CS 341: ALGORITHMS
Lecture 4: divide & conquer
Readings: see website
Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

DIVIDE-AND-CONQUER DESIGN STRATEGY
• divide: Given a problem instance I,
  construct one or more smaller problem instances I_1, ..., I_a
• These are called subproblems
• Usually, want subproblems to be small
  compared to the size of I (e.g., half the size)
• conquer: For 1 ≤ j ≤ a, solve instance I_j recursively,
  obtaining solutions S_1, ..., S_a
• combine: Given solutions S_1, ..., S_a,
  use an appropriate combining function to find
  the solution S to the problem instance I
  i.e., S = Combine(S_1, ..., S_a).

D&C PROTO-ALGORITHM

CORRECTNESS
• Prove base cases are correct
• Inductively assume subproblems are solved correctly
• Show they are correctly assembled into a solution

RUNTIME/SPACE COMPLEXITY?
• Techniques covered in this lecture
  • Model complexities using recurrence relations
  • Solve with substitution, master theorem, etc.
WORKED EXAMPLE: DESIGN OF MERGESORT

Here, a problem instance consists of an array A of n integers, which we want to sort in increasing order. The size of the problem instance is n.

divide: Split A into two subarrays: A_L consists of the first \( \lceil n/2 \rceil \) elements in A and A_R consists of the last \( \lfloor n/2 \rfloor \) elements in A.

conquer: Run Mergesort on A_L and A_R.

combine: After A_L and A_R have been sorted, use a function Merge to merge A_L and A_R into a single sorted array. Recall that this can be done in time \( O(n) \) with a single pass through A_L and A_R. We simply keep track of the “current” element of A_L and A_R, always copying the smaller one into the sorted array.

MERGE: CONQUER AND COMBINE

MERGE SIMULATION

PSEUDOCODE FOR MERGESORT

```
1. Mergesort(A[1..n])
2.   if n == 1 then return A
3.   aL = A[1..\lceil n/2 \rceil]
4.   aR = A[\lceil n/2 \rceil+1..n]
5.   nL = Mergesort(aL)
6.   nR = Mergesort(aR)
7.   return Merge(nL, nR)
```

PSEUDOCODE FOR MERGE

```
Merge(A[1..m], B[1..n])
1. \( a_{out} \) empty array
2. \( l = 1 \), \( r = 1 \)
3. while \( l \leq m \) and \( r \leq n \)
4.   if \( A[l] \leq B[r] \)
5.     \( a_{out} \leftarrow A[l] \)
6.     \( l \leftarrow l + 1 \)
7.   else if \( A[l] \geq B[r] \)
8.     \( a_{out} \leftarrow B[r] \)
9.     \( r \leftarrow r + 1 \)
10. return \( a_{out} \)
```
ANALYSIS OF MERGESORT

1. MergeSort(A[1..n]) = O(n)
2. if n == 1 then return A
3. nl = ceiling(n/2)
4. al = A[1..nl]
5. ar = A[nl+1..n]
6. sl = MergeSort(al)
7. sr = MergeSort(ar)
8. return Merge(sl, sr)

So, MergeSort(A) takes O(n) time plus the time for its two recursive calls.

How can we analyze this recursive program structure?

RECURSION RELATIONS

A crucial analysis tool for recursive algorithms.

MATHEMATICALLY EXPRESSING THE COMPLEXITY OF MERGESORT

Let T(n) denote the time to run MergeSort on an array of length n.

divide takes time Θ(n)
conquer takes time T([n/2]) + T([n/2])
combine takes time Θ(n)

Recurrence relation:

\[ T(n) = \begin{cases} T\left(\left\lceil \frac{n}{2} \right\rceil \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + O(n) & \text{if } n > 1 \\ Θ(1) & \text{if } n = 1 \end{cases} \]

T(n) is a function of T(\(\lceil n/2 \rceil\)) so T is a recurrence relation.

How can we compute/solve for T(n)?

Recursion tree method

RECURSION TREE METHOD

Evaluating recurrences with T(n/2) terms

If parts were parts, would it wear them like this?

Compare vs.

Recursion tree

Level | # of modes | runtime per node | total runtime for level
--- | --- | --- | ---
0 | 1 | cn | cn
1 | 2 | c(n/2) | 2c(n/2) = cn
2 | 2 | c(n/4) | 4c(n/4) = cn
... | ... | ... | ...
\log n | 1 | c(1/2^n) | n(1/2^n) = cn

Total = cn + # levels
Total = cn log(n)

So, mergesort has runtime \(\Theta(n \log n)\).

Can also compute using a table...

O(1) O(1) O(n)

???

O(n)

Hulk(n) = Face - Chin + Hulk(n-1)
Sample recurrence for two recursive calls on problem size $n/2$:

$$T(n) = \frac{2T(\frac{n}{2}) + cn}{d}$$

where $c$ and $d$ are constants.

We can solve this recurrence relation when $n$ is a power of two, by constructing a recursion tree, as follows:

**Step 1:** Start with a single node, say $N_0$, having the value $T(n)$.
**Step 2:** Grow two children of $N_0$. These children, say $N_1$ and $N_2$, have the value $T(n/2)$, and the value of $N$ is replaced by $cn$. Repeat this process recursively, terminating when a node reaches the value $T(1) = d$.
**Step 3:** Sum the values on each level of the tree, and then compute the sum of all these sums; the result is $T(n)$.

**SUBSTITUTION METHOD: WORKED EXAMPLE**

Recurrence: $T(0) = 4; \quad T(n) = T(n-1) + 6n - 5$

* $T(n) = (T(n-2) + 6(n-1) - 5) + 6n - 5$ (substitute)
* $= T(n-2) + 6n - 6 + 6n - 5$ (compare new terms)
* $= (T(n-3) + 6(n-2) - 5) + 2(6n - 5) - 6$ (substitute)
* $= T(n-3) + 6n - 5 + 2(6n - 5) - 6$ (new terms)

... identify patterns and guess what happens in the limit
* $= T(0) + m(6n - 5) - 6(1 + 2 + 3 + \cdots + (n-1)) = guess(n)$

**PROOF**

Recall: $T(0) = 4; T(n) = T(n-1) + 6n - 5$; $guess(n) = 3n^2 - 2n + 4$

Want to prove: $guess(n) = T(n)$ for all $n$

Base case: $guess(0) = 3(0)^2 - 2(0) + 4 = T(0)$

Inductive case: suppose $guess(n) = T(n)$ for $n \geq 0$,

show $guess(n+1) = T(n+1)$,

* $T(n+1) = T(n) + 6(n+1) - 5$ (by definition)
* $= guess(n) + 6(n+1) - 5$ (by inductive hypothesis)
* $= 3n^2 + 4n + 5$ (substitute & simplify)

$guess(n+1) = 3(n+1)^2 - 2(n+1) + 4$ (by definition)

$= 3n^2 + 4n + 5 = T(n+1)$ (simplify)

**ANOTHER APPROACH**

Suppose you look for a while at the previous recurrence:

* $T(0) = 4; T(n) = T(n-1) + 6n - 5$

With some experience, you might just guess it’s quadratic

If you’re right, it should have the form:

$an^2 + bn + c$ for some unknown constants $a$, $b$, $c$

So, just carry the unknown constants into the proof!

You can then determine what the constants must be for the proof to work out.
**MASTER THEOREM FOR RECURRENCES**

- Provides a formula for solving many recurrence relations
- We start with a simplified version
- Consider recurrence: \( T(n) = aT(n/b) + \Theta(n^k) \)
  where \( a \geq 1, b \geq 2 \) and \( n = b^i \) for integer \( i \)

Example corresponding algorithm:

```plaintext
if BaseCase(n) return Result(n)
for i = 1 to log_b n
    let x = subproblem of size n/b
    solution[i] = combine in n^i time
return solution
```

**REARRANGING**

- \( T(n) = da^i + \sum_{i=0}^{k-1} c_i (n/b)^i \)
  - \( x = \log_a n \) \# of subproblems to their size
  - Rearranging we have \( d = x \)
  - \( d = a_i + c_i \sum_{i=0}^{k-1} (n/b)^i \)
  - \( d = a_i + c_i \sum_{i=0}^{k-1} (b^{-i})^i \)
  - \( d = a_i + c_i \sum_{i=0}^{k-1} (b^{-i})^i \)

**SOLVING THE GEOMETRIC SEQ**

- \( T(n) = dn^x + c_i \sum_{i=0}^{k-1} r^i \)
  - Recall formula: \( \sum_{i=0}^{k-1} r^i = \frac{r^{k-1} - 1}{r - 1} \) if \( r > 1 \)
  - \( \sum_{i=0}^{k-1} r^i = \frac{r^{k-1} - 1}{r - 1} \)
  - \( \sum_{i=0}^{k-1} r^i = \frac{r^{k-1} - 1}{r - 1} \)

- So different solutions depending on \( r \)
  - Case 1: \( r = b^{x+y} > 1 \) \( \iff \) \( x > y \) \( \iff \) \( x > y \)
  - Case 2: \( r = b^{-x} < 1 \) \( \iff \) \( x > y \) \( \iff \) \( y < x \)
  - Case 3: \( 0 < r = b^{x+y} < 1 \) \( \iff \) \( x < y \) \( \iff \) \( x < y \)
\[ \sigma_i = 0 \quad \text{for} \quad i = j - 1 \quad \Rightarrow \quad r_i \Delta \sigma_i \]

\[ \begin{align*}
\sigma_i &= 0 \\
\sigma_j &= 0 \\
\sigma_k &= 0
\end{align*} \]

Case 1: \( r = b^{n^y} > 1 \) \( \Rightarrow \) \( x - y > 0 \) \( \Rightarrow \) \( x > y \)

\[ T(n) = dn^n + cn^y \sum_{i=0}^{r-1} r^i \in \Theta(dn^n + cn^y) \]

\[ T(n) \in \Theta(n^2 + n^y) \]

Recall \( b^i = n \), so \( T(n) \in \Theta(n^2 + n^y) = \Theta(n^2 + n^{y+1}) \)

So \( T(n) \in \Theta(n^2) \)

\[ \sum_{i=0}^{r-1} r^i \in \Theta(n^y) \]

\[ x = \log_b a \]

Note that the base case constant \( d \) is not present in any of these complexities!

### Master Theorem for Recurrences

- Simplified version

Consider recurrence:

\( T(n) = aT(\frac{n}{b}) + \Theta(n^y) \) where \( a \geq 1, b \geq 2 \) and \( n = b^i \)

And let \( x = \log_b a \)

\[ T(n) \begin{cases} 
\Theta(n^x) & \text{if } y < x \\
\Theta(n^x \log n) & \text{if } y = x \\
\Theta(n^y) & \text{if } y > x 
\end{cases} \]

\[ \text{Recall: simplified master theorem} \]

Suppose that \( a \geq 1 \) and \( b > 1 \). Consider the recurrence

\[ T(n) = aT(\frac{n}{b}) + \Theta(n^y) \]

Denote \( z = \log_b n \)

\[ T(n) = \begin{cases} 
\Theta(n^x) & \text{if } y < x \\
\Theta(n^x \log z) & \text{if } y = x \\
\Theta(n^y) & \text{if } y > x 
\end{cases} \]

Questions: \( a = 2, b = 2, y = 2 \) \( x = 2 \) which \( \Theta \) function?
Arbitrary function of $n$ (not just $c n^q$)

Must reason about relationship between $f(n)$ and $n^q$

REVISITING THE RECURSION TREE METHOD

- Some recurrences with complex $f(n)$ functions (such as $f(n) = \log n$) can still be solved “by hand”

Example: Let $a = 3^x$: $T(1) = 1; T(n) = 3T(n/3) + n \log n$

| level $j$ | nodes | value at each node | value of the level
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$j = 1$</td>
<td>$n/3$</td>
<td>$1$</td>
<td>$3^1 = 3$</td>
</tr>
<tr>
<td>$j = 2$</td>
<td>$n/9$</td>
<td>$2^1$</td>
<td>$3^2 = 9$</td>
</tr>
<tr>
<td>$j = 3$</td>
<td>$n/27$</td>
<td>$2^2$</td>
<td>$3^3 = 27$</td>
</tr>
<tr>
<td>$j = 4$</td>
<td>$n/81$</td>
<td>$2^3$</td>
<td>$3^4 = 81$</td>
</tr>
<tr>
<td>$j = 5$</td>
<td>$n/243$</td>
<td>$2^4$</td>
<td>$3^5 = 243$</td>
</tr>
</tbody>
</table>

Note: $\log_{3^n} = j$

So $j^2 = n \log_{3^n} n$

And $j^2 - j^{-1} = n \log_{3^n} n$

If $n$, $b$ is not always an integer
- Floors/ceilings are hard
- Not a geometric sequence

Suppose we get a big-O bound for $b^{-1} < n < b^j$
- By instead considering the larger problem size $b^j$

So $T(n) \leq T(b^j)$

- If $y < x$, $T(n) \in \Theta(b^j)$
- If $y = x$, $T(n) \in \Theta\left(b^j \log b^j\right)$
- If $y > x$, $T(n) \in \Theta\left(b^j \log^2 b^j\right)$

MASTER THEOREM WHEN $b^{1/2} < n < b^j$

- $T(n) \leq T\left(b^{1/2}\right)$
- $T(n) \in \Theta\left(b^{1/2}\right)$
- $T(n) \leq T\left(b^j\right)$
- $T(n) \in \Theta\left(b^j\right)$

Observation: $b^j < b^n$ since $n$ is between $b^{1/2}$ and $b^j$

So $T(n) \leq T\left(b^{1/2}\right)$

- if $y < x$, $T(n) \in \Theta\left(b^{1/2}\right)$
- if $y = x$, $T(n) \in \Theta\left(b^{1/2} \log b^{1/2}\right)$
- if $y > x$, $T(n) \in \Theta\left(b^{1/2} \log^2 b^{1/2}\right)$

Can tackle $\Omega$ similarly to get $\Theta$