Simple Sorting

Chapter 3

Contents

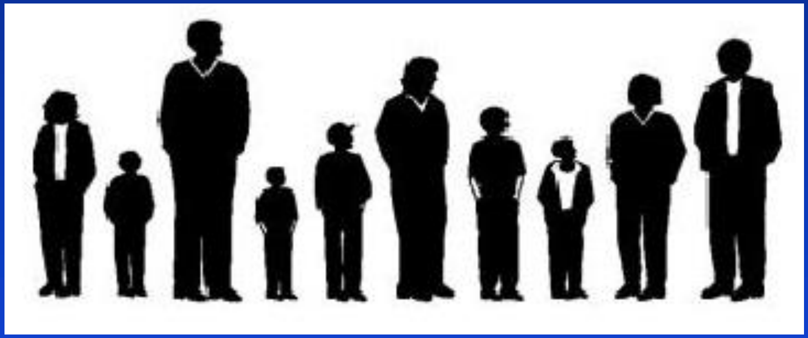
- Bubble Sort
- Selection Sort
- Insertion Sort
- Comparison

Sorting in Databases

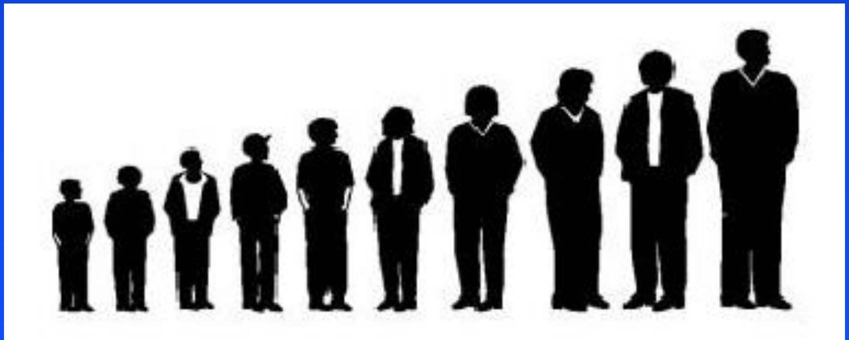
- Many possibilities
  - Names in alphabetical order
  - Students by grade
  - Customers by zip code
  - Home sales by price
  - Cities by population
  - Countries by GNP
  - Stars by magnitude

Example

- Unordered:



- Ordered:



Simple Sorts

- All three algorithms involve two basic steps, which are executed repeatedly until the data is sorted
  - Compare two items
  - Either (1) swap two items, or copy one item

- They differ in the details and order of operations

Bubble Sort

- Compare two elements.
- If the one on the left is larger, swap them
- Move one position right to continue another comparison
- When reach the first sorted number, start over at the left.

Continue the process until all the data elements are in order.

Bubble Sort

- The algorithm involve two steps :
  1. Compare two items.
  2. Swap two items.

Bubble Sort : source code

```
public void bubbleSort()
 {
     int out, in;

     for(out=nElems-1; out>0; out--)   // outer loop (backward)
       for(in=0; in<out; in++)         // inner loop (forward)
         if( a[in] > a[in+1] )         // out of order?
           swap(in, in+1);             // swap them
  }  // end bubbleSort()
```

Bubble Sort : Example

Bubble Sort : Efficiency

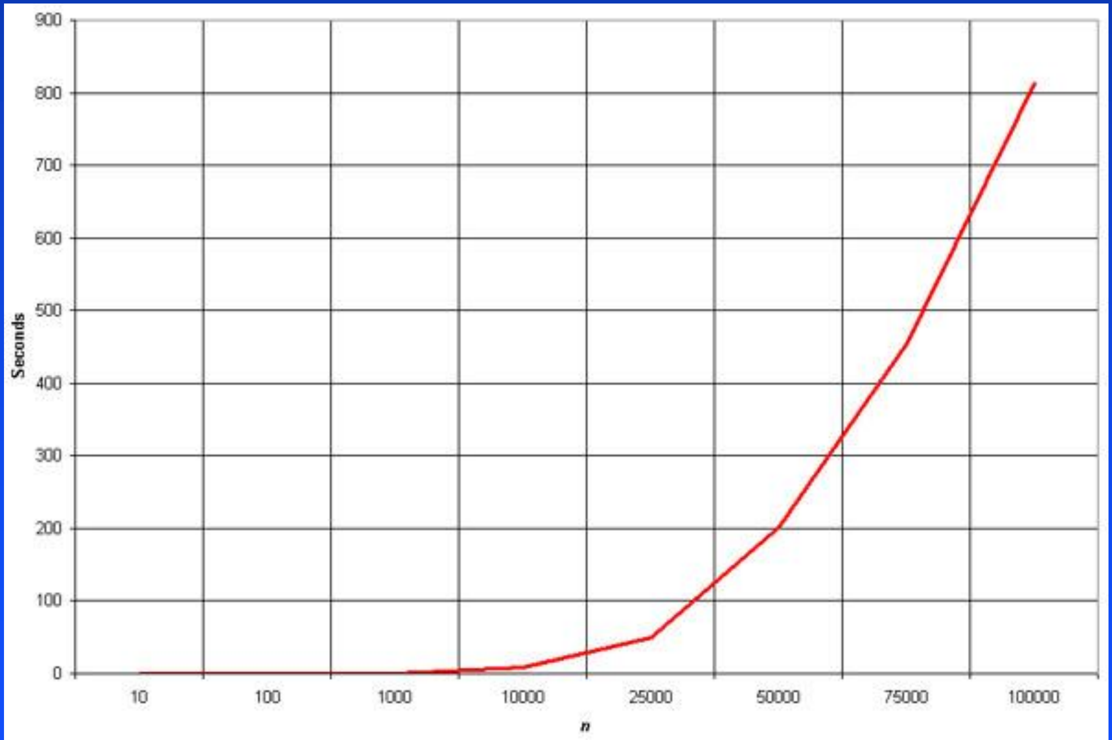
- N elements in the array.
- First pass: N-1 comparisons and somewhere between 0 and N-1 swaps.
- Second pass: N-2 comparisons and somewhere between 0 and N-2 swaps
- Third pass: N-3 comparisons and somewhere between 0 and N-3 swaps
- (n-1)th pass: 1 comparisons and somewhere between 0 and 1 swap
- Then it is sorted.

Bubble Sort : Efficiency

- In general, where N is the number of items in the array, there are N-1 comparisons on the first pass, N-2 on the second, and so on. The formula for the sum of such a series is
- $(N-1) + (N-2) + (N-3) + \ldots + 1 = N * (N-1)/2$
- The algorithm makes about $N^2/2$ comparisons.
- If the data is random, a swap is necessary about half the time, so there will be about $N^2/4$.
- Bubble sort runs in $O(N^2)$ time.

Bubble Sort : Efficiency

- **Pros:** Simplicity and ease of implementation.
- **Cons:** Horribly inefficient.



Average of a hundred runs against random data sets on a single-user 250MHz UltraSPARC II

Selection Sort

- Pass through all the numbers and select the smallest one.
- This smallest one is then swapped with the number on position 0.
- Now the leftmost number is sorted and won't need to be moved again.
- The next time, start at position 1, and pass through the data and finding the smallest, swap with position 1.
- This process continues until all the numbers are sorted.

Selection Sort

- There are two basic steps used for the selection sort algorithm:
  1. Compare the items
  2. Swap the items

Selection Sort: Source code

```
public void selectionSort()
 {
        int out, in, min;

        for(out=0; out<nElems-1; out++)   // outer loop
        {
          min = out;                       // minimum
          for(in=out+1; in<nElems; in++)  // inner loop
            if(a[in] < a[min] )            // if min greater,
                min = in;                  // we have a new min
          swap(out, min);                  // swap them
         }  // end for(out)
  }  // end selectionSort()
```

Selection Sort: Source code

How many operations total then?

- First pass: n-1 comparisons, 1 swap
- Second pass: n-2 comparisons, 1 swap
- Third pass: n-3 comparisons, 1 swap
- (n-1)th pass: 1 comparison, 1 swap
- Then it's sorted
- So we have (worst case):
  - $(n-1)+(n-2)+\dots+1$ comparisons = $n(n-1)/2 = (n^2 - n)/2$
  - $1+1+\dots+1$ swaps = $n-1$

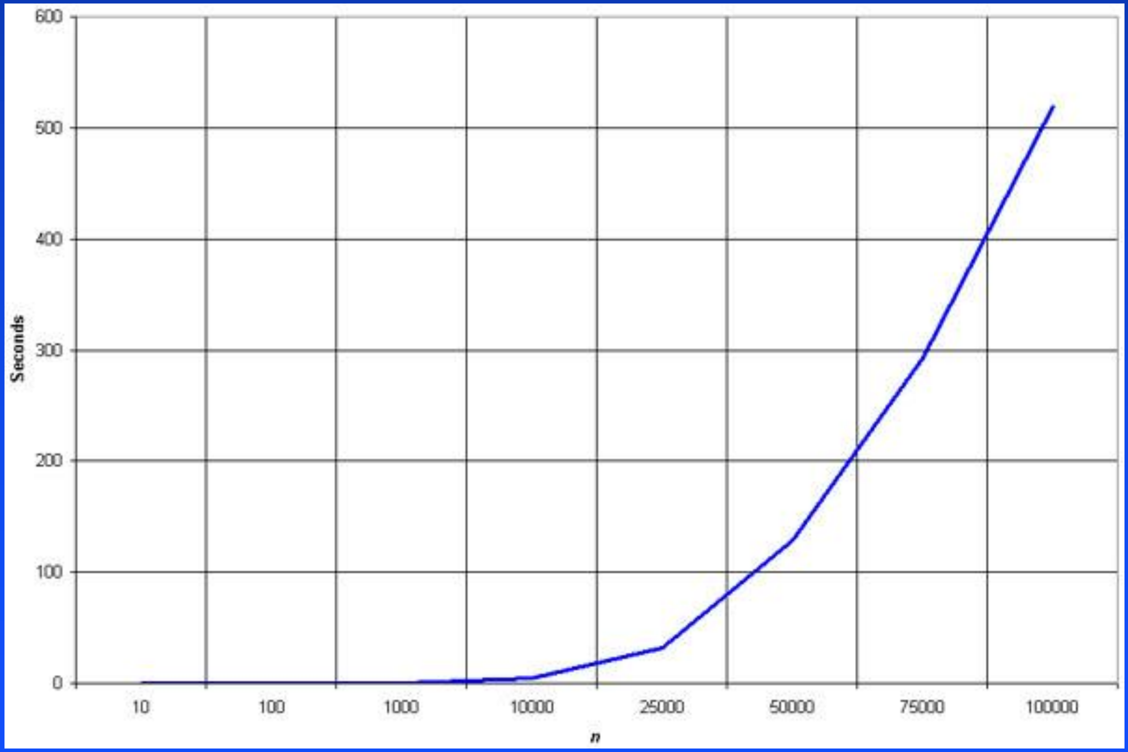
- Total: $(n^2 - n)/2 + (n - 1) = n^2/2 + n/2 - 1 = O(n^2)$

Selection Sort: Efficiency

- The selection sort runs in : $O(N^2)$ time
- But, only needs N time in swap.
- It yields a 60% performance improvement over the bubble sort

Selection Sort: Efficiency

- **Pros:** Simple and easy to implement.
- **Cons:** Inefficient for large lists



Average of a hundred runs against random data sets on a single-user 250MHz UltraSPARC II

Insertion Sort

- It inserts each item into its proper place in the final list.
- Move the current item past the already sorted items and repeatedly copying it with the preceding item until it is in place.
  - Compare two elements
  - If the one on the left is larger, copy to the right
  - Continue to compare another two elements (current one and previous one)
  - If the one on the left is larger, copy to the right
  - Until no more larger the previous data
  - Insert the current item
- Keep making the left side of the array sorted until the whole array is sorted.

Insertion Sort

- There are two basic steps used for the selection sort algorithm:
  1. Compare the items
  2. Copy the items

Insertion Sort : Code

```
public void insertionSort()
    {
       int in, out;

       for(out=1; out<nElems; out++)      // out is dividing line
         {
              long temp = a[out];            // remove marked item
              in = out;                      // start shifts at out
              while(in>0 && a[in-1] >= temp) // until one is smaller,
              {
                     a[in] = a[in-1];           // shift item to right
                      --in;                     // go left one position
              }
                a[in] = temp;                // insert marked item
          }  // end for
      }  // end insertionSort()
```

Insertion Sort : Example

Insertion Sort : Efficiency

- First pass: 1 comparison, maximum 1 copy
- Second pass: 2 comparisons, 2 copies
- Third pass: 3 comparisons, 3 copies
- …
- (n-1)th pass: n-1 comparison, n-1 copies
- Then it's sorted
- So we have (worst case):
  - $(n-1)+(n-2)+\ldots.+1$ comparisons = $n(n-1)/2 = (n^2- n)/2$
  - $(n-1)+(n-2)+\ldots.+1$ copies = $n(n-1)/2 = (n^2- n)/2$
- Total: $2*((n^2- n)/2) = n^2- n = O(n^2)$

Insertion Sort : Efficiency

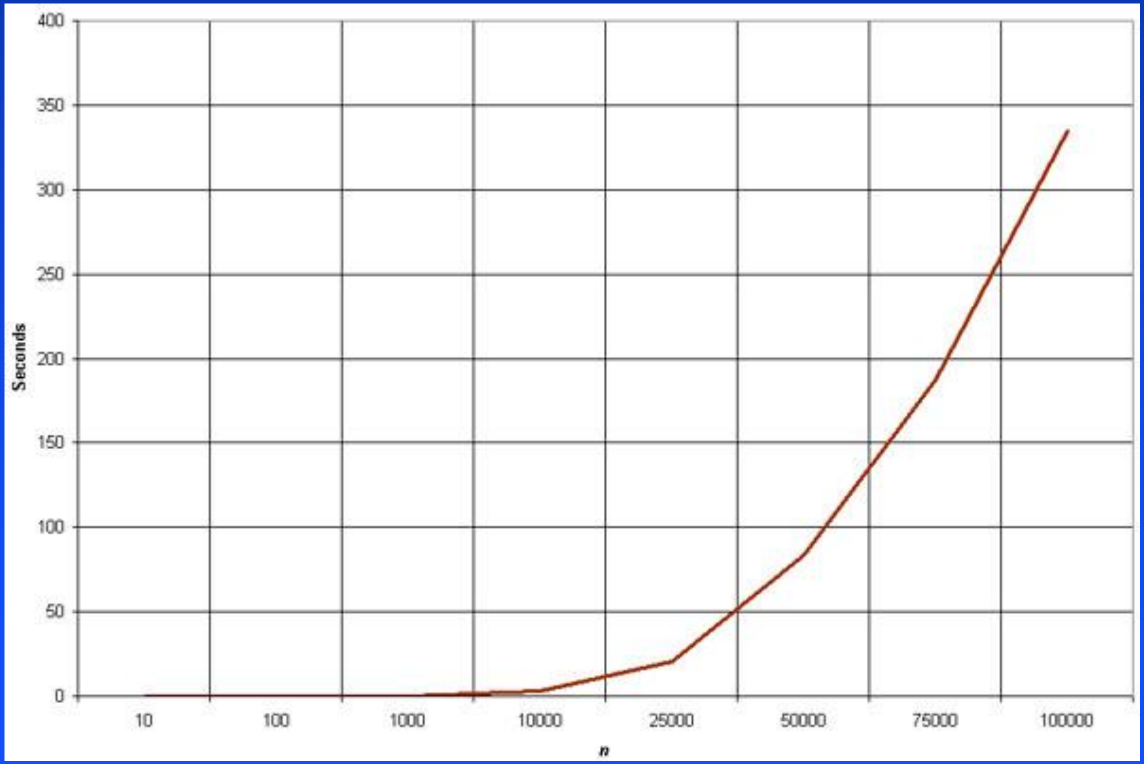
- Comparison Time:
  - The insertion sort runs in $O(N^2)$ time for random data.
  - For data that is already sorted or almost sorted, the insertion sort does much better. So, the best case for the algorithm takes N time.

Insertion Sort : Efficiency

- The insertion sort is over twice as fast as the bubble sort and almost 40% faster than the selection sort.
- The insertion sort shouldn't be used for sorting lists larger than a couple thousand items or repetitive sorting of lists larger than a couple hundred items.

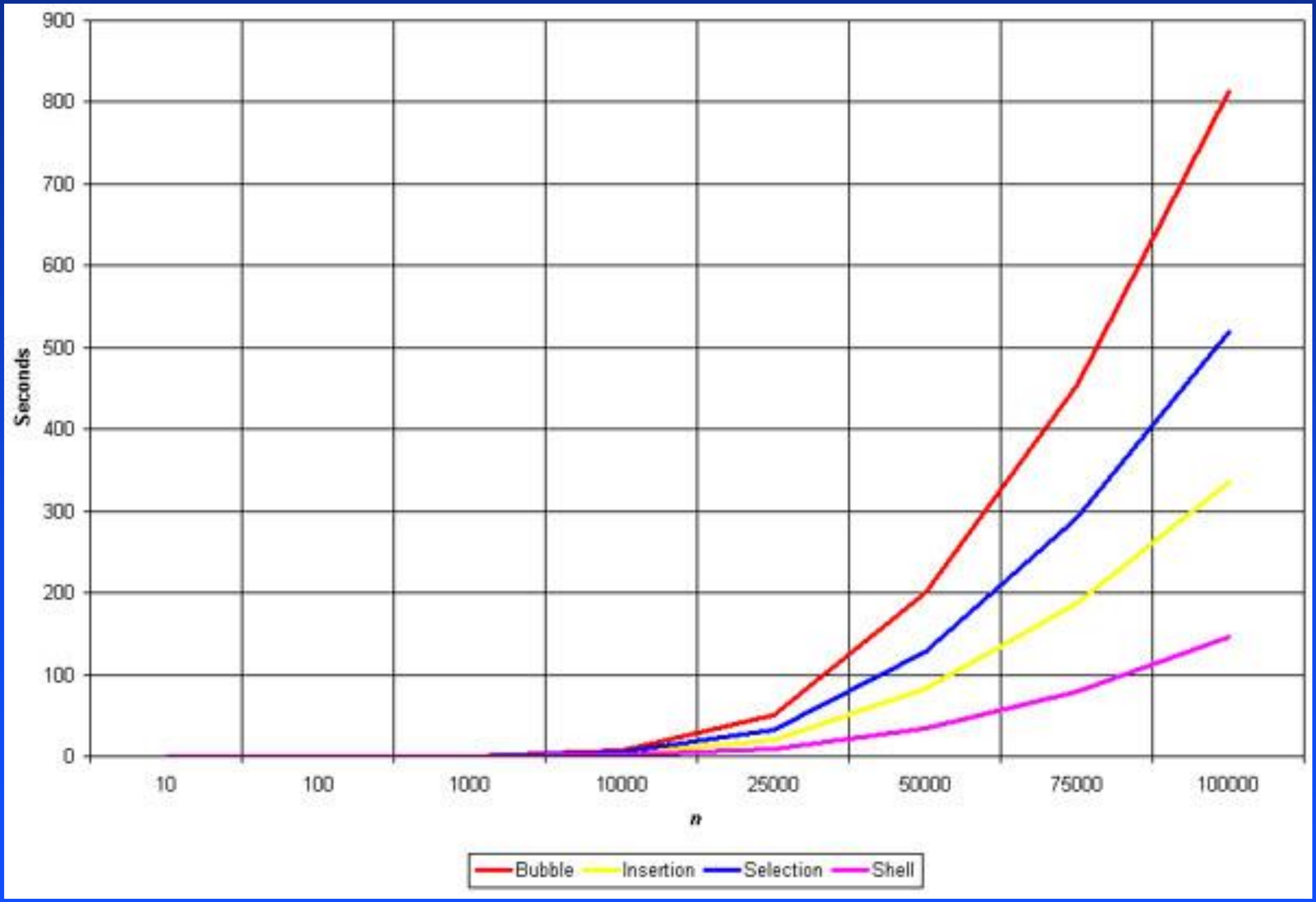
Insertion Sort : Efficiency

- **Pros:** Relatively simple and easy to implement.
- **Cons:** Inefficient for large lists.



Average of a hundred runs against random data sets on a single-user 250MHz UltraSPARC II

Comparison : Efficiency ($O(N^2)$)



Comparison: Summary

- Bubble Sort – hardly ever used
  - Too slow, unless data is very small
- Selection Sort – slightly better
  - Useful if data is quite small and swapping is time-consuming compared to comparisons
- Insertion Sort – most versatile
  - Best in most situations
  - Still for large amounts of highly unsorted data, there are better ways
- Memory requirements are not high for any of these

Comparison of Sorting Algorithms

Sort	Best	Average	Worst
Bubble	$O(N^2)$	$O(N^2)$	$O(N^2)$
Selection	$O(N^2)$	$O(N^2)$	$O(N^2)$
Insertion	$O(N)$	$O(N^2)$	$O(N^2)$