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

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

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &\leq 2\Big[2T(n/4) + c(n/2)\Big] + cn \\ &\leq 2\Big[2(2T(n/8) + c(n/4)) + c(n/2)\Big] + cn \end{aligned}$$

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

Unrolling the Recurrence (cont.)

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &\leq 2\Big[2T(n/4) + c(n/2)\Big] + cn \\ &= 2^2 \cdot T(n/4) + 2 \cdot c(n/2) + cn \\ &= 2^2 \cdot T(n/4) + 2 \cdot cn \\ &\leq 2^2 \cdot (2T(n/8) + c(n/4)) + 2 \cdot cn \\ &= 2^3 \cdot T(n/8) + 3 \cdot cn \end{aligned}$$

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.

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 $O(n \log n)$

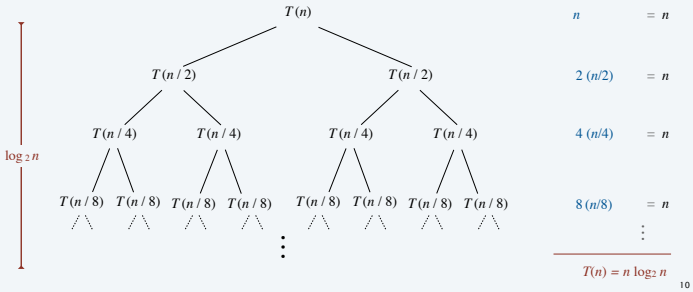
Idea 2: Draw *recursion tree*, track work done at each level (same unrolling approach, different organization)

Divide-and-conquer recurrence: recursion tree

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

assuming n is a power of 2



slide credit: Kevin Wayne / Pearson

Solution Idea 3: Guess and Verify

Guess solution

Prove by (strong) induction

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

Base case: $n = 2$, $T(2) \leq c2 \log 2 = 2c$. True?
If not, work with $c' = \max(T(2)/2, c)$

Guess and Verify: Induction Step

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

$$\begin{aligned} T(n) &\leq 2 \cdot T(n/2) + cn \\ &\leq 2 \cdot c(n/2) \log(n/2) + cn \\ &= cn(\log n - 1) + cn \\ &= cn \log n \end{aligned}$$

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)

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

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**
      sum += $A[j]$ ▷ HW1: alternating sum
      $B[i, j]$ = sum
    end for
  end for
  Return maximum entry of $B[i, j]$

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

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

$$A = \overbrace{\underbrace{4, -3, 5}_{L=6}, -2, -1, \underbrace{2, 6}_{R=8}}^{M=11}, -2$$

- Return $\max(L, R, M)$
- In picure, 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** ▷ Base case
    return $\max(A$[left], 0)
  end if

  mid = $\lfloor \frac{\text{left+right}}{2} \rfloor$ ▷ Recurse on left and right halves
  L = MSS(A, left, mid)
  R = MSS(A, mid+1, right)

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

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

  return $\max(L, R, M)$ ▷ Return max

MSS(A, left, right)
  if left == right **then**
    return $\max(A$[left], 0)
  end if

  mid = $\lfloor \frac{\text{left+right}}{2} \rfloor$
  L = MSS(A, left, mid)
  R = MSS(A, mid+1, right)

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

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

$$\boxed{T(n) \le q \cdot T(n/2) + cn}$$

- 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

$$1 + r + r^2 + \ldots + r^d = \frac{1 - r^{d+1}}{1 - r} = \frac{r^{d+1} - 1}{r - 1}$$

Case $q = 1$: Unrolling

$T(n) \le T(n/2) + cn, \qquad T(1) \le c$

$$\begin{aligned} T(n) &\le T(n/2) + cn \\ &\le T(n/4) + cn/2 + cn \\ &\le T(n/8) + cn/4 + cn/2 + cn \\ &\ldots \\ &\le \sum_{i=0}^{\log n - 1} cn/2^i \\ &\le 2cn \end{aligned}$$

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

Case $q = 1$: Guess + Induction

$T(n) \le T(n/2) + cn, \qquad T(1) \le c$

Guess $T(n) \le kn$

Base case: need $k \ge c$, then $T(1) \le c \le k \cdot 1$

Induction step:

$$T(n) \le T(n/2) + cn \le kn/2 + cn$$

Need $T(n) \le kn$, thus $kn/2 + cn \le kn$, thus $k \ge 2c$

Choosing $k = 2c$ suffices and thus by induction $T(n) \le 2cn = O(n)$

Case $q > 2$

$T(n) \le qT(n/2) + cn, \qquad T(1) \le c$

$$\begin{aligned} T(n) &\le qT(n/2) + cn \\ &\le q^2T(n/4) + qcn/2 + cn \\ &\le q^3T(n/8) + q^2cn/4 + qcn/2 + cn \\ &\ldots \\ &\le \sum_{i=0}^{\log n - 1} cn(q/2)^i \\ &= cn \sum_{i=0}^{\log n - 1} (q/2)^i \end{aligned}$$

Case $q > 2$ (continued)

$$\begin{aligned} T(n) &\le cn \sum_{i=0}^{\log n - 1} (q/2)^i \\ &= cn\Big(\frac{(q/2)^{\log n} - 1}{q/2 - 1}\Big) \\ &\le cn\Big(\frac{(q/2)^{\log n}}{q/2 - 1}\Big) \\ &= \Big(\frac{c}{q/2 - 1}\Big) n \cdot (q/2)^{\log n} \\ &= \Big(\frac{c}{q/2 - 1}\Big) n \cdot n^{\log(q/2)} \\ &= \Big(\frac{c}{q/2 - 1}\Big) n \cdot n^{\log q - 1} \\ &= \Big(\frac{c}{q/2 - 1}\Big) n^{\log q} = O(n^{\log q}) \end{aligned}$$

Summary

Useful general recurrence and its solutions:

$$\boxed{T(n) \le q \cdot T(n/2) + cn}$$

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