Divide and Conquer

CSC 349

Christopher Siu

Divide and Conquer

Definition

A **divide and conquer** algorithm is one that:

1. Divides a problem into smaller instances of the same problem.
2. Solves these subproblems, obtaining subsolutions.
3. Combines these subsolutions into one to the original problem.

- The divide and conquer approach suggests, but does not require, a recursive algorithm.
- Some divide and conquer algorithms resemble binary search, where the work is primarily in dividing up the problem.
- Other divide and conquer algorithms resemble merge sort, where the work is primarily in combining the solutions.

Binary Search

Binary Search

$\textsc{BinarySearch}(x, A = (a_0, a_1, \ldots, a_{n-1}))$

Input: An integer x and a finite, non-empty sequence A of n integers sorted in ascending order

Output: Whether or not x is an element of A

let mid be $\lfloor {}^{n}/_{2} \rfloor$
if $a_{\mathsf{mid}} = x$ **then**
  return T
else if $n = 1$ **then**
  return F
else if $a_{\mathsf{mid}} > x$ **then**
  return $\textsc{BinarySearch}(x, (a_0, a_1, \ldots, a_{\mathsf{mid}-1}))$
else
  return $\textsc{BinarySearch}(x, (a_{\mathsf{mid}}, a_{\mathsf{mid}+1}, \ldots, a_{n-1}))$

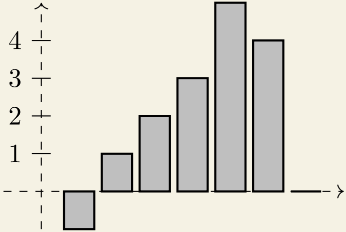
Finding Peaks

Consider the following problem:

- Given a unimodal sequence of distinct integers, find the **peak**, the maximum element.

Example

Given $(-1, 1, 2, 3, 5, 4, 0)$:



...the peak element is 5.

FindingPeaks

$\textsc{FindPeak}(A = (a_0, a_1, \ldots, a_{n-1}))$

Input: A finite, non-empty, unimodal seq. A of n distinct integers
Output: The peak element of A

let mid be $\lfloor n/2 \rfloor$
if $n = 1$ **then**
    return a_0
else if $n = 2$ **then**
    return $\max\{a_0, a_1\}$
else if $a_{\mathsf{mid}} > a_{\mathsf{mid}-1}$ **and** $a_{\mathsf{mid}} > a_{\mathsf{mid}+1}$ **then**
    return a_{mid}
else if $a_{\mathsf{mid}-1} < a_{\mathsf{mid}} < a_{\mathsf{mid}+1}$ **then**
    return $\textsc{FindPeak}((a_{\mathsf{mid}+1}, a_{\mathsf{mid}+2}, \ldots, a_{n-1}))$
else
    return $\textsc{FindPeak}((a_0, a_1, \ldots, a_{\mathsf{mid}-1}))$

Finding Peaks

Theorem

FindPeak is correct.

Lemma

Let A be a finite, non-empty, unimodal seq. of distinct integers. If $\textsc{FindPeak}(A)$ returns x, then $x \in A$ and $\forall a \in A\,(x \geq a)$.

Example

The complexity of FindPeak is given by:

$$\begin{aligned} T(n) &= a \cdot T(m) + O(g(n)) \\ &= 1 \cdot T({}^{n}/_{2}) + O(1) \\ &\approx O(\log n) \end{aligned}$$

Merge Sort

Merge Sort

$\textsc{MergeSort}(A = (a_0, a_1, \ldots, a_{n-1}))$

Input: A finite, non-empty sequence A of n integers
Output: A permutation of A sorted in ascending order

let mid be $\lfloor {}^{n}/_{2} \rfloor$
if $n = 1$ **then**
  return A
else
  return $\textsc{Merge}(\textsc{MergeSort}((a_0, a_1, \ldots, a_{\mathsf{mid}-1}))\,,$
  $\textsc{MergeSort}((a_{\mathsf{mid}}, a_{\mathsf{mid}+1}, \ldots, a_{n-1})))$

Merge Sort

$\textsc{Merge}(A = (a_0, a_1, \ldots, a_{n-1}), B = (b_0, b_1, \ldots, b_{m-1}))$

Input: Two finite, non-empty sequences A and B of n and m integers sorted in ascending order

Output: A permutation of $A \mathbin{+\!\!+} B$ sorted in ascending order

if $n = 0$ **then**
    return B
else if $m = 0$ **then**
    return A
else if $a_0 \leq b_0$ **then**
    return $(a_0) \mathbin{+\!\!+} \textsc{Merge}((a_1, a_2, \ldots, a_{n-1}), B)$
else
    return $(b_0) \mathbin{+\!\!+} \textsc{Merge}(A, (b_1, b_2, \ldots, b_{m-1}))$

Merge Sort

Theorem

$\textsc{MergeSort}$ is correct.

Lemma

Let A and B be finite, non-empty sequences of integers sorted in ascending order. If $\textsc{Merge}(A, B)$ returns C, then C is a permutation of $A + B$ sorted in ascending order.

Lemma

Let A be a finite, non-empty sequence of integers. If $\textsc{MergeSort}(A)$ returns A', then A' is a permutation of A sorted in ascending order.

Merge Sort

Example

The complexity of MERGE is given by:

$$\begin{aligned} T(n) &= a \cdot T(m) + O(g(n)) \\ &= 1 \cdot T(n-1) + O(1) \\ &= O(n) \end{aligned}$$

Thus, the complexity of MERGESORT is given by:

$$\begin{aligned} T(n) &= a \cdot T(m) + O(g(n)) \\ &= 2 \cdot T({}^{n}/_{2}) + O(n) \\ &= O(n \log n) \end{aligned}$$

Matrix Multiplication

Consider the following problem:

- Given two $n \times n$ matrices, find their product.

Definition

The **product** of two $n \times n$ matrices X and Y, denoted XY, is an $n \times n$ matrix Z wherein:

$$z_{ij} = \sum_{k=0}^{n-1} x_{ik} y_{kj}$$

Example

$$\begin{bmatrix} 0 & 2 & 1 & 4 \\ 1 & 0 & 0 & 2 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 4 & 2 \\ 2 & 3 & 0 & 3 \\ 2 & 1 & 1 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 6 & 7 & 5 & 9 \\ 1 & 0 & 6 & 2 \\ 5 & 6 & 5 & 8 \\ 6 & 3 & 7 & 9 \end{bmatrix}$$

Matrix Multiplication

$\textsc{MMult}(X, Y)$

Input: Two $n \times n$ matrices X and Y, where n is a power of 2
Output: The product of X and Y

```
if n = 1 then
    return [x_0 · y_0]
else
    let A, B, C, D be n/2 × n/2 matrices, such that X = [A B; C D]
    let E, F, G, H be n/2 × n/2 matrices, such that Y = [E F; G H]
    return [MMult(A,E)+MMult(B,G)  MMult(A,F)+MMult(B,H); MMult(C,E)+MMult(D,G)  MMult(C,F)+MMult(D,H)]
```

1: **if** $n = 1$ **then**
2: **return** $\left[x_0 \cdot y_0\right]$
3: **else**
4: **let** A, B, C, D be ${}^{n}/_{2} \times {}^{n}/_{2}$ matrices, such that $X = \left[\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right]$
5: **let** E, F, G, H be ${}^{n}/_{2} \times {}^{n}/_{2}$ matrices, such that $Y = \left[\begin{smallmatrix} E & F \\ G & H \end{smallmatrix}\right]$
6: **return** $\begin{bmatrix} \textsc{MMult}(A,E)+\textsc{MMult}(B,G) & \textsc{MMult}(A,F)+\textsc{MMult}(B,H) \\ \textsc{MMult}(C,E)+\textsc{MMult}(D,G) & \textsc{MMult}(C,F)+\textsc{MMult}(D,H) \end{bmatrix}$

Matrix Multiplication

Example

The complexity of $\textsc{MMult}$ is given by:

$$\begin{aligned} T(n) &= a \cdot T(m) + O(g(n)) \\ &= 8 \cdot T(^n/_2) + O\left(n^2\right) \\ &= O\left(n^3\right) \end{aligned}$$

- The basic divide and conquer algorithm offers no improvement over a naïve algorithm.
- The multiplications that must be performed have not been *reduced*; they have merely been *rearranged*.

Strassen's Algorithm

Theorem

Let X and Y be $n \times n$ matrices such that $X = \left[\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right]$ and $Y = \left[\begin{smallmatrix} E & F \\ G & H \end{smallmatrix}\right]$, where A, B, ..., H are each ${}^{n}/_{2} \times {}^{n}/_{2}$ matrices. If:

$$
\begin{aligned}
P_1 &= A(F - H) & P_5 &= (A + D)(E + H) \\
P_2 &= (A + B)H & P_6 &= (B - D)(G + H) \\
P_3 &= (C + D)E & P_7 &= (A - C)(E + F) \\
P_4 &= D(G - E) & &
\end{aligned}
$$

Then:

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

Strassen's Algorithm

Example

The complexity of Strassen's algorithm is given by:

$$\begin{aligned} T(n) &= a \cdot T(m) + O(g(n)) \\ &= 7 \cdot T(^n/_2) + O\left(n^2\right) \\ &\approx O\left(n^{2.807}\right) \end{aligned}$$

- As of 2020, the best known matrix multiplication algorithm has complexity $O\left(n^{2.3728596}\right)$.
- Note that any matrix multiplication algorithm must necessarily have complexity $\Omega\left(n^2\right)$.