 6)	
			(4, 2)	(4, 6)	
				(2, 6)	

Inversions

Given a permutation of n elements

$$a_0, a_1, \ldots, a_{n-1}$$

an inversion is defined as a pair of entries which are reversed

That is, (a_j, a_k) forms an inversion if

$$j < k \text{ but } a_j > a_k$$

Inversions

Therefore, the permutation

1, 3, 5, 4, 2, 6

contains four inversions:

(3, 2) (5, 4) (5, 2) (4, 2)

Number of Inversions

There are $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of numbers in any set of n objects

Consequently, each pair contributes to

- the set of ordered pairs, or
- the set of inversions

For a random ordering, we would expect approximately half of all pairs, or

$$\frac{1}{2}\binom{n}{2} = \frac{n(n-1)}{4} = \mathbf{O}(n^2), \text{ inversions}$$

Any Algorithm that sorts by exchanging adjacent elements requires O(n^2)

Number of Inversions

Let us consider the number of inversions in our first three lists:

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

Each list has 20 entries, and therefore:

- There are $\binom{20}{2} = \frac{20(20-1)}{2} = 190$ pairs
- On average, $190/2 = 95$ pairs would form inversions

Number of Inversions

The first list

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

has 13 inversions:

(16, 12) (26, 25) (35, 33) (58, 45) (58, 42) (58, 56) (45, 42)

(83, 75) (83, 74) (83, 81) (75, 74) (86, 81) (99, 95)

This is well below 95, the expected number of inversions

- Therefore, this is likely not to be a *random* list

Complexity Analysis

Worst case analysis

Worst case occurs when for every *j* the inner loop has to move all elements $A[1], \ldots, A[j-1]$ (which happens when $A[j]$ = key is smaller than all of them), that takes $\Theta(i-1)$ time. Totally, we have:

$T(n)= \Theta(1) + \Theta(2) + \ldots + \Theta(n-1) = \Theta(n^2)$

Average case analysis

The time complexity of the algorithm depends on the number of inversion in the an array:

$T(n) = O(n+d)= O(n+n(n-1)/2)= O(n^2)$ with d is the number of **inversions**

Note: If the number of inversions in an array is low, the time complexity is low

Best case analysis

$T(n)= \Theta(n)$ when $d = \Theta(n)$ => the array has very few inversions

Additional Space requirement: O(1)

Proof of correctness of an Algorithm: loop invariants

To use this, we need to prove three conditions:

1. **Initialization:** The loop invariant is satisfied at the beginning of the for loop.
2. **Maintenance:** If the loop invariant is true before the *i*th iteration, then the loop invariant will be true before the *i* + 1st iteration.
3. **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Note that this is basically **mathematical induction** (the initialization is the base case, and the maintenance is the inductive step).

Proof of correctness of Insertion sort

Initialization: Before the first iteration (which is when $P = 1$), the subarray $[1..P - 1]$ is just the first element of the array, $A[0]$. This subarray is sorted, and consists of the elements that were originally in $A[0]$.

Maintenance: Suppose $A[1..P - 1]$ is sorted. Informally, the body of the for loop works by moving $A[P - 1]$, $A[P - 2]$, $A[\mathrm{P} - 3]$ and so on by one position to the right until it finds the proper position for $A[P]$ (lines 4), at which point it inserts the value of $A[P]$ (line 5). The subarray $A[1..P]$ then consists of the elements originally in $A[1..P]$, but in sorted order. Incrementing P for the next iteration of the for loop then preserves the loop invariant.

Termination: The condition causing the for loop to terminate is that $P > N$. Because each loop iteration increases j by 1, we must have $j = P + 1$ at that time. By the initialization and maintenance steps, we have shown that the subarray $A[1..N + 1 - 1] = A[1..N]$ consists of the elements originally in $A[1..N]$, but in sorted order.

Summary

- Introduction of sorting
- Insertion sort
- Lower bound of simple sorting Algorithms
- Proof of correctness of Algorithms: loop invariants
- Next week: Merge Sort, Quick Sort, and Bucket Sort