COMPSCI 311: Introduction to Algorithms  
Lecture 9: Divide and Conquer

Marius Minea
University of Massachusetts Amherst

slides credit: Dan Sheldon

Divide and Conquer: Recipe

▶ Divide problem into several parts
▶ Solve each part recursively
▶ Combine solutions to sub-problems into overall solution

Comparison

Greedy | Divide and Conquer
--- | ---
Formulate problem | ? | ?
Design algorithm | easy | hard?
Prove correctness | hard | easy
Analyze running time | easy | hard?

A Classic Algorithm: Mergesort

```plaintext
MergeSort(List)
▷ complexity \( T(n) = ? \)
if List.length = 1 then
  return List  ▷ Base case
else
  split List in halves List1 and List2  ▷ \( \Theta(n) \)
  MergeSort(List1) ▷ \( T(n/2) \)
  MergeSort(List2) ▷ \( T(n/2) \)
  return Merge(List1, List2) ▷ \( \Theta(n) \)
end if
```

Clicker Question 1

How many total calls to MergeSort are made? (choose best answer)  
(Base case: \( n = 1 \))

A) \( \lceil \log n \rceil \)
B) At most \( n \)
C) At most \( 2n \)
D) At most \( n^2 \)

Recurrence

▶ Recurrence with convenient base case
\[
T(n) = 2T(n/2) + \Theta(n)
\]
\[
T(1) = O(1)
\]

▶ How do we solve the recurrence for simple expression for \( T(n) \)?
  First, use definition of \( \Theta \) (Big-O suffices for upper bound)
\[
T(n) \leq 2T(n/2) + cn
\]
\[
T(1) \leq c
\]

▶ Same constant? Doesn't matter, choose largest one.
▶ What next?
Solution Idea 1: Unroll the Recurrence

\[ T(n) \leq 2T(n/2) + cn \]
\[ \leq 2\left[2T(n/4) + c(n/2)\right] + cn \]
\[ \leq 2\left[2\left(2T(n/8) + c(n/4)\right) + c(n/2)\right] + cn \]

▶ This can get messy, but works if we group terms

Clicker Question 2

Which of the two terms in the sum contributes more asymptotically?

A) \(2^k T(n/2^k)\)
B) \(k \cdot n\)
C) Both equally

First term will sum to \(n\) base cases, \(O(n)\) work; second term will be \(O(n \log n)\)

Idea 2: Draw recursion tree, track work done at each level (same unrolling approach, different organization)

Solution Idea 3: Guess and Verify

Guess solution

Prove by (strong) induction

Guess \(T(n) = c \cdot n \log n\)

Base case: \(n = 2\), \(T(2) \leq c \cdot 2 \log 2 = 2c\). True?
If not, work with \(c' = \max(T(2)/2, c)\)

Unrolling the Recurrence (cont.)

\[ T(n) \leq 2T(n/2) + cn \]
\[ \leq 2\left[2T(n/4) + c(n/2)\right] + cn \]
\[ = 2^2 \cdot T(n/4) + 2 \cdot c(n/2) + cn \]
\[ = 2^n \cdot T(n/2^{\log_2 n}) + 2 \cdot cn \]
\[ \leq 2^n \cdot \left(2T(n/8) + c(n/4)\right) + 2 \cdot cn \]
\[ = 2^n \cdot T(n/8) + 3 \cdot cn \]

Do you see a pattern? \(2^k T(n/2^k) + k \cdot cn\)

When does this stop? Base case after \(k = \log n\) unrollings.

Guess and Verify: Induction Step

Strong induction:

Assume \(T(m) \leq c \cdot m \log m\) for all \(m < n\), prove for \(n\)

\[ T(n) \leq 2 \cdot T(n/2) + cn \]
\[ \leq 2 \cdot c \cdot n \log (n/2) + cn \]
\[ = cn(\log n - 1) + cn \]
\[ = cn \log n \]

The induction proof is complete, \(T(n) \leq cn \log n\)
Another Problem: Maximum Subsequence Sum (MSS)

- **Input**: array $A$ of $n$ numbers, e.g. $A = 4, -3, 5, -2, -1, 2, 6, -2$
- **Find**: value of the largest subsequence sum $A[i] + A[i+1] + \ldots + A[j]$
- (empty subsequence allowed and has sum zero)
- **MSS in example?** 11 (first 7 elements)

What is a simple algorithm for MSS?

A Problem from HW1

MSS($A$)

Initialize all entries of $n \times n$ array $B$ to zero

for $i = 1$ to $n$ do
  sum = 0
  for $j = i$ to $n$ do
    $B[i, j] = sum$
  end for
end for

Return maximum entry of $B[i, j]$

Running time? $O(n^2)$. Can we do better?

Clicker Question 3

Which of the following is true for a maximum-sum subsequence

A) It has more positive than negative numbers
B) No left subsequence (prefix) of it has negative sum
C) Any maximal sequence of negative numbers is bordered by a sequence of positive numbers with sum larger in absolute value

Divide-and-conquer for MSS

- **Recursive solution for MSS**
- **Idea**: 
  - Find MSS $L$ in left half of array
  - Find MSS $R$ in right half of array
  - Find MSS $M$ for sequence that crosses the midpoint

  $A = 4, -3, 5, -2, -1, 2, 6, -2$
  $L=6$
  $R=8$

- Return $\max(L, R, M)$
- In picture, $M$ encompasses $L$ and $R$. Coincidence?
  Yes, $M$ need not look like in the picture (but it won’t stop in the middle of either $L$ or $R$, why?)

MSS($A$, left, right)

if left == right then
  return max($A[\text{left}]$, 0)
end if

mid = \left\lfloor \frac{\text{left} + \text{right}}{2} \right\rfloor

L = MSS($A$, left, mid)
R = MSS($A$, mid+1, right)

Set $sum = 0$ and $L' = 0$
for $i = \text{mid}$ down to 1 do
  $sum += A[i]$
  $L' = \max(L', sum)$
end for

Set $sum = 0$ and $R' = 0$
for $i = \text{mid}+1$ to right do
  $sum += A[i]$
  $R' = \max(R', sum)$
end for

$M = L' + R'$

return max($L$, $R$, $M$)

running time?

- Let $T(n)$ be running time of MSS on array of size $n$
- Two recursive calls on arrays of size $n/2$: $2T(n/2)$
- Work outside of recursive calls: $O(n)$
- Running time

  $T(n) = 2T(n/2) + O(n)$

  We’ve seen that

  $T(n) = O(n \log n)$
A More General Recurrence

\[ T(n) \leq q \cdot T(n/2) + cn \]

- What does the algorithm look like?
  - \( q \) recursive calls to itself on problems of half the size
  - \( O(n) \) work outside of the recursive calls

- Exercises: \( q = 1, q > 2 \)

- Useful fact (geometric sum): if \( r \neq 1 \) then
  \[
  1 + r + r^2 + \ldots + r^d = \frac{1 - r^{d+1}}{1 - r}
  \]

Case \( q > 2 \) (continued)

\[ T(n) \leq q^2 T(n/4) + qcn/4 + c n/2 + cn \]

\[ \leq q^3 T(n/8) + q^2 cn/4 + qcn/2 + cn \]

\[ \ldots \]

\[ \leq \sum_{i=0}^{\log n-1} c n/2^i \]

\[ \leq 2cn \]

Conclusion: \( T(n) = O(n) \)

Case \( q = 1 \):

Unrolling

\[ T(n) \leq T(n/2) + cn, \quad T(1) \leq c \]

\[ T(n) \leq T(n/2) + cn \]

\[ \leq T(n/4) + cn/2 + cn \]

\[ \leq T(n/8) + cn/4 + cn/2 + cn \]

\[ \ldots \]

\[ \leq \sum_{i=0}^{\log n-1} c n/2^i \]

\[ = c n \sum_{i=0}^{\log n-1} (q/2)^i \]

Conclusion: \( T(n) = O(n) \)

Case \( q > 2 \)

\[ T(n) \leq q^2 T(n/4) + qcn/4 + qcn/2 + cn \]

\[ \leq q^3 T(n/8) + q^2 cn/4 + qcn/2 + cn \]

\[ \ldots \]

\[ \leq \sum_{i=0}^{\log n-1} c n/2^i \]

\[ = c n \sum_{i=0}^{\log n-1} (q/2)^i \]

Summary

Useful general recurrence and its solutions:

\[ T(n) \leq q \cdot T(n/2) + cn \]

1. \( q = 1 \): \( T(n) = O(n) \) work outside recursion dominates
2. \( q = 2 \): \( T(n) = O(n \log n) \) equal contributions
3. \( q > 2 \): \( T(n) = O(n^{\log_2 q}) \) base-case subproblems dominate

Algorithms with these running times?

Mergesort. MSS

Counting inversions, Closest points, Integer multiplication