Divide and Conquer

Hao Jiang

Boston College
Divide-and-Conquer Algorithms

- Break problems into subproblems
- Recursively solve each subproblem
- Combine the result
Binary Search

function find_up_point(A[0..N-1])
    i = 0; j = N-1;
    // The target is in the [i,j], A[i] < A[j]
    while (i != j-1)
        m = (i+j)/2;
        if (A[m] < A[m+1]) return m;
        else
            if (A[m] > A[i]) j = m;
            else i = m+1;

    return i;
Find the range

Given a sorted integer array \( A[0..N-1] \), and a query number \( q \), find \([i,j]\) so that \( A[i] \leq q \leq A[i+1]\)
Given a sorted integer array $A[0..N-1]$, and a query number $q$, find the maximum $A[i]$ so that $A[i] \leq q$.

```plaintext
function floor(A, s, e, q)
// return the index of the
// maximal element in A[s..e]
// that is not greater than q

if (s > e) return -1;

m = (s + e)/2;
if (A[m] == q) return m;
if (q < A[m]) return floor(A, s, m-1, q);
t = floor(A, m+1, e, q);
if (t != -1) return t;
else return m;
```
Merge Sort

Problem: Given an array of numbers, sort them into ascending order.

Idea: Instead of sorting the whole sequence directly, we partition the input into two halves and recursively sort each and combine

sort(A[1..n])

sort(A[1..floor(n/2)])  sort(A[ceil(n/2)..n])
Merge Sort

sort(A[1..n])

sort(A[1..floor(n/2)])

sort(A[ceil(n/2)..n])

combine
**Sequo-code**

```python
function mergesort(a[1...n])
Input: An array of numbers a[1...n]
Output: A sorted version of this array

if n > 1:
    return merge(mergesort(a[1...[n/2]]), mergesort(a[[n/2]+1...n]))
else:
    return a
```
How to merge?

function merge(a[1..n], b[1..m])

    if n = 0 return b
    if m = 0  return a
    if a[1] < b[1]
        return [a[1], merge(a[2..n], b[1..m])]
    else
        return [b[1], merge(a[1..n], b[2..m])]
function merge( A[0..N-1], B[0..M-1] )
% Input: A and B which are sorted in the
increasing order
% Output: Sorted C that contains elements
from A and B

creat C that has N+M elements;
i = 0; j = 0; k = 0;
// {Invariant: A[0..i) and B[0..j) have
been merged into C[0..k) }
while ( i!= N || j!=M ) // !(i == N && j == M)
    if (i == N) C[k] = B[j]; ++j;
    else if (j == M) C[k] = A[i]; ++i;
    else if (A[i] < B[j]) C[k] = A[i]; ++i;
    else C[k] = B[j]; ++j;
++k;

return C;
function iterative-mergesort(a[1...n])
Input: elements $a_1, a_2, ..., a_n$ to be sorted

$Q = [\ ]$ (empty queue)
for $i = 1$ to $n$:
    inject($Q, [a_i]$)
while $|Q| > 1$:
    inject($Q, merge(eject(Q), eject(Q))$)
return eject($Q$)
Complexity

# of nodes:

1

2

4

2^\log(n) = n

Problem size

n

n/2

n/4

2^\log(n) = n

Level k complexity = O(n/(2^k)) * 2^k = O(n)

Total complexity = O(n) * \log n = O(n \log n)
A divide and conquer method calls itself “a” times and each subproblem's is $1/b$ of the original size, and the complexity of merging/splitting result is $O(n^d)$, then

$$T(n) = aT(\text{ceil}(n/b)) + O(n^d)$$

- Expanding factor
- Ceiling function
- Combining complexity
- Subproblem size shrinking factor
The complexity

Size $n$

Size $n/b$

Size $n/b^2$

Depth $\log_b n$

Width $a^{\log_b n} = n^{\log_b a}$
Master Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$, $a > 0$, $b > 1$, $d > 0$ then

$$T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}.$$
Order Statistics

Problem: Given an array of numbers, find the k th smallest number.

Example: 1 5 2 6 9

The 1st smallest is 1
The 3rd (the median) is 5
Median Filtering in Denoising

Original Image

Add 10% pepper noise
Median Filtering for Denoising

Median filter with 3x3 square structure element
Median Filtering for Denoising

Median filter with 5x5 square structure element
Compared with Gaussian Filtering

Kernel size 5x5 and sigma 3

Kernel size 11x11 and sigma 5
Partition

Pick up a number \( v \) from the array, then reorganize the array into:

\[
\begin{array}{ccc}
< v & ==v & > v \\
\end{array}
\]

This can be done in \( O(n) \) time.
Case 1: $k > p$ && $k < q$, the result is $v$

Case 2: $k \leq p$, we work on the subsequence $a[1..p]$ and find the $k$ th smallest number.

Case 3: $k \geq q$, we work on the subsequence $a[q..n]$ and find the $(k-q+1)$ th smallest number.
function select(a[1..n], k)

Choose an element v from a[1..n]
Organize a into three parts,

1   p   q   n
   < v   ==v   > v

if k > p and k < q, return v
if k <= q, return select(a[1..p], k)
if k >= q, return select(a[q..n], k-q+1)
Quick Partition

function partition (A[0..N-1], v)
// Input: An array and a pivot element v
// Output: A is partitioned into 3 regions.
//         Left region < v. Middle == v and
//         Right region > v.

i = 0; j = 0; k = N-1;
// {Invariant: A[0..i) < v; A[i..j) == v;
//     A(k.. N-1] > v, i <= j }
while ( j != k+1 ) // !(j == k+1)
    if (A[j] == v) j++;
    else if (A[j] < v) swap(A, i, j); i++; j++;
    else swap(A, j, k); k--;
The (worst case) Complexity

If we are very lucky, then each time the sequence is at least halved

\[ T(n) = T(n/2) + O(n) \implies T(n) = n \]

If we are very unlucky, each time the length of the sequence reduces by only 1

\[ T(n) = T(n-1) + O(n) \implies T(n) = 1 + 2 + 3 + \ldots + (n-1) = O(n^2) \]
The Average Complexity

We have \( \frac{1}{2} \) chance to pick up a \( v \) that is located in the range from 25% to 75%.

\[
T(n) = T\left(\frac{3}{4}n\right) + \text{time to reduce the array from } n \text{ to } \frac{3}{4}n
\]

The average complexity \( T(n) = O(n) \)
Quicksort

Quicksort picks up a pivot element $v$ and partition a sequence into two parts:

\[
\begin{array}{c|c}
& v \\
\hline
\leq v & > v
\end{array}
\]

Then recursive sort the left and eight part
Quicksort

```
quicksort(A[1..n])
    if (n > 1)
        choose a pivot element A[p]
        k <- partition(A, p)
        quicksort(A[1..k-1]);
        quicksort(A[k+1..n]);
```
Quicksort Complexity

With good luck:
\[ T(n) = 2T(n/2) + O(n) \]  \implies  \[ T(n) = O(n \log n) \]

With bad luck:
\[ T(n) = T(n-1) + O(n) \]  \implies  \[ T(n) = O(n^2) \]

On the average:
\[ T(n) = O(n \log n) \]
Finding # of Reversed Pairs

Question: Given an array of distinct numbers, find how many number pairs are in the reversed order.

Example:

6 1 3 2 7

4 reversed pairs: (6,1), (6,3), (6,2), (3,2)
Divide and Conquer

\[ \text{count}(A[1..n]) \]

\[ \text{count}(A[1..\text{floor}(n/2)]) \]

\[ \text{count}(A[\text{ceil}(n/2)..n]) \]

combine
Divide and Conquer

\[ \text{count}(A[1..n]) \]

\[ \text{count}(A[1..\text{floor}(n/2)]) \quad \text{count}(A[\text{ceil}(n/2)..n]) \]

combine

But how to combine ???

We need a merging method this is faster than \( O(n) \)
Divide and Conquer

sort_count(A[1..n])

sort_count(A[1..floor(n/2)])

sort_count(A[ceil(n/2)..n])

combine

Now there is a O(n) merging method.
merge_count

function [C, k] = merge_count(A[0..N-1], B[0..M-1])
// Input: A and B are sorted
// Output: C is the merged A and B. k is the number of pairs (p,q) for which p > q and P is from A and q is from B

i = 0; j = 0; n = 0; k = 0;
// {invariant: A[0..i) and B[0..j) are merged into C[0..n) and k is # of the inversed pairs }
while (i != N || j != M)
    if (i == N) C[n] = B[j]; j++;
    else
        if (j == M) C[n] = A[i]; i++;
        else if (A[i] <= B[j]) C[n] = A[i]; i++;
        else C[n] = B[j]; j++;
    n++;
return [C, k];
The algorithm

function sort_count(A[1..n])
    if n = 0
        return (A, 0)
    (L, n) = sort_count(A[1..floor(n/2)])
    (R, m) = sort_count(A[ceil(n/2), n])
    (B, k) = merge_count(L, R)
    return (B, n+m+k)
Matrix Multiplication

\[ Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}. \]

\[ T(n) = O(n^3) \]
Divide and Conquer

\[
\begin{array}{cc}
A & B \\
C & D \\
\end{array}
\times
\begin{array}{cc}
E & F \\
G & H \\
\end{array}
= 
\begin{array}{cc}
AE+BG & AF+BH \\
CE+DG & CF+DH \\
\end{array}
\]
Divide and Conquer

\[
\begin{array}{|c|c|}
A & B \\
\hline
C & D \\
\end{array}
\times
\begin{array}{|c|c|}
E & F \\
\hline
G & H \\
\end{array}
= 
\begin{array}{|c|c|}
AE+BG & AF+BH \\
\hline
CE+DG & CF+DH \\
\end{array}
\]

\[T(n) = 8T(n/2) + O(n^2)\]

\[\Rightarrow T(n) = T(n^3)\]
Can we do better?

\[ XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 \\ P_3 + P_4 \\ P_3 + P_4 \end{bmatrix} \begin{bmatrix} P_1 + P_2 \\ P_1 + P_5 - P_3 - P_7 \end{bmatrix} \]

\[
P_1 = A(F - H) \quad P_5 = (A + D)(E + H) \\
P_2 = (A + B)H \quad P_6 = (B - D)(G + H) \\
P_3 = (C + D)E \quad P_7 = (A - C)(E + F) \\
P_4 = D(G - E) \]

We only need 7 matrix multiplications in here

\[ T(n) = 7T(n/2) + O(n^2) \implies T(n) = n^{\log_2(7)} \]
Finding Closest Point Pair

Which two points are the closest?
1D case
Divide and Conquer

Find d on the left
Find d on the right
Merge
Smart merge

No need to consider

No need to consider
Smart merge

Red dot to light blue dot distance must $> d$
Smart merge

There are no more than 4 blue dots!
Smart merge

There are no more Than 8 blue dots!
function get_smallest_d(Point p[1..n])
  sort points according to x and y coordinates
  and put into two arrays Px[1..n] and Ay[1..n]
  
dleft = get_smallest_d(Px[1..floor(n/2)])
dright = get_smallest_d(Px[ceil(n/2), n])
d = min(dleft, dright)
  sequentially select point in Ay that falls
  in the band and form Py
dy = get_smallest_8(Py)
return min(d, dy)