CS 171: Introduction to Computer Science II

Simple Sorting

Li Xiong
Bonus question

Give the big Oh cost of the following code fragments:

a. int sum = 0;
   for (int i = 1; i < N; i *= 2)
      for (int j = 0; j < N; j++)
         sum++;

b. int sum = 0;
   for (int i = 1; i < N; i *= 2)
      for (int j = 0; j < i; j++)
         sum++;
Useful Approximations

• Harmonic sum
  \[ 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N} \sim \ln N \]

• Triangular sum
  \[ 1 + 2 + 3 + \ldots + N = \frac{N(N+1)}{2} \sim \frac{N^2}{2} \]

• Geometric sum
  \[ 1 + 2 + 4 + \ldots + N = 2N - 1 \sim 2N \text{ when } N = 2^n \]

• Stirling’s approximation
  \[ \lg N! = \lg 1 + \lg 2 + \lg 3 + \ldots + \lg N \sim N \lg N \]
Roadmap

• Algorithm Analysis

• Simple sorting algorithms
  – Bubble sort
  – Selection sort
  – Insertion sort

• Advanced sorting algorithms
### Sorting problem

**Ex.** Student records in a university.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>First Name</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
</tr>
</tbody>
</table>

### Sort

Rearrange array of $N$ items into ascending order.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Name</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
</tr>
</tbody>
</table>
Sorting Problem
Bubble Sort and Selection Sort

• Intuition:
  – Find the biggest (smallest) number
  – Find the second biggest (smallest) number
  – ...

• Bubble sort
  – Repeatedly swapping two adjacent numbers in each pass -> biggest number pushed to the end

• Selection sort
  – Select the biggest (smallest) number in each pass
Bubble Sort

• After one pass, we find the biggest number.

• It’s like the biggest ‘bubble’ floats to the top of the surface, hence the name ‘bubble sort’.
Bubble Sort

• In the second pass, we repeat the same process, but now we only have N-1 numbers to work on.

• The third pass is the same, with only N-2 numbers.

• ...

• Repeat until all players are in order.
Analysis of Bubble Sort

• Number of comparisons?

• Number of swaps?
Analysis of Bubble Sort

• Number of comparisons?
\[ \frac{N(N - 1)}{2} = O(N^2) \]

• Number of swaps?
best case: \( O(1) \)
worst cast: \[ \frac{N(N - 1)}{2} = O(N^2) \]
average: \[ \frac{N(N - 1)}{4} = O(N^2) \]
Selection Sort

1. Keep track of the **index** of the smallest number in each round.
2. Swap the smallest number towards the beginning of the array.
3. Repeat the above two steps.
Two useful sorting abstractions

**Helper functions.** Refer to data through compares and exchanges.

**Less.** Is item $v$ less than $w$?

```java
private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
}
```

**Exchange.** Swap item in array $a[]$ at index $i$ with the one at index $j$.

```java
private static void exch(Comparable[] a, int i, int j) {
    Comparable swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```
Selection Sort

Algorithm. $\uparrow$ scans from left to right.

Invariants.
- Entries the left of $\uparrow$ (including $\uparrow$) fixed and in ascending order.
- No entry to right of $\uparrow$ is smaller than any entry to the left of $\uparrow$. 

in final order $\uparrow$
Selection Sort

- Move the pointer to the right.
  
  i++;

- Identify index of minimum entry on right.
  
  ```
  int min = i;
  for (int j = i+1; j < N; j++)
    if (less(a[j], a[min]))
      min = j;
  ```

- Exchange into position.
  
  ```
  exch(a, i, min);
  ```
public class Selection {
    public static void sort(Comparable[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++) {
            int min = i;
            for (int j = i + 1; j < N; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }

    private static boolean less(Comparable v, Comparable w) {
        /* as before */
    }

    private static void exch(Comparable[] a, int i, int j) {
        /* as before */
    }
}
Selection Sort

• Number of comparisons?

• Number of swaps?
Selection Sort

- Number of comparisons?
  \[ O(N^2) \]

- Number of swaps?
  \[ O(N) \]
Selection Sort

• Online demo

• Gypsy dance demo
  – http://www.youtube.com/watch?v=Ns4TPTC8whw
Insertion Sort

• How do you sort a hand of poker cards?
Insertion Sort

• Idea
  – Assume the left portion of the array is *partially sorted* (however, unlike selection sort, the elements are not necessarily in their final positions)
  – For each remaining element on the right portion, insert it to the left portion (similar to insertion in an ordered array).
  – Repeat until done.
Algorithm. \uparrow scans from left to right.

Invariants.
- Entries to the left of \uparrow (including \uparrow) are in ascending order.
- Entries to the right of \uparrow have not yet been seen.
• Move the pointer to the right.

    ```
    i++;
    ```

• Moving from right to left, exchange $a[i]$ with each larger entry to its left.

    ```
    for (int j = i; j > 0; j--)
        if (less(a[j], a[j-1]))
            exch(a, j, j-1);
        else break;
    ```
public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0; j--)
                if (less(a[j], a[j-1]))
                    exch(a, j, j-1);
                else break;
    }

    private static boolean less(Comparable v, Comparable w)
    { /* as before */ }

    private static void exch(Comparable[] a, int i, int j)
    { /* as before */ }
}
Insertion Sort

• Online demo
  – http://www.sorting-algorithms.com/insertion-sort

• Romanian dance demo
  – http://www.youtube.com/watch?v=ROalU379l3U
Insertion Sort

- Number of comparisons?

- Number of exchanges?
Insertion sort

• Best case
  – N-1 comparisons
  – 0 exchanges

• Worst case
  – \( \sim N^2/2 \) comparisons
  – \( \sim N^2/2 \) exchanges
Summary

• **Bubble sort** uses repeated comparisons and swaps to find the biggest (or smallest) element in each pass

• **Selection sort** selects the smallest element in each pass and reduces the cost of exchanges

• **Insertion sort** inserts each element in the left assorted portion and reduces the cost of comparisons

• All have average comparison cost of $O(N^2)$