ECE430.217 Data Structures

Insertion sort

Textbook: Weiss Chapter 7.2, 7.3

Byoungyoung Lee

https://compsec.snu.ac.kr

byoungyoung@snu.ac.kr

Lecture slides were prepared based on Douglas W. Harder's notes (**dwharder@alumni.uwaterloo.ca).**

Outline

This topic discusses the insertion sort

We will discuss:

- the algorithm
- an example
- run-time analysis
  - worst case
  - average case
  - best case

Observation

Consider the following observations:

- A list with one element is sorted
- Suppose you are given a sorted list of k items
  - In general, we can always find a right spot to insert a new item, which creates a sorted list of size $k + 1$

Observation: Example

For example, consider this sorted array containing of eight sorted entries

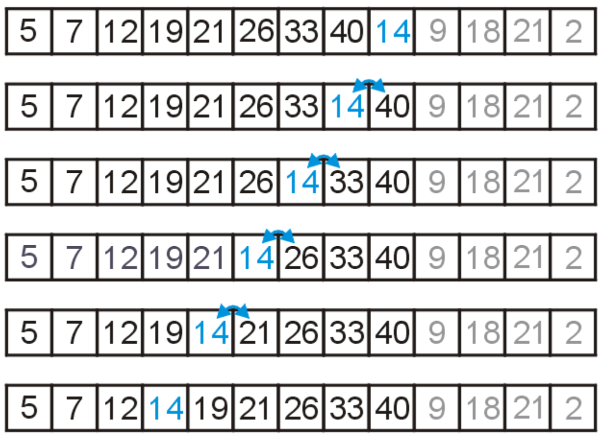
5	**7**	**12**	**19**	**21**	**26**	**33**	**40**	**14**	9	18	21	2

Suppose we want to insert 14 into this array leaving the resulting array sorted

Observation: Example

Starting at the back, if the number is greater than 14, copy it to the right

- Once an entry less than 14 is found, it’s now sorted



The Algorithm: Insertion Sort

For any unsorted list:

- Treat the first element as a sorted list of size 1

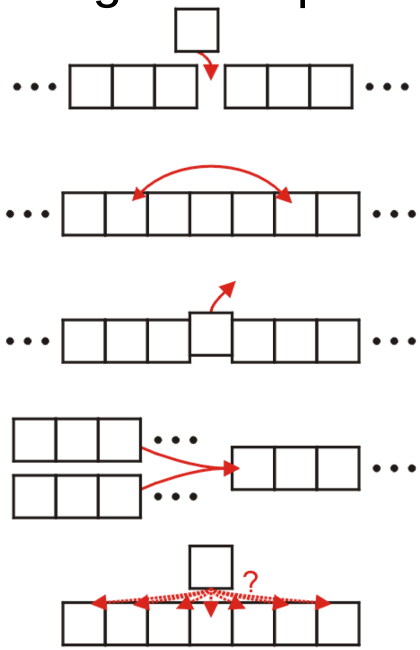
Then, given a sorted list of size $k - 1$

- Insert the k^{th} item in the unsorted list
- The sorted list is now of size k

The Algorithm: Insertion Sort

Recall the five sorting techniques:

- Insertion
- Exchange
- Selection
- Merging
- Distribution

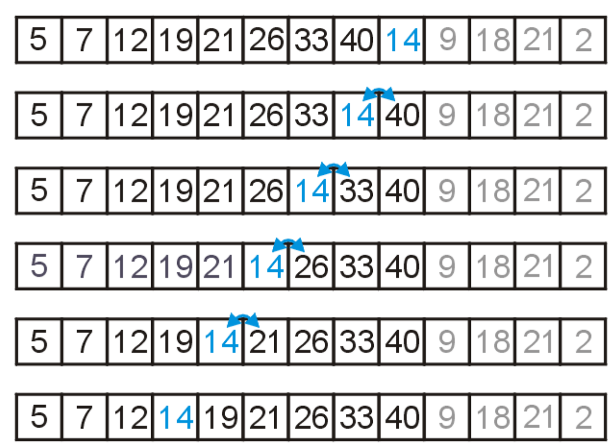


Clearly insertion falls into the first category

Implementation

Code for this would be:

```
template <typename Type>
void insertion_sort( Type *const array, int const n ) {
    for (int k = 1; k < n; ++k ) {
        for (int j = k; j > 0; --j ) {
            if (array[j - 1] > array[j] ) {
                std::swap( array[j - 1], array[j] );
            } else {
               // As soon as we don't need to swap,
               // the (k + 1)st is in the correct location
               break;
            }
        }
    }
}
```



Runtime Analysis?

- **Q. How do we compute the runtime of insertion sort?**
  - Worst case?
  - Average case?
  - Best Case?

- **It's tricky to compute the runtime based on the implementation**
  - The inner loop conditionally breaks out of the loop

Inversions

- **Idea**: Count the number of "swap()" calls

- To count the number of "swap()", let's talk about inversions
  - The number of inversions impact the runtime of insertion sort
  - In fact, this is **true for all swap/exchange based sorting algorithms**

Consider the following list:

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

Q. To what degree is this list sorted (or unsorted)?

Inversions

The list requires only a few exchanges to make it sorted

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

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

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)

You can notice that 11 of these pairs of numbers are in order.

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

Inversions

Given a permutation of n elements

$$a_0, a_1, ..., a_{n-1}$$

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

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

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

Ref: Bruno Preiss, *Data Structures and Algorithms*

Inversions

Exchanging (or swapping) two adjacent entries either:

- **Case A. removes an inversion**, *e.g.*, (4,2)

  1 3 5 4 2 6

  1 3 5 2 4 6

- **Case B. introduces a new inversion**, *e.g.*, (5, 3) with

  1 3 5 4 2 6

  1 5 3 4 2 6

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 either

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

For a random ordering, we would expect approximately half of all pairs are inverted.

So the number of inverted pairs are

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

Runtime Analysis

- The number of inversions impact the runtime of insertion sort
  - Worst: $O(n^2)$
    - Reverse sorted
  - Average: $O(n^2)$
    - The random list has $\mathrm{O}(n^2)$ inversions on average
  - Best: $O(n)$
    - Very few inversions (i.e., $O(n)$)

References

Wikipedia, http://en.wikipedia.org/wiki/Insertion_sort

[1] Donald E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd Ed., Addison Wesley, 1998, §5.2.1, p.80-82.

[2] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw Hill, 1990, p.2-4, 6-9.

[3] Weiss, *Data Structures and Algorithm Analysis in C++*, *3rd Ed.*, Addison Wesley, §8.2, p.262-5.

[4] Edsger Dijkstra, *Go To Statement Considered Harmful*, Communications of the ACM 11 (3), pp.147–148, 1968.

[5] Donald Knuth, *Structured Programming with Goto Statements*, Computing Surveys 6 (4): pp.261–301, 1972.