Divide and Conquer (and solving recurrences)

You've seen (in 1st year & 240) quite a few examples of divide and conquer:

- divide - break the problem into smaller problems
- recurse - solve the smaller subproblems
- conquer - combine the solutions to get a soln to whole problem.

Examples:
- binary search - search in a sorted array for an element e
- try middle, recurse on first half or second half

There is only one subproblem and no "conquer" step.

Let $T(n) = \text{max runtime on array of length } n$,

$$T(n) = 1 + T\left(\frac{n}{2}\right)$$

Actually, $T(n) = 1 + \max\left\{ T\left(\frac{n}{2}\right), T\left(\frac{n-1}{2}\right) \right\}$

and the solution (as you know) is $T(n) \in O(\log n)$.

- sorting
- mergesort - easy divide, $O(n)$ work to conquer
- quicksort - $O(n)$ work to divide, easy conquer

Mergesort recurrence:

$T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n$

$T(n) \in O(n \log n)$
Solving Recurrence Relations

Two basic approaches

- recursion tree method.
- guess a solution and prove correct by induction.

Recursion tree method for mergesort recurrence:

\[ T(n) = 2T\left(\frac{n}{2}\right) + cn \text{, } n \text{ even} \]

\[ T(1) = 0 \text{ (if we count comparisons, else } T(1) = c) \]

So for \( n \) a power of 2,

\[ T(n) \quad \text{---} \quad Cn \]

\[ T\left(\frac{n}{2}\right) \quad \text{---} \quad C\frac{n}{2} \]

\[ T\left(\frac{n}{4}\right) \quad \text{---} \quad C\frac{n}{4} \quad \text{---} \quad C\frac{n}{4} \quad \text{---} \quad C\frac{n}{4} \quad \text{---} \quad 0 \]

Total Sum \( Cn \log n \)

Caution: Even something this simple gets complicated if we are precise.

\[ (*) \quad T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2} - 1\right) \]

Solve: \( T(n) = n \log n - 2 \log n + 1 \) but not trivial.

Luckily we often only want the rate of growth and runtimes are usually increasing.

E.g., \( T(n) \leq T(n') \) \( n' = \text{smallest power of 2 bigger than } n \).

Note: \( n' \leq 2n \).

For mergesort, this gives \( T(n) \in O(n \log n) \).
Guess and prove by induction for mergesort recurrence

prove $T(n) \leq c \cdot n \log n$ by induction $\forall n \geq 2$ for (*).

Separating into odd and even $n$ - this is one way to be rigorous about floor and ceiling.

**basis.** $n = 2$  $T(2) = 2T(1) + 1 = 1$  $c \cdot n \log n = 2c$ for $n = 2$

So $T(n) \leq c \cdot n \log n$ for $n = 2$ if $c \geq \frac{1}{2}$

**basis of $n = 1$ would suffice** $T(1) = 0$  $c \cdot n \log n = 0$

**induction step**

$n$ even $T(n) = 2T\left(\frac{n}{2}\right) + n - 1$

$\leq 2c \cdot \frac{n}{2} \log \frac{n}{2} + n - 1$

by induction

$= c \cdot n \log n + n - 1 = c \cdot n (\log n - 1) + n - 1$

$= c \cdot n \log n - c \cdot n + n - 1 \leq c \cdot n \log n$ if $c \geq 1$.

$n$ odd $T(n) = T\left(\frac{n-1}{2}\right) + T\left(\frac{n+1}{2}\right) + n - 1$

by induction

$c \cdot \left(\frac{n-1}{2}\right) \log \frac{n-1}{2} + c \cdot \left(\frac{n+1}{2}\right) \log \frac{n+1}{2} + n - 1$

---

**CAUTION. What's wrong with this?**

$T(n) = 2T\left(\frac{n}{2}\right) + n$

Claim: $T(n) \in O(n)$

**Proof** $T(n) \leq c \cdot n$ $\forall n \geq n_0$

Assume by induction $T(n') \leq c \cdot n'$ $\forall n' < n$, $n' \geq n_0$.

Then $T(n) = 2T\left(\frac{n}{2}\right) + n$

$\leq 2 \cdot c \cdot \frac{n}{2} + n$ by induction

$= \frac{(c+1)n}{2}$

So $T(n) \in O(n)$ - false

constant — growing constant
Example — changing the induction hypothesis

\[ T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + 1 \]

\[ T(1) = 1 \]

Guess \( T(n) \in O(n) \)

Prove by induction: \( T(n) \leq c \cdot n \) for some \( c \)

\[ T(n) \leq c \left\lfloor \frac{n}{2} \right\rfloor + c \left\lceil \frac{n}{2} \right\rceil + 1 = c \cdot n + 1 \quad \text{whoops!} \]

So is the guess wrong?

No, e.g. \( n \) a power of 2 gives

\[ T(n) = 2T\left(\frac{n}{2}\right) + 1 = 4T\left(\frac{n}{4}\right) + 2 + 1 = \ldots \]

\[ = 2^k T\left(\frac{n}{2^k}\right) + (2^{k-1} + \ldots + 2 + 1) \quad n = 2^k \]

\[ = 2^k + 2^{k-1} + \ldots + 2 + 1 = 2^{k+1} - 1 = 2n - 1 \]

Try to prove by induction: \( T(n) \leq c \cdot n - 1 \)

\[ T(n) \leq c \cdot \left\lfloor \frac{n}{2} \right\rfloor - 1 + c \cdot \left\lceil \frac{n}{2} \right\rceil - 1 + 1 = c \cdot n - 1 \]

So, curiously, we make the induction work by lowering the bound.
Example - changing variables

\[ T(n) = 2T(\sqrt{n}) + \log n \]

Let \( m = \log n \) so \( n = 2^m \)

\[ T(2^m) = 2T(2^{m/2}) + m \]

Let \( S(m) = T(2^m) \)

so \( S(m/2) = T(2^{m/2}) \)

Then \( S(m) = 2S(m/2) + m \) which we know \( S(m) \in O(m \log m) \) so \( T(2^m) = O(m \log m) \)

\[ T(n) = O(\log n (\log \log n)) \]
We often get recurrences of the form
\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k \]

**Note:** the recurrence relations you studied in MATH 239 were homogenous i.e. last term was 0.

This arises if we divide problem of size \( n \) into

@ subproblems of size \( \frac{n}{b} \) and do \( c \cdot n^k \) extra work.

E.g., \( k = 1 \)

\[ a = b = 2 \]

\[ T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n \] mergesort

\[ O(n \log n) \]

\[ a = 1 \quad b = 2 \]

\[ T(n) = T\left(\frac{n}{2}\right) + c \cdot n \] \( O(n) \)

\[ a = 4 \quad b = 2 \]

\[ T(n) = 4 \cdot T\left(\frac{n}{2}\right) + c \cdot n \] \( O(n^2) \)
Theorem ("Master Theorem")

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^k \]

\(a \geq 1, b > 1, c > 0, k \geq 1\) (corrected from class)

Then

\[ T(n) \in \begin{cases} 
\Theta(n^k) & \text{if } a < b^k \text{ i.e. } \log_b a < k \\
\Theta(n^k \log n) & \text{if } a = b^k \\
\Theta(n^{\log_b a}) & \text{if } a > b^k 
\end{cases} \]

Notes

- CLRS has more general version with \(f(n)\) in place of \(c \cdot n^k\)
- You are not responsible for the proof but must know & apply the theorem

A rigorous proof is by induction.

We'll just make sense of it using recursion tree (written out)

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^k \]

\[ = a \cdot a \cdot T \left( \frac{n}{b^2} \right) + c \cdot \left( \frac{n}{b} \right)^k \]

\[ = a^2 \cdot T \left( \frac{n}{b^2} \right) + a \cdot c \cdot \left( \frac{n}{b^2} \right)^k \]

\[ = \ldots = a^3 \cdot T \left( \frac{n}{b^3} \right) + a^2 \cdot c \cdot \left( \frac{n}{b^3} \right)^k \]

\[ = \ldots = a^{\log_b n} \cdot T(1) + \sum_{i=0}^{\log_b n - 1} a^i \cdot c \cdot \left( \frac{n}{b^i} \right)^k \]

\[ a \log_b n = \log_b a \]

change base of log

\[ \frac{n}{b^i} = 1 \text{ so } i = \log_b n \]

\[ = n \log_b a \cdot T(1) + c \cdot n^k \sum_{i=0}^{\log_b n - 1} \left( \frac{a}{b^k} \right)^i \]
• If \( a < b^k \) i.e. \( \log_b a < k \) then \( \sum (\frac{a}{b^k})^i \) is a geometric series with \( \frac{a}{b^k} < 1 \) so \( \sum \) is constant and
  \[ T(n) = n \log_b a \cdot T(1) + \Theta(n^k) \]
  \[ T(n) = \Theta(n^k) \]

• If \( a = b^k \) then
  \[ \sum_{i=0}^{\log_b n - 1} (\frac{a}{b^k})^i = \sum_{i=0}^{\log_b n - 1} 1 = \Theta(\log_b n) \]
  so \( T(n) = n \log_b a \cdot T(1) + c \cdot n^k \cdot (\Theta(\log_b n)) \)
  \[ T(n) = \Theta(n^k \log n) \]

• If \( a > b^k \) then
  \[ \sum (\frac{a}{b^k})^i \] is a geometric series with \( \frac{a}{b^k} > 1 \) so the last term dominates (check this out)
  \[ T(n) = n \log_b a \cdot T(1) + \Theta\left(n^k \left(\frac{a}{b^k}\right) \log_b n\right) \]
  \[ = \Theta\left(a \log_b n \cdot \frac{n^k}{(\log_b n)^k}\right) = n^k \]
  \[ = \Theta\left(n \log_b a\right) \] terms balance
  \[ T(n) = \Theta\left(n \log_b a\right) \]
Divide and Conquer Examples

Counting Inversions

Some websites try to match your preferences (for music, movies, books) with others.

How do you compare two rankings?

E.g., I like best B D C A weakest

You like A D B C

Count: how many pairs do we rank differently?

i.e. how many pairs of lines cross (out of 6 pairs)

BD BA DA CA

= \frac{4 \cdot 3}{2} = 6

(The pairs that don’t cross are BC DC)

Equivalently my ranking 1 2 3 4

Your ranking 4 2 1 3

and we count \# inversions = \# pairs out of order in 2nd list.

Brute Force: check all \( \binom{n}{2} \) pairs \( O(n^2) \)

Does sorting help? Doesn’t seem to.

Better with divide and conquer

Given list \( a_1 \ldots a_n \), count \# inversions.

Divide list in two \( m = \lceil \frac{n}{2} \rceil \)

\( A = a_1 \ldots a_m \) \( B = a_{m+1} \ldots a_n \)

Recursively count \# inversions in each half, \( r_A, r_B \)
Combine: \( \text{answer} \leftarrow r_A + r_B + r \)

\[ r = \# \text{inversions with one element in } A, \text{ one in } B \]
\[ = \# \text{pairs } a_i, a_j \text{ such that } a_i \in A, a_j \in B, a_i > a_j \]

How do we find \( r \)?

Can we count, for each \( a_j \in B \), how many larger elements are there in \( A \)? \(- r_j \)

then \( r = \sum_{a_j \in B} r_j \)

Think about: mergeSort; sort \( A \), sort \( B \), merge

When \( a_j \) is output to merged list \( r_j \leftarrow k \)

Whole algorithm

\text{Sort-and-Count} (L) - returns sorted \( L \), \#inversions

* divide \( L \) into \( A \), \( B \)
  * first half, second half
* \( (r_A, A) \leftarrow \text{sort-and-count} \; (A) \)
* \( (r_B, B) \leftarrow \text{sort-and-count} \; (B) \)
* \( r \leftarrow 0 \)
* do merge of \( A \) and \( B \)
  * When element is moved from \( B \) to output
    \( r \leftarrow r + \# \text{elements remaining in } A \)
  * return \( (r_A + r_B + r, \text{merged list}) \)
\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) \]

Solution: \( T(n) = O(n \log n) \) (as in merge sort)

Question: Is there a better algorithm?

\( O(n \log n / \log \log n) \) 189

\( O(n \sqrt{\log n}) \) 2010 Timothy Chan et al.
using techniques/model where sorting is \( O(n \log n) \)