Lecture Summary

Sorting Methods II

Andrew C. Lee

EECS, Syracuse

Divide and Conquer Based Methods

Two examples:

1. Merge Sort
2. Quick Sort
3. Reading: CLRS Chapter 2 and Chapter 7

Five Basic Design Questions

- Ideas and Problem Formulation
- Pseudocode
- Correctness
- Complexity
- Limitations

Merge Sort: The ideas

[The Divide-and-Conquer Paradigm (CLRS, page 30)]

1. (Divide) Divide the n-element sequence to be sorted into two subsequences of $\frac{n}{2}$ elements each.
2. (Conquer) Sort the two subsequences recursively using merge sort.
3. (Combine) Merge the two sorted subsequences to produce the sorted answer.

Background: Strong Induction

Fact (Principle of Mathematical Induction: Strong Form)

Let $P(n)$ a statement for each non-negative integers n. Suppose that

1. *(Base Case) $P(0)$ is true.*
2. *(Induction Step)* $P(0) \wedge \ldots \wedge P(k) \Rightarrow P(k+1)$ *for any $k \geq 0$*
3. *(Conclusion) The statement $P(n)$ is true for any non-negative integer n.*

Question What's the relationship between the divide-and-conquer paradigm and strong induction ?

Merge Sort: The ideas

[The Divide-and-Conquer Paradigm (CLRS, page 30)]

1. (Divide) Divide the n-element sequence to be sorted into two subsequences of $\frac{n}{2}$ elements each.
2. (Conquer) Sort the two subsequences recursively using merge sort.
3. (Combine) Merge the two sorted subsequences to produce the sorted answer.

Question What's the relationship between the divide-and-conquer paradigm and strong induction ?

The Combine Step: Algorithm Merge

```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```

Figure: Algorithm Merge

The Combine Step: Algorithm Merge

Discussions

Examine both the pseudocode for algorithm

$$\text{Merge } (A,p,q\ r)$$

and the illustrations (CLRS, page 32-33) given

In your own words, describe how Algorithm Merge works

Algorithm: Merge Sort

MERGE-SORT.$(A; p; r)$

1. if $(p < r)$
2. $\quad q = \lfloor \frac{p+r}{2} \rfloor$
3. $\quad$ MERGE-SORT $(A;p;q)$
4. $\quad$ MERGE-SORT $(A; q+1; r)$
5. $\quad$ MERGE $(A;p; q; r)$

Question

Is the Algorithm Correct ? Informally, What do we have to check ?
How to analyze its running time, the function $T(n)$?

Analyzing Divide-and-Conquer Algorithms

Divide the problem into a subproblems
(each of which is $\frac{1}{b}$ the size of the original.)
Let
$D(n)$: time used to divide the problem
$C(n)$: time to combine the solutions

$$T(n) = \begin{cases} \Theta(1) & n \leq c \text{ (a constant)} \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Question: From the expression, can we determine the order of growth of $T(n)$?

Analyzing Merge Sort

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \text{ (a constant)} \\ 2T(\frac{n}{2}) + \Theta(n) & \text{otherwise} \end{cases}$$

There are general techniques for solving the above *recurrence*. We will also illustrate how to determine $T(n)$ (informally) using a tree structure called *recurrence tree*.

Recurrence Trees

Consider

$$T(n) = \begin{cases} 1 & n \leq 1 \text{ (a constant)} \\ 2T(\frac{n}{2}) + n & \text{otherwise} \end{cases}$$

Unwrap the recurrence calculations in the form of a Tree and re-group the terms. What can we obtain in this simple case ?

Remarks

We need to work with

1. Rigorous definitions of O, Θ and Ω
2. Have solid understanding of the growth rate of functions such as (a and k are some constants)

$$n^k \text{ (polynomials)}, a^n \text{ (exponential functions)}, nlog\ n, etc.$$

3. To analyze divide and conquer algorithms, we need to learn general methods to determine the growth rate $T(n)$, where

$$T(n) = \begin{cases} \Theta(1) & n \leq c \text{ (a constant)} \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Quicksort

The ideas ..

$$A[1] \ldots A[n]$$

- Divide: Partition Array A into two:

  $$A[1], \ldots, A[q] \text{ and } A[q+1] \ldots A[n]$$

  Any members from the first part is less than or equal to any members from the second part
- Conquer: Apply recursion
- Combine: Do we need to do any work ?

How to partition

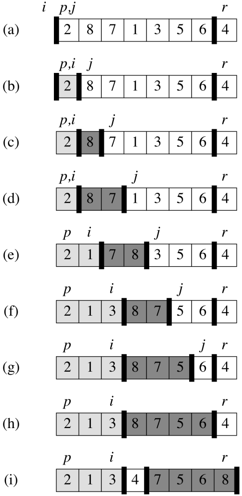

Figure: How Partition Works: note the movement of i and j

How to partition

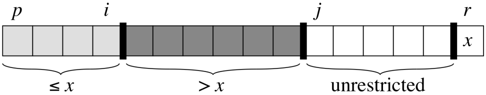

Figure: How Partition Works: why there are progresses

How to partition

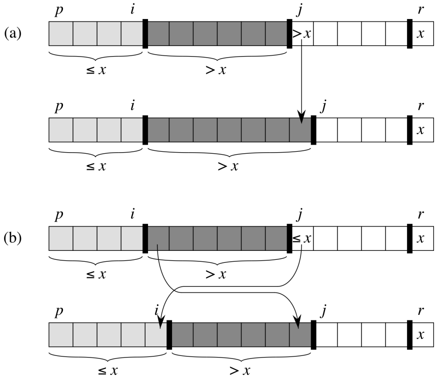

Figure: Why Partition is correct

Partition: The algorithm

PARTITION(A, p, r)

$x = A[r]$
$i = p - 1$
for $j = p$ **to** $r - 1$
    if $A[j] \leq x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$

Figure: The Partition Algorithm

Quicksort: The algorithm

$\textsc{Quicksort}(A, p, r)$

if $p < r$

$q = \textsc{Partition}(A, p, r)$

$\textsc{Quicksort}(A, p, q - 1)$

$\textsc{Quicksort}(A, q + 1, r)$

Figure: The Quicksort Algorithm

Remarks on Quicksort

1. Worst Case can be bad: It is $\Theta(n^2)$, why ?
2. Runs well in practice
3. Needs to *spice it up* with "coin flips"