Running Time
Divide and Conquer
Masters Method

Algorithms

Topics

- Running Time Analysis
- Sequence
  - Explicit Formula
  - Recursive Formula
- Summation/Series
- Recurrence Relations
- Divide and Conquer
  - Binary Search & Recurrences
  - Masters Method
  - Merge Sort

Worse Case Performance

- Pick an ordering of the input that yields longest possible runtime for this algorithm
  - "Cooking" the input
    - Example: Insertion sort on a list in reverse order
- Analysis gives the worse case performance
- Provides a theoretical bound on the algorithm but may not be the practical answer

Average Case Performance

- Two views
  - Long term average
    - Average over lots of input sets – Amortized Analysis, or
  - Performance for “random” input
- Analysis generally ranges from difficult to very difficult

Best Case Performance

- Pick an ordering of the input that yields shortest possible runtime for this algorithm
  - More “Cooking” the input

Concentrate on Worse Case

- Worse case analysis gives an upper bound on the algorithm's performance
- Worse case performance occurs for a large number of interesting (relevant) cases
- Best case performance is frequently the same for all algorithm of a specific purpose
- Analysis of average-case performance is usually difficult

Arbitrary Algorithms Approaches

- Three approaches
  – Informal Analysis
  – Code Analysis
  – Recurrence Relations

Method 1: Informal Analysis

- Easiest and the least rigorous method
  - least rigorous == unsure
- Arguments may blur the difference between average and worst case

Informal Analysis Example

- Binary search of sorted array for key value
  - Make the current list the entire array
  - Check middle element
    - If == key, done
    - If > key, make current list the upper half of array
    - Otherwise, Make current list the lower half of array
    - Repeat
  - Must eventually find key or report failure(empty list)
  - Halve the array $\lceil \lg n \rceil$ times
    - For n = 100, $\lceil \lg 100 \rceil$ = 7

Method 2: Code Analysis

- Presumes that the code (pseudo code) is correct (!)
- Look for loops and branches
  - Treat function calls as in-line
- Loops:
  - Count the number of passes
- Nested loops
  - Multiplicative

Code Analysis Example

- Selection sort
  - Find the minimum value of current array: loop from 1 to n-1 (compare to next element)
  - Swap minimum into position 1
  - Repeatedly loop over array
    - From 2 to n-1 (n-2 steps)
    - Then from 3 to n-1 (n-3 steps),
    - … n-2 to n-1 (1 step)
  - Total steps = sum {(n-1), (n-2), … , 1} = n(n-1)/2

Method 3: Recurrence Relations

- Many algorithms are specified as recursively
- Write a statement of the amount of work done in terms of
  - Actual # operations for the first step
  - Number of pieces (often 2) into which the input set is divided for further work
  - Multiplier factor, if necessary, on those pieces

Sequence (From CSCI 1900)

- A ***sequence*** is a list of ordered objects.
  - Ex. 1,2,3,2,1,0,1,2,3,1,2
  - Can be finite or infinite
  - Ex 1,0,1,0,1,0,…
  - Elements can be repeated

Formulas

- Explicit Formula, $a_n=2^n+n-1$
  - $a_1=2^1+1-1=2$ (We start with n=1)
  - $a_2=2^2+2-1=5$
  - n^{th} term is defined in terms of n
- Recursive Formula
  - $a_1=0$, $a_2=1$, $a_n=a_{n-1}+a_{n-2}$ $n>1$
  - $a_3=1$, $a_4=2$, $a_5=3$, $a_6=5$
  - Fibonacci Numbers (occur in nature)
  - n^{th} term is defined in terms of n and previous terms

Old 1900 Quiz

Series/Summation Practice

Recurrence Relation Example

- Binary Search
  - First step compares key to one element (the middle one)   => one operation
  - Split the input set in half
  - Do not process both halves, only one => multiplier is one
  - Notation is $T(n)$ for operation count
  - Recurrence relation is: $T(n) = T(n/2) + 3$
  - Now solve the relation for $T(n)$ in an explicit form (no other references to T(anything))

Recurrence Relation Example (cont.)

- Substitution method
  - Apply the relation inductively until T(1) is reached
    - $T(n) = T(n/2) + 3 = T(n/4) + 3 + 3 = T(n/8) + 3 + 3 + 3 = \ldots = T(n/n) + 3 + 3 \ldots + 3$
  - Must be able to evaluate the end condition
    - $T(n/n) = T(1) = 1$, search an array of 1 element
    - Note: assumes n is a power of 2, so round up (apply ceiling function if necessary)
    - so $T(n) = 3 + \ldots + 3$ lg n terms => $T(n) = \lg n$

Recurrence Relation Form

- A recurrence relation is a recursive form of an equation, for example:

$$T(1) = 3$$
$$T(n) = T(n-1) + 2$$

- A recurrence relation can be put into an equivalent closed form without the recursion

Converting Recurrence Relations

- Begin by looking at a series of equations with decreasing values of *n*:

$$T(n) = T(n-1) + 2$$

$$T(n-1) = T(n-2) + 2$$

$$T(n-2) = T(n-3) + 2$$

$$T(n-3) = T(n-4) + 2$$

$$T(n-4) = T(n-5) + 2$$

Converting Recurrence Relations

- Now, we substitute back into the first equation:

$$\mathrm{T}(n) = \mathrm{T}(n-1) + 2$$

$$\mathrm{T}(n) = (\mathrm{T}(n-2) + 2) + 2$$

$$\mathrm{T}(n) = ((\mathrm{T}(n-3) + 2) + 2) + 2$$

$$\mathrm{T}(n) = (((\mathrm{T}(n-4) + 2) + 2) + 2) + 2$$

$$\mathrm{T}(n) = ((((\mathrm{T}(n-5) + 2) + 2) + 2) + 2) + 2$$

Converting Recurrence Relations

- We stop when we get to $T(1)$:

$$
\begin{aligned}
\mathrm{T}(n) &= \mathrm{T}(n-1) + 2 \\
\mathrm{T}(n) &= (\mathrm{T}(n-2) + 2) + 2 \\
&\vdots \\
\mathrm{T}(n) &= (\cdots((\mathrm{T}(1) + 2) + 2)\cdots + 2) + 2
\end{aligned}
$$

- How many "+ 2" terms are there? Notice we increase them with each substitution.

Converting Recurrence Relations

- We must have *n* – 1 of the "+ 2" terms because there was one at the start and we did *n* – 2 substitutions:

$$\mathrm{T}(n) = \mathrm{T}(1) + \sum_{i=1}^{n-1} 2$$

- So, the closed form of the equation is:

$$\mathrm{T}(n) = 3 + 2(n - 1)$$

Recurrence Relation

- A recurrence relation is an equation that defines a sequence recursively (recursive Formula)
- Solving a recurrence relation is finding a closed-form solution (explicit formula)
- You should have seen this when working with Taylor series (Calculus 2)
- Also used in ODE (ordinary differential equations)
- For CS (Running Time)
  - We can call a function recursively to solve problems (Divide and Conquer)
  - Example Binary Search
  - This will yield a running time which is a recurrence Relation
- Some recurrence relations can be solved using a method called Masters Method (we will see today)
- Other methods
  - Characteristic equation
  - Generating functions

Recurrence Relation Example (cont.)

- <u>Master Theorem Method</u>
  - A recipe for solving any recurrence of the form

$$T(n) = aT(n/b) + f(n)$$

  - where
    - a and b are constants with $a \geq 1$ and $b>1$
  - The above recurrence describes the running time of an algorithm that divides a problem set of size n into a pieces to be processed, each of size n/b. The cost of dividing and combining the results is given by f(n)

Recurrence Relation Example (cont.)

- Must memorize and apply 3 cases
- Case 1: $f(n) = O\left(n^{\log_b(a) - \varepsilon}\right)$

  for $\varepsilon > 0$

  $$T(n) = \Theta\left(n^{\log_b(a)}\right)$$

- Case 2:

  $$f(n) = \Theta\left(n^{\log_b(a)}\right)$$

  $$T(n) = \Theta\left(n^{\log_b(a)} \lg n\right)$$

Recurrence Relation Example (cont.)

– Case 3: $f(n) = \Omega\left(n^{\log_b(a)+\varepsilon}\right)$

for $\varepsilon > 0$ and if $a f(n/b) \le c f(n)$ for some c < 1 then

$$T(n) = \Theta\left(f(n)\right)$$

Using the Master Method

- Example 1: Binary Search
  - $T(n) = T(n/2) + 3$
    - $a = 1$ $\quad$ $b = 2$ $\quad$ $f(n) = 3$
      $\log_b a = \lg 1 = 0$
      $n^0 = 3$
    - Case 2: (with $k = 0$)
      $T(n) = \theta(n^0 \lg n) = \theta(\lg n)$

Using the Master Method (cont)

- Example 2:
  - $T(n) = 9T(n/3) + n$
    - $a = 9$ $b = 3$ $f(n) = n$
      $\log_b a = \log_3 9 = 2$ $f(n) = n^{(2-1)}$ $\varepsilon = 1$
    - Case 1
      $T(n) = \theta(n^2)$

Divide and Conquer

- Divide the problem into smaller subproblems
- Conquer the subproblems recursively until the subproblem is small enough to solve
- Example Binary Search, Merge Sort
- Power function

Binary Search (Recurrence)

- Binary Search (array A, first, last, v)
  - If (last <first) output -1
  - Mid = (last + first) /2
  - If A[mid]= v   output mid
  - If A[mid]>v
    - Binary_search (A, first, mid-1,v)
  - Else Binary_search (A, mid+1, last,v)

Binary Search

- Let $T(n)$ be the worst case running time.
- Each iteration of Binary Search halves the size of the data to be checked.
- $T(n)= T(n/2) +1$
- $T(1)=1$, i.e. $\Theta(1)$
- We could use induction to prove that $T(n)= \Theta(\lg n)$.
- We will just use Masters Method
- $T(n)=aT(n/b) + g(n)$
  - Divide and conquer recurrence relation

Masters Method

Let a >=1, b>1 be constants, let f(n) be a function, and let T(n) be defined on the nonnegative integers by the recurrence.

T(n)=aT(n/b) + f(n). Then

- 1) If $f(n) = O(n^{log\_b(a) - \varepsilon})$ for some constant $\varepsilon>0$, then $T(n)=\Theta(n^{log\_b(a)})$
- 2) If $f(n)= \Theta(n^{log\_b(a)})$, then $T(n)=\Theta(n^{log\_b(a)} \lg n)$
- 3) If $f(n) =\Omega(n^{log\_b(a)+ \varepsilon})$ for some constant $\varepsilon>0$, and if af(n/b) <= cf(n) for some constant c<1 and all sufficiently large *n*, then $T(n)=\Theta(f(n))$

T(n)

- $T(n) = T(n/2) + 1$
- B=2, a=1, so b>1, and a >=1
- $\mathrm{Log}_b\ a = \mathrm{Log}_2\ 1 = 0$
- $n^{log\_b(a)} = n^0 = 1$
- $f(n)=1$
- Case 2 applies here $1 \in \Theta(1)$ so
  - $T(n) = \Theta\ (n^{log\_b(a)} \lg n) = \Theta\ (\lg n)$

T(n)

- $T(n) = 9T(n/3) + n$
- B=3, a=9, so b>1, and a >=1
- $\text{Log}_b\ a = \text{Log}_3\ 9 = 2$
- $n^{log\_b(a)} = n^2$
- $f(n)=n$
- Does here $n \in O(n^{2-\varepsilon})$ for some here $\varepsilon>0$?
- Yes take $\varepsilon = .5$, or even $\varepsilon = 1$, but you can not take $\varepsilon = 0$.
- Case 1 applies here $n \in O(n^1)$ using so
  - $T(n) = \Theta(n^{log\_b(a)}) = \Theta(n^2)$

T(n)

- $T(n)= 3T(n/4) + n \lg n$
- B=4, a=3, so b>1, and a >=1
- $\text{Log}_b\, a = \text{Log}_4\, 3 \approx 0.793$
- $n^{\log\_b(a)} \approx n^{0.793}$
- $f(n)=n \lg n$
- Now is $n \lg n \in \Omega\, (n^{0.793 + \varepsilon})$ for $\varepsilon > 0$
- Yes $n \lg n \in \Omega\, (n)$, so take $\varepsilon = 1-0.793 \approx .207$
- Case 3 applies here if we can show af(n/b) <= cf(n) for some constant c<1 and sufficiently large n.
- Well, $af(n/b) = 3\, (n/4) \lg (n/4) = ¾\, n \lg (n/4) <= ¾\, n \lg (n)$
- So when c=3/4<1 , af(n/b) <= cf(n)

T(n)

- $T(n)= 2T(n/2) + n \lg n$
- B=2, a=2, so b>1, and a >=1
- $\text{Log}_b\ a = \text{Log}_2\ 2 = 1$
- $n^{\log\_b(a)} = n^1$
- $f(n)=n\lg n$
- Case 2 is $n \lg n \in \Theta(n^1)$?
  - No
- Case 3 is $n \lg n \in \Omega(n^{1+\varepsilon})$ for $\varepsilon > 0$?
  - No
- We can not use Regular Masters Method here

Karatsuba: Multiplying Integers

- Let A and B be n bit numbers
- $A = (A_1 A_0)$ $B = (B_1 B_0)$
  - A_1, A_0, B_1, B_0 are n/2 bit numbers
  - $A=2^{n/2} A_1 + A_0$
  - $B=2^{n/2} B_1 + B_0$
  - $A*B=(2^{n/2} A_1 + A_0)(2^{n/2} B_1 + B_0)$
    - $= 2^{n/2} 2^{n/2} A_1 B_1 + 2^{n/2} A_0 B_1 + 2^{n/2} A_1 B_0 + A_0 B_0$
    - $T(n)=4T(n/2)+4n$
    - Case 1, n^2
  - $A*B= 2^n A_1 B_1 + 2^{n/2} A_0 B_1 + 2^{n/2} A_1 B_0 + A_0 B_0$
  - $= 2^n A_1 B_1 + 2^{n/2} (A_0 B_1 + A_1 B_0) + A_0 B_0$
  - $=2^n A_1 B_1 + 2^{n/2} \{ (A_1 + A_0) (B_1 + B_0) - A_1 B_1 - A_0 B_0 \} + A_0 B_0$

  - $(2^{n/2}) A_1 B_1 + 2^{n/2}(A_1 + A_0) (B_0 + B_1) + (2^{n/2}-1) A_0 B_0$
  - $T(n)=3T(n/2)+6n$
  - Case 1, $n^{\lg 3} \approx n^{1.585}$
  - Because of overhead of recursion, this is used for long values of n
  - Small values of n, long multiplication used.

Merge Sort Example

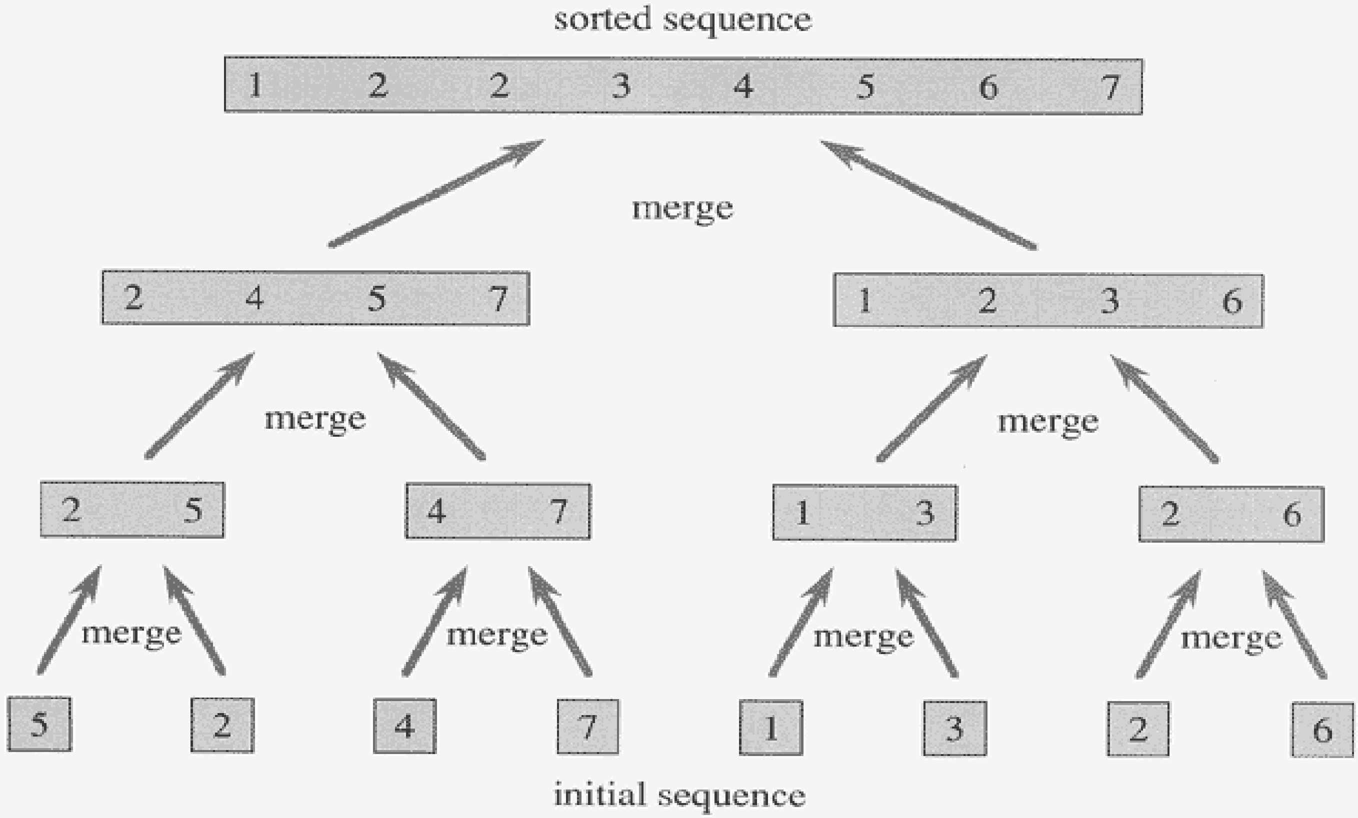

Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.

Quiz T(n) + Merge Sort