Welcome to 6.046J/18.410: Introduction to Algorithms

Today’s Handouts:
• #1: Course Objectives
• #2: Signup Sheet
• #3: Course Information
• #4: Course Calendar
• #5: Reference List
• #6: Diagnostic Survey
• #7: Kingston (Reading for recitation)

What is an algorithm?

A sequence of elementary computational steps that transforms the input (of some mathematical function) into the output.

Multiplication

Input: Positive Integers $a, b$
Output: $a \times b$.

• Well defined mathematical function.
• How to compute it?
Opening example

Problem: Sorting

Input: Sequence of \( n \) numbers \( \langle a_1, a_2, \ldots, a_n \rangle \).

Output: Permutation \( \langle a'_1, a'_2, \ldots, a'_n \rangle \) such that \( a'_1 \leq a'_2 \leq \cdots \leq a'_n \).

How do we describe an algorithm?

Pseudocode notations

- liberal use of English
- use of indentation for block structure
- employ any clear and concise expressive methods
- typically are not concerned with software engineering issues:
  - error handling
  - data abstraction
  - modularity.
Insertion Sort

Sorts \( A[1..n] \) in place

\[ \text{INSERTION-SORT}(A) \]
\[ \begin{align*}
1 & \text{ for } j \leftarrow 2 \text{ to } \text{length}[A] \\
2 & \quad \text{do } key \leftarrow A[j] \\
3 & \quad \quad \triangleright \text{ Insert } A[j] \text{ into the sorted sequence } A[1..j - 1]. \\
4 & \quad i \leftarrow j - 1 \\
5 & \quad \text{while } i > 0 \text{ and } A[i] > key \\
6 & \quad \quad \text{do } A[i + 1] \leftarrow A[i] \\
7 & \quad \quad i \leftarrow i - 1 \\
8 & \quad A[i + 1] \leftarrow key
\end{align*} \]

Insertion Sort Idea

\[ A[1 .. j - 1] \] — currently sorted part
\[ A[j + 1 .. n] \] — currently unsorted part

- pick element \( A[j] \) (line 2)

Operation of Insertion Sort Algorithm

Running time

- For a given input, we can compute a running time.
- What do we mean by “time”?
  - CPU? Wall-clock?
  - doesn’t it depend on implementation?
Running time

- Depends on
  - input size (e.g. 6 elements vs. 6000)
  - input itself (e.g. partially sorted already)
- Generally want upper bound
- Experiments can suggest performance, but analysis gives...
- Need promise of performance to user

---

Kinds of analysis

- (usually) - Worst case: \( T(n) = \max \text{ time on any input of size } n \)
- (sometimes) - Average case:
  \( T(n) = \text{average time over all inputs of size } n \) (assumes statistical distribution of inputs)
- (never) - Best case: Bad
  Cheat with slow algorithm that works fast on some input. Good only for showing bad lower bound.

---

For insertion sort, what is worst-case time?

Depends on speed of primitive operations

- relative speed (on same machine)
- absolute speed (on different machines)
Asymptotic analysis

- Ignore machine-dependent constants
- Look at growth of $T(n)$ as $n \to \infty$
- $\Theta$ notation
  - Drop low-order terms
  - Ignore leading constants
  
  $3n^3 + 90n^2 - 2n + 5 = \Theta(n^3)$

\[
\begin{aligned}
T(n) & \\
\text{min value for } n_0 & \\
\end{aligned}
\]

better

Insertion Sort Pseudocode

Sorts $A[1..n]$ in place

\begin{algorithm}
\caption{INSERTION-SORT($A$)}
1 \textbf{for} $j \leftarrow 2$ \textbf{to} length[$A$] \\
2 \quad \textbf{do} key $\leftarrow A[j]$ \\
3 \quad \triangleright$ Insert $A[j]$ into the sorted \\
4 \quad \quad \text{sequence } A[1..j-1]$. \\
5 \quad $i \leftarrow j - 1$
6 \quad \textbf{while} $i > 0$ and $A[i] > key$ \\
7 \quad \quad \textbf{do} $A[i+1] \leftarrow A[i]$ \\
8 \quad \quad $i \leftarrow i - 1$
9 \quad $A[i+1] \leftarrow key$
\end{algorithm}

Insertion Sort Analysis

Write time equations by looking at pseudocode:

\textbf{Worst case}: (input reverse sorted)

Inner loop is $\Theta(j)$ for $j$ from 2 to $n$:

\[ T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta\left(\sum_{j=2}^{n} j\right) = \Theta(n^2) \]

(arithmetic series)

(Note casual manipulation of $\Theta$ with $\Sigma$ below.) Why is it OK?
Average case: (all permutations equally likely)
Inner loop is $\Theta(j/2)$

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$

Often, average case not much better than worst case.

Best case: (input already sorted)

$$T(n) = \Theta(n).$$

Is this a fast sorting algorithm?

- Yes, for small $n$.
- No, for large $n$.

Why: small leading constant for a $\Theta(n^2)$ algorithm.

Merge Sort

To sort $n$ numbers:

1. if $n=1$, done.
2. recursively sort 2 lists of numbers with $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ elements, respectively.
3. “merge” 2 sorted lists.

Basic step:

Merge two already sorted lists into one sorted list:

\[
\begin{array}{c}
2 & 3 & 7 & 8 \\
1 & 4 & 5 & 6
\end{array} \quad \begin{array}{c}
1 & 2 & 3 & 4 & 5 \ldots \\
& 2 & 3 & 4 & 5 & \ldots
\end{array}
\]

Running time of `Merge` procedure:

- each step in `Merge` takes constant $\Theta(1)$ time
- $n$ such steps

thus, can merge two sorted lists of total length $n$ in $\Theta(n)$ time.
Recursive algorithm.

Call MERGE-SORT(\(A, 1, n\)) to sort \(A[1 \ldots n]\).

\[
\text{MERGE-SORT}(A, p, r) \quad \triangleright \quad T(n)
\]

if \(p = r\)
then return
\[
\text{MERGE-SORT}(A, p, \lfloor(p + r)/2\rfloor) \quad \triangleright \quad 2T(n/2)
\]
\[
\text{MERGE-SORT}(A, \lceil(p + r)/2\rceil + 1, r)
\]
MERGE results and return \(\triangleright \Theta(n)\)

Recurrence

- describes a function recursively in terms of itself
- describes performance of recursive algorithms

Recurrence for merge sort:

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
2T(n/2) + \Theta(n) & \text{if } n > 1.
\end{cases}
\]

\(2T(n/2)\) should be \(T([n/2]) + T([n/2])\), but this bit of sloppiness turns out not to matter.

How do we find a good upper bound on \(T(n)\) in closed form?

- Generally, will assume \(\exists c\) s.t. \(\forall n < n_0, T(n) < c\), i.e. \(T(n) = \Theta(1)\) for sufficiently small \(n\).

- Write the recurrence

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
2T(n/2) + \Theta(n) & \text{if } n > 1.
\end{cases}
\]

as simply

\[
T(n) = 2T(n/2) + \Theta(n).
\]

(Convention called “integer breakage”.)
Recursion tree

- Model execution to compute running time
- Draw recursion tree, putting in $T()$ at each level then expanding it to $\Theta()$ with $T()$'s under it.

Recursion tree for MERGE-SORT

Note: Total implicitly use $(\log n) \cdot \Theta(n) = \Theta(n \log n)$

Conclusions

- $\Theta(n \log n)$ grows more slowly than $\Theta(n^2)$.
  e.g. for $n = 10^6$: $n^2 = 10^{12}$, $n \log n \approx 2 \cdot 10^7$.
- Therefore merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.