Issues In Algorithm Analysis

Many Ways to Skin a Cat

\[
\text{SQ1}(x) \\
\quad \text{return } x \times x
\]

\[
\text{SQ2}(x) \\
\quad \text{ans } \leftarrow 0 \\
\quad \triangleright \text{ Add } x \text{ to ans } x \text{ times.} \\
\quad \text{for } i \leftarrow 1 \text{ to } x \text{ do} \\
\quad \quad \text{ans } \leftarrow \text{ans } + \text{x} \\
\quad \text{return ans}
\]

\[
\text{SQ3}(x) \\
\quad \text{ans } \leftarrow 0 \\
\quad \triangleright \text{ Add 1 to ans } x^2 \text{ times.} \\
\quad \text{for } i \leftarrow 1 \text{ to } x \text{ do} \\
\quad \quad \text{for } j \leftarrow 1 \text{ to } x \text{ do} \\
\quad \quad \quad \text{ans } \leftarrow \text{ans } + 1 \\
\quad \text{return ans}
\]

Choosing a Barometer

What should we count to measure time?

- Number of arithmetic operations (+, *, <, etc.)?
- Number of assignments (\(\leftarrow\)) performed?
- Number of times a line of code is executed?

E.g., suppose we count arithmetic operations:

<table>
<thead>
<tr>
<th>x</th>
<th>SQ1</th>
<th>SQ2</th>
<th>SQ3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Details:

1. Some operations may be more expensive than others!
2. Do we count "hidden" increments, tests, and assignments in \textbf{for} loops?
3. Must pick representative line(s), usually bodies of inner loops.

**Model of Computation**

Running times depend on model of computation!

- Typically assume that numerical operations take constant time.
- Addition would take linear time if model only supported increment operations.
- In practice, operations take time proportional to number of bits (lg n).
- We will generally assume sequential rather than parallel model. (See CS331 for Parallel algorithms.)

![Diagram of addition tree with numbers 17, 8, 23, 42, 6, 37, 19, 91]

---

**Measuring Input Size**

Standard assumptions:

- Size of numerical input is the input itself
- Size of array input is length of array

Not always obvious:

- Size of tree may be number of nodes or height.
- Size of number n may be n or number of bits (lg n).

Can have more than one size:

- Searching for string of length m in text of length n.
- Processing a graph with V vertices and E edges.
Running Time for a Particular Input Size

Running time may differ greatly for different inputs of the same size.

How to characterize?

- **Worst-case analysis**: consider maximum time for every input size.
- **Best-case analysis**: consider minimum time for every input size (not a good idea!).
- **Average-case analysis**: expected running time based on probability distribution of inputs (can be difficult!).

![Graphs showing running time for different inputs of size n]
Example: Insertion Sort

- divide a problem into subproblems
- conquer the subproblems by solving them recursively
- glue the solutions of the subproblems to form the solution of the whole problem

```
InsertionSort(A, k)
  ▷ Sort A[1..k] via insertion sort method.
  ▷ Initially call InsertionSort(A,length[A])
  if n > 0 then ▷ A[1..0] = empty array is trivially sorted; do nothing
    InsertionSort(A, k - 1)
    Insert(A, k)

Insert(A, i)
  ▷ Assume A[1..i-1] is sorted. Make A[1..i] sorted
  if i > 1 then ▷ A[1..i] is trivially sorted; do nothing
    if A[i-1] < A[i] then
      Swap(A, i - 1, i) ▷ Swap contents of A[i-1] and A[i]
      Insert(A, i - 1)
```
Analysis of Insert

Worst-case =

<table>
<thead>
<tr>
<th>i</th>
<th># calls to Insert</th>
<th># &lt;</th>
<th># Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best-case =

<table>
<thead>
<tr>
<th>i</th>
<th># calls to Insert</th>
<th># &lt;</th>
<th># Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average-case =

<table>
<thead>
<tr>
<th>i</th>
<th># calls to Insert</th>
<th># &lt;</th>
<th># Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analysis of InsertionSort

Worst-case =

<table>
<thead>
<tr>
<th>k</th>
<th># Insert</th>
<th># InsertionSort</th>
<th># &lt;</th>
<th># Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best-case =

<table>
<thead>
<tr>
<th>k</th>
<th># Insert</th>
<th># InsertionSort</th>
<th># &lt;</th>
<th># Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average-case =

<table>
<thead>
<tr>
<th>k</th>
<th># Insert</th>
<th># InsertionSort</th>
<th># &lt;</th>
<th># Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another Version of Insertion Sort

CLR version of insertion sort:

- no procedure call overhead.
- iterative algorithm.
- invariant: $A[1..j]$ is in sorted order after every iteration of for loop.
- see CLR for detailed analysis.

InsertionSort(A)
    for $j \leftarrow 2$ to length[A] do
        key $\leftarrow A[j]$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$ do
            $A[i + 1] \leftarrow A[i]$
            $i \leftarrow i - 1$
        $A[i + 1] \leftarrow key$
Merge Sort

**UNSORTED**

**UNSORTED**

**UNSORTED**

**UNSORTED**

**UNSORTED**

**NUOSRTDE**

**NOSUDE**

**DENORSTU**

**Idea:** Divide array into two equal-sized subproblems, recursively sort, then merge results.

```plaintext
MergeSort(A, p, r)
  ▷ Sort A[p..r] by insertion sort method.
  ▷ Initially call MergeSort(A, 1, length[A]).
  if p < r then
    q ← ⌊(p + r) / 2⌋
    MergeSort(A, p, q)
    MergeSort(A, q + 1, r)
    Merge(A, p, q, r)
```

`Merge` left as an exercise. Can be done in linear ($\Theta(n)$) time.

Model running time as the solution to

$$T(n) = 2T(n/2) + \Theta(n), n \geq 1$$

$$T(n) = 0, n < 1$$
The Next Four Lectures

1. **Asymptotic Notation:** coarse-grained comparisons of algorithms.
2. **Recurrences:** coarse-grained analysis of algorithms.
3. **Counting and Probability:** Tools for measuring average cases (also for algorithms that use randomness).
4. **Probabilistic Analysis:** Using probability in analysis.

---

**Pop Quiz**

What is the main topic of this course?

1. Quotes of the previous Vice President.
2. Patterns of growth in water flora.
3. Recipes and resources.