CS140 Pragmatism

What is the asymptotic, worst-case running time of the algorithm?

Types of Algorithms

• Recursive Algorithm: one that calls itself
• Purely Iterative Algorithm: one that doesn’t

Run Time Analysis

• Iterative algorithm → Loop counting
• Recursive algorithm → Recurrence relations

Iterative Sorting Algorithms

• Insertion-sort
• Bubble-sort
• Modified Bubble-sort

Insertion-sort(S)

(in pseudo-code) \( S \) is an array of \( n \) integers: \( S(1), S(2), \ldots, S(n) \)

For \( j = 2 \) to \( n \)
key = \( S(j) \)
i = \( j-1 \)

While \( i > 0 \) and \( S(i) > \text{key} \)
\( S(i+1) = S(i--) \)  \( C \) decrement operator
\( S(i+1) = \text{key} \)

Insertion Sort – Status after each change

<table>
<thead>
<tr>
<th>( j )</th>
<th>( \text{key} )</th>
<th>( i )</th>
<th>( S(1) )</th>
<th>( S(2) )</th>
<th>( S(3) )</th>
<th>( S(4) )</th>
<th>( S(5) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Correctness

• Is Insertion-sort correct?
• Proof by loop invariant:
  When the for loop executes for the k\textsuperscript{th} time, S(1), S(2), …, S(k) are sorted in ascending order.
  (Prove claim by induction on k.)

Loop Counting: Insertion-sort(S)

\[
\sum_{j=2}^{n} \sum_{i=0}^{j-1} (1 + 1 + \cdots + 1) + 1 = O(n^2)
\]

Bubble-sort(S)

Bubble-sort(S)
  For i=n down to 2
    For j=1 to i-1
      If S(j) > S(j+1) then swap(S(j), S(j+1))
  Return

Correctness

• Is Bubble-sort correct?
• Proof by loop invariant:
  When the i-loop completes its k\textsuperscript{th} execution,
  • S(n-k+1), S(n-k+2), …, S(n) is sorted in ascending order, and
  • the max(S(1), …, S(n-k)) ≤ S(n-k+1).

BubbleSort – Status after each change

\[
\begin{array}{|c|c|c|c|c|}
\hline
i & j & S(1) & S(2) & S(3) \hline
\hline
5 & 1 & 3 & 5 & 2 \hline
5 & 1 & 3 & 5 & 2 \hline
5 & 2 & 1 & 3 & 5 \hline
\hline
\end{array}
\]

Does Bubble-sort do too much work?

1, 3, 2, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 4

• Repeat on smaller list
Modified Bubble-sort

Modified-Bubble-sort(S)
SWAP=T
For i=n down to 2
   If SWAP=F then return
   SWAP=F
For j=1 to i -1
   If S(j)>S(j+1) then swap(S(j),S(j+1)) and set
   SWAP=T
Return

Example

1. 3. 2. 4. 5
1. 2. 3. 4. 5
1. 2. 3. 4. 5
• Repeat on smaller list
  unless no swaps are made

Loop counting: M-Bubble-sort

\[
1 + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (3 + \sum_{k=1}^{j-1} 4)) = O(n^2)
\]

Loop Counting: Bubble-sort

• Also O(n^2)

Summation

\[
\sum_{i=2}^{n} \sum_{j=1}^{i-1} c = c \sum_{i=2}^{n} (i-1) = c \sum_{i=2}^{n} i - c \sum_{i=2}^{n} 1 - c \sum_{i=2}^{n} n = O(n^2)
\]

Series

• A series is a summation of terms
  - Arithmetic series: 1+2+...+n
  - Geometric series: 1+a+a^2+...+a^n

\[
\sum_{i=2}^{n} \sum_{j=1}^{i-1} c
\]
Series
Things we want to do:

- Solve exactly
- Bound above or below
- Prove that a solution (or bound) is correct

Closed form solutions to some common series

- \( f(n) = 1 + 2 + ... + n = \frac{n(n+1)}{2} \)
- \( f(n) = 1^2 + 2^2 + ... + n^2 = \frac{(2n^3 + 3n^2 + n)}{6} \)
- \( f(n) = 1 + a + a^2 + ... + a^n = \frac{n}{1-a} \) if \( 0 \leq a < 1 \)
- \( f(n) = 1 + a + a^2 + ... = \frac{(a^{n+1}-1)}{(a-1)} \) else

Upper Bounds on series

For any constant \( k \):

\[
\sum_{i=1}^{n} i^k \leq \sum_{i=\lceil n/2 \rceil}^{n} n^k \geq \sum_{i=\lceil n/2 \rceil}^{n} \lceil n/2 \rceil^k \geq \Omega(n^{k+1})
\]

So \( \sum_{i=1}^{n} i^k = \Theta(n^{k+1}) \)

Series
Things we want to do:

- Solve exactly
- Bound above or below
- Prove that a solution (or bound) is correct
Proving correctness

- Claim: \( \sum_{i=1..n} i^2 = \frac{(2n^3 + 3n^2 + n)}{6} \)
- Claim holds for \( n = 1 \).
- If the claim holds for \( n \) then it holds for \( n+1 \):
  \[
  \sum_{i=1..n+1} i^2 = (n+1)^2 \cdot \frac{(2n^3 + 3n^2 + n)}{6} + \frac{n(n+1)(2n+1)}{6}
  = (2(n+1)^3 + 3(n+1)^2 + (n+1))/6
  \]

Run Time Analysis

- Iterative algorithm \( \rightarrow \) Loop counting
- Recursive algorithm \( \rightarrow \) Recurrence relations

Sort3: A Recursive Algorithm for SIAO

Sort3(S)
If ||S||< 1
  Return: S
Else
  Return: Sort3(S\max-element(S)),max-element(S)

Example: Sort3(3,1,5,2,4)
Sort3(3,1,5,2,4) = Sort3(3,1,2,4),5
= Sort3(1,2,3,4,5)
= Sort3(1),2,3,4,5
= 1,2,3,4,5

Recursive Algorithms

What about Sort3?

Sort3(S)
If ||S||< 1
  Return: S
Else
  Return: Sort3(S\max-element(S)),max-element(S)

Let \( T(n) \) be the running time of Sort3:
\[
T(1) = c_1 \\
T(n) = c_n + T(n-1), \quad n > 1
\]

Recurrence Relations

- Methods to solve or bound:
  - Guess and prove
  - Unwinding
  - Master method
  - WORK TREES
Guess and Prove (bound)

• Guess: \( T(n) = O(n^2) \)

• Proof: We need to show that there exists constants \( c \) and \( M \) such that \( T(n) \leq cn^2 \) for all \( n \geq M \)

\[
T(1) \leq c, \quad \text{provided } c \geq c_1
\]

• Suppose \( T(n-1) \leq c(n-1)^2 \).
  – \( T(n) = c n^2 + T(n-1) \)
  – \( \leq c n^2 + c(n-1)^2 \)
  – \( = c n^2 + c(n^2 - 2n + 1) \)
  – \( = cn^2 - (2c - c_2)n + c \)
  – \( \leq cn^2, \quad \text{provided } c \geq c_2 \) and \( n \geq 1 \)

Guess and Prove cont.

• \( T(n) \leq cn^2 \) for all \( n \geq 1 \), where \( c = \max(c_1, c_2) \)

\[
T(n) \leq O(n^2)
\]

Unwinding

\[
T(n) \leq c_1 n + T(n-1) \\
\leq c_1 n + c_2 (n-1) + T(n-2) \\
\vdots \\
\leq c_1 n + \sum_{i=2}^{n} c_i = (c_1 - c_j) + c_j(n-1)/2 \\
= O(n^2)
\]

Master Theorem

• Read the book

Work Tree

• A rooted tree for algorithm A on input size \( n \):
  – Each node corresponds to a (recursive) call of A
  – An edge from \( u \) to \( v \) represents the fact that the recursive call \( v \) is made from within \( u \).
Example: Sort3(3,1,5,2,4)

Work Tree
- The “work” done at a node is the number of steps performed by the algorithm within the recursive call

Sort3 Work Tree
- Consider a node at level i, where the root is at level 0:
  - What is the input size? n-i
  - What is the work done? \( c(n-i) \)
  - How many nodes are there at level i? 1
  - What is the total work done at level i? \( c(n-i) \)
  - How many levels are there in the tree? n
  - What is the total work done? \( \sum_{i=0}^{n-1} c(n-i) = O(n^2) \)

Merge-sort
Merge-sort(S={s_1, s_2, ... , s_n})
If n=1 return(S)
Else
  S_1 = Merge-sort(s_1, ... , s_{n/2})
  S_2 = Merge-sort(s_{n/2+1}, ... , s_n)
Return Merge(S_1, S_2)

Merge(s_1,s_2,...,s_k; t_1,t_2,...,t_j)
(k>0 and j>0)
- If \( s_i \leq t_i \) then output \( s_1, \) Merge(s_{i+1},...,s_k; t_1,t_2,...,t_j)
- Else output t_1, Merge(s_1, s_2,...,s_k; t_2,...,t_j)

Example: Sort3(3,1,5,2,4)

Input size: 5, work 5c
Input size: 4, work 4c
Input size: 3, work 3c
Input size: 2, work 2c
Input size: 1, work c
Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)

Input: 4,1
Merge-sort(4)

Input: 4

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)=1,4
Merge-sort(3,2)

Input: 3,2
Merge-sort(3)=3
Merge-sort(2)=2
Merge(3,2)=2,3

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)=1,4
Merge-sort(3,2)=3,4
Merge(1,4; 2,3) = 1,2,3,4
Is Merge-sort correct?

- If n=1 then yes
- If n>1 then
  - We can assume Merge-sort(S(1),...,S(\lfloor n/2 \rfloor)) and Merge-sort(S(\lfloor n/2 \rfloor+1),...,S(n)) return correctly sorted lists.
  - So the merge of these lists is a correctly sorted list.

How fast is Merge-sort?

(Assume n=2^m)

- m=0: T(1) = c
- m>0: T(2^m) = 2T(2^{m-1}) + c2^m

Work Tree for Merge-sort

Input Size: 1 (m=0)

Input Size: 2 (m=1)

Input Size: 4 (m=2)

Input Size: n=2^m

A root with two sub-trees
- Root
  - Input Size: n
  - Work: cn
- Each child
  - Roots a work tree with Input Size 2^{m-1}
Work Tree for Merge-sort
Input Size: $n=2^m$

Properties of nodes at level $i$ (root is at level 0):
- Input size: $2^{m-i}$
- Work: $c2^{m-i}$

Properties of level $i$:
- Number of nodes at level $i$: $2^i$
- Total work of nodes at level $i$: $c2^m$

Property of tree:
- Number of levels: $m+1$
- Total work: $c(m+1)2^m$ or $O(n \lg n)$

What if $n \neq \lceil n \rceil$?

- Claim 1: $T(n) = O(T(\lceil n \rceil))$
- Claim 2: $\lceil n \rceil \lg \lceil n \rceil = O(n \lg n)$