Insertion Sort and Merge Sort

CSE 2011
Winter 2007

Insertion Sort: Main Idea

1) Initially $p = 1$
2) Let the first $p$ elements be sorted
3) Insert the $(p+1)$th element properly in the list so that now $p+1$ elements are sorted.
4) Increment $p$ and go to step (3)
Insertion Sort: Example

<table>
<thead>
<tr>
<th>Original</th>
<th>34</th>
<th>8</th>
<th>64</th>
<th>51</th>
<th>32</th>
<th>21</th>
<th>Positions Moved</th>
</tr>
</thead>
<tbody>
<tr>
<td>After p = 1</td>
<td>8</td>
<td>34</td>
<td>64</td>
<td>51</td>
<td>32</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>After p = 2</td>
<td>8</td>
<td>34</td>
<td>64</td>
<td>51</td>
<td>32</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>After p = 3</td>
<td>8</td>
<td>34</td>
<td>51</td>
<td>64</td>
<td>32</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>After p = 4</td>
<td>8</td>
<td>32</td>
<td>34</td>
<td>51</td>
<td>64</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>After p = 5</td>
<td>8</td>
<td>21</td>
<td>32</td>
<td>34</td>
<td>51</td>
<td>64</td>
<td>4</td>
</tr>
</tbody>
</table>

Insertion Sort: Algorithm

```c
for (int p = 1; p < a.size(); p++)
{
    int tmp = a[p];
    for (j = p; j > 0 && tmp < a[j-1]; j--)
    {
        a[j] = a[j-1];
    }
    a[j] = tmp;
}
```

- Consists of N - 1 passes
- For pass p = 1 through N - 1, ensures that the elements in positions 0 through p are in sorted order
  - elements in positions 0 through p - 1 are already sorted
  - move the element in position p left until its correct place is found among the first p + 1 elements

[See applet](http://www.cis.upenn.edu/~matuszek/cse121-2003/Applets/Chap03/Insertion/InsertSort.html)
Example 2

To sort the following numbers in increasing order:
34  8  64  51  32  21

\[ p = 1; \ tmp = 8; \]
34 > tmp, so second element a[1] is set to 34: \{8, 34\}...
We have reached the front of the list. Thus, 1st position a[0] = tmp=8
After 1st pass: 8  34  64  51  32  21  
(first 2 elements are sorted)

\[ P = 2; \ tmp = 64; \]
34 < 64, so stop at 3rd position and set 3rd position = 64
After 2nd pass: 8  34  64  51  32  21  
(first 3 elements are sorted)

\[ P = 3; \ tmp = 51; \]
51 < 64, so we have 8  34  64  64  32  21,
34 < 51, so stop at 2nd position, set 3rd position = tmp,
After 3rd pass: 8  34  51  64  32  21  
(first 4 elements are sorted)

\[ P = 4; \ tmp = 32; \]
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 stop at 1st position and set 2nd position = 32,
After 4th pass: 8  32  34  51  64  21

\[ P = 5; \ tmp = 21, \ldots \]
After 5th pass: 8  21  32  34  51  64
Analysis: Worst-case Running Time

for (int p=1; p< size; p++)
{
    int tmp=a[p];
    for (j=p; j>0 && tmp < a[j-1]; j--)
        a[j] = a[j-1];
    a[j] = tmp;
}

● Inner loop is executed p times, for each p=1..N-1
  ⇒ Overall: 1 + 2 + 3 + . . . + N-1 = ... = O(N²)
● Space requirement is O(?)

Analysis

● The bound is tight  Θ(N²).
● There exist inputs that actually use Ω(N²) time.
● Consider a reversed sorted list as input:
  ○ When a[p] is inserted into the sorted a[0..p-1], we need to compare a[p] with all elements in a[0..p-1] and move each element one position to the right  ⇒  Ω(i) steps
  ○ The total number of steps is Ω(Σ₁⁻¹ i) = Ω(N(N-1)/2) = Ω(N²)
Analysis: Best-case Running Time

- The input is already sorted in increasing order:
  - When inserting $a[p]$ into the sorted $a[0..p-1]$, only need to compare $a[p]$ with $a[p-1]$ and there is no data movement.
  - For each iteration of the outer for-loop, the inner for-loop terminates after checking the loop condition once $\Rightarrow O(N)$ time

- If input is *nearly sorted*, insertion sort runs fast.

---

Merge Sort
Merge Sort: Main Idea

Based on divide-and-conquer strategy
- Divide the list into two smaller lists of about equal sizes.
- Sort each smaller list recursively.
- Merge the two sorted lists to get one sorted list.

How do we divide the list? How much time needed?
How do we merge the two sorted lists? How much time needed?

Dividing

- If the input list is a linked list, dividing takes $\Theta(N)$ time:
  - Scan the linked list, stop at the $\lfloor N/2 \rfloor$th entry and cut the link.

- If the input list is an array $A[0..N-1]$: dividing takes $O(1)$ time:
  - Represent a sub-array by two integers $left$ and $right$.
  - To divide $A[left .. right]$, compute $center=(left+right)/2$ and obtain $A[left .. center]$ and $A[center+1 .. right]$. 
Merge Sort: Algorithm

- Divide-and-conquer strategy
  - recursively sort the first half and the second half
  - merge the two sorted halves together

```c
void mergesort(int & A[], int left, int right)
{
    if (left < right) {
        int center = (left + right) / 2;
        mergesort(A, left, center);
        mergesort(A, center+1, right);
        merge(A, left, center+1, right);
    }
}
```

[Diagram showing the recursive process of merge sort]

[Link: http://www.cosc.canterbury.ac.nz/people/mukundan/dsal/MSort.html]
Merging

- Input: two sorted array A and B
- Output: an output sorted array C
- Three counters: Actr, Bctr, and Cctr
  - initially set to the beginning of their respective arrays

The smaller of A[Actr] and B[Bctr] is copied to the next entry in C, and the appropriate counters are advanced.

When either input list is exhausted, the remainder of the other list is copied to C.

Merge: Example
Example: Merge (cont’d)

“Merge” algorithm: Figure 7.10, page 261

Merge: Analysis

- Running time analysis:
  - Merge takes $O(m_1 + m_2)$, where $m_1$ and $m_2$ are the sizes of the two sub-arrays.

- Space requirement:
  - merging two sorted lists requires linear extra memory
  - additional work to copy to the temporary array and back
Analysis of Merge Sort

- Let $T(N)$ denote the worst-case running time of **mergesort** to sort $N$ numbers.
- Assume that $N$ is a power of 2.

- Divide step: $O(1)$ time
- Conquer step: $2T(N/2)$ time
- Combine step: $O(N)$ time

- Recurrence equation:
  \[
  T(1) = 1 \\
  T(N) = 2T(N/2) + N
  \]

Solving the Recurrence

\[
T(N) = 2T\left(\frac{N}{2}\right) + N \\
= 2(2T\left(\frac{N}{4}\right) + \frac{N}{2}) + N \\
= 4T\left(\frac{N}{4}\right) + 2N \\
= 4(2T\left(\frac{N}{8}\right) + \frac{N}{4}) + 2N \\
= 8T\left(\frac{N}{8}\right) + 3N = \Lambda \\
= 2^k T\left(\frac{N}{2^k}\right) + kN
\]

Since $N=2^k$, we have $k=\log_2 n$

\[
T(N) = 2^k T\left(\frac{N}{2^k}\right) + kN \\
= N + N \log N \\
= O(N \log N)
\]
Next time …

- Quick Sort
- Lower Bound for Sorting
- Bucket Sort