Divide and Conquer

CS 141, Spring 2019
(Chapter 2)

Divide and Conquer

• *Divide*: If the input size is too large to deal with in a straightforward manner, divide the data into two or more disjoint subsets

• *Recur*: Use divide and conquer to solve the sub-problems associated with the data subsets

• *Conquer*: Take the solutions to the sub-problems and “merge” these solutions into a solution for the original problem
Divide and Conquer

Outline

• Already covered/known
  – Sorting: Mergesort
  – Searching: Binary Search
• Integer Multiplication (Karatsuba)
• Matrix Multiplication (Strassen)
• Closest Pair
• Linear-time selection
Integer multiplication (Karatsuba)

• Given positive integers $y, z$, compute $x = y \times z$
• A naïve multiplication algorithm is below

```python
def naive_mul(y, z):
    x = 0
    while z > 0:
        if z % 2 == 1:
            x += y
        y *= 2
        z /= 2
    return x
```

Remark: these two operations can be implemented as $O(1)$ shifts
Integer multiplication

Addition takes $O(n)$ bit operations, where $n$ is the number of bits in $y$ and $z$. The naive multiplication algorithm takes $O(n)$ $n$-bit additions. Therefore, the naive multiplication algorithm takes $O(n^2)$ bit operations.

Can we multiply using fewer bit operations?

---

Integer multiplication

Suppose $n$ is a power of 2. Divide $y$ and $z$ into two halves, each with $n/2$ bits.

\[
\begin{array}{c|c|c}
\hline
y & a & b \\
\hline
z & c & d \\
\hline
\end{array}
\]
Integer multiplication

Then

\[ y = a2^{n/2} + b \]
\[ z = c2^{n/2} + d \]

and so

\[ yz = (a2^{n/2} + b)(c2^{n/2} + d) \]
\[ = ac2^n + (ad + bc)2^{n/2} + bd \]

This computes \( yz \) with 4 multiplications of \( n/2 \) bit numbers, and some additions and shifts. Running time given by \( T(1) = c \), \( T(n) = 4T(n/2) + dn \), which has solution \( O(n^2) \) by the General Theorem. No gain over naive algorithm!

**Example 5.7:** Consider the recurrence

\[ T(n) = 4T(n/2) + n. \]

In this case, \( a^{\log_2 n} = n^{\log_2 4} = n^2 \). Thus, we are in Case 1, for \( f(n) \) is \( O(n^{2+v}) \) for \( v = 1 \). This means that \( T(n) \) is \( \Theta(n^2) \) by the master method.
Integer multiplication (Karatsuba algorithm)

• Consider the product
  \((a-b)(d-c) = (ad + bc) - (ac + bd)\)
• It contains two of the products we need \((ad\) and \(bc)\)
• Then
  \(yz = ac2^n + [(a-b)(d-c) + (ac+bd)]2^{n/2} + bd\)
• We need three multiplications of \(n/2\) bits and \(O(n)\) additional work

Therefore,
\[
T(n) = \begin{cases} 
  c & \text{if } n = 1 \\
  3T(n/2) + dn & \text{otherwise}
\end{cases}
\]
where \(c, d\) are constants.

Therefore, by our general theorem, the divide and conquer multiplication algorithm uses
\[
T(n) = O(n \log^3 n) = O(n^{1.59})
\]
bit operations.
def multiply(y, z):
    l = max(len(y), len(z))
    if l == 1:
        return [y[0] * z[0]]
    y = [0 for i in range(len(y), l)] + y;
    z = [0 for i in range(len(z), l)] + z;
    m0 = (l + 1) / 2
    a = y[:m0]
    b = y[m0:]
    c = z[:m0]
    d = z[m0:]

Remark: pad y and z so that they have the same length

Karatsuba algorithm (continued)

p0 = multiply(a, c)
p1 = multiply(add(a, b), add(c, d))
p2 = multiply(b, d)

z0 = p0
z1 = subtract(p1, add(p0, p2))
z2 = p2

Remark: compute
z1 = p1 - p0 - p2
Remark: compute
z0 b^l + z1 b^(l/2) + z2

z0prod = z0 + [0 for i in range(0, l)]
z1prod = z1 + [0 for i in range(0, l / 2)]

return add(add(z0prod, z1prod), z2)
Matrix multiplication (Strassen)

**Problem**: Given two matrices $Y$ and $Z$ compute $X = Y \times Z$.
Matrix multiplication

```python
def mult(Y, Z):
    X = zero(len(Y), len(Z[0]))

    for i in range(len(Y)):
        for j in range(len(Z[0])):
            for k in range(len(Z)):
                X[i][j] += Y[i][k] * Z[k][j]

    return X
```

Algorithm `mult(Y, Z)` is $O(n^3)$, can we do better?  

Matrix multiplication

Divide $X, Y, Z$ each into four $(n/2) \times (n/2)$ matrices.

$$
X = \begin{bmatrix}
    I & J \\
    K & L
\end{bmatrix}
$$

$$
Y = \begin{bmatrix}
    A & B \\
    C & D
\end{bmatrix}
$$

$$
Z = \begin{bmatrix}
    E & F \\
    G & H
\end{bmatrix}
$$
Matrix multiplication

Then

\[
\begin{align*}
I &= AE + BG \\
J &= AF + BH \\
K &= CE + DG \\
L &= CF + DH
\end{align*}
\]

Matrix multiplication

Let \( T(n) \) be the time to multiply two \( n \times n \) matrices.

\[
T(n) = \begin{cases} 
  c & \text{if } n = 1 \\
  8T(n/2) + dn^2 & \text{otherwise}
\end{cases}
\]

where \( c, d \) are constants.
Matrix multiplication

Therefore,

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

\[ = 8(8T(n/4) + d(n/2)^2) + dn^2 \]

\[ = 8^2T(n/4) + 2dn^2 + dn^2 \]

\[ = 8^3T(n/8) + 4dn^2 + 2dn^2 + dn^2 \]

\[ = 8^iT(n/2^i) + dn^2 \sum_{j=0}^{i-1} 2^j \]

\[ = 8^{\log n}T(1) + dn^2 \sum_{j=0}^{\log n-1} 2^j \]

\[ = cn^3 + dn^2(n - 1) \]

\[ = O(n^3) \]

Master Theorem case 1:

\[ f(n) \in O(n^{\log_2 8 - \varepsilon}) ? \]

\[ dn^2 \in O(n^{1-\varepsilon}) ? \text{ true for } \varepsilon=1 \]

Then \( T(n) \in \Theta(n^3) \)

---

Matrix multiplication

- The naïve Divide and Conquer algorithm is no better than the straightforward algorithm
- However, it gives us an insight on the next algorithm
- Strassen’s algorithm uses only 7 multiplications instead of 8
Strassen algorithm

Compute

\[
\begin{align*}
M_1 & := (A + C)(E + F) \\
M_2 & := (B + D)(G + H) \\
M_3 & := (A - D)(E + H) \\
M_4 & := A(F - H) \\
M_5 & := (C + D)E \\
M_6 & := (A + B)H \\
M_7 & := D(G - E)
\end{align*}
\]

Strassen algorithm

Then,\n
\[
\begin{align*}
I & := M_2 + M_3 - M_6 - M_7 \\
J & := M_4 + M_6 \\
K & := M_5 + M_7 \\
L & := M_1 - M_3 - M_4 - M_5
\end{align*}
\]
Strassen algorithm

\[ I := M_2 + M_3 - M_6 - M_7 \]
\[ = (B + D)(G + H) + (A - D)(E + H) \]
\[ - (A + B)H - D(G - E) \]
\[ = (BG + BH + DG + DH) \]
\[ + (AE + AH - DE - DH) \]
\[ + (-AH - BH) + (-DG + DE) \]
\[ = BG + AE \]

Strassen algorithm

\[ J := M_4 + M_6 \]
\[ = A(F - H) + (A + B)H \]
\[ = AF - AH + AH + BH \]
\[ = AF + BH \]
Strassen algorithm

\[ K := M_5 + M_7 \]
\[ = (C + D)E + D(G - E) \]
\[ = CE + DE + DG - DE \]
\[ = CE + DG \]

Strassen algorithm

\[ L := M_1 - M_3 - M_4 - M_5 \]
\[ = (A + C)(E + F) - (A - D)(E + H) \]
\[ - A(F - H) - (C + D)E \]
\[ = AE + AF + CE + CF - AE - AH \]
\[ + DE + DH - AF + AH - CE - DE \]
\[ = CF + DH \]
def strassen(Y, Z):
    if len(Y) <= 2:
        return mult(Y, Z)
    else:
        A, B, C, D = partition(Y)
        E, F, G, H = partition(Z)
        M1 = strassen(add(A, C), add(E, F))
        M2 = strassen(add(B, D), add(G, H))
        M3 = strassen(sub(A, D), add(E, H))
        M4 = strassen(A, sub(F, H))
        M5 = strassen(add(C, D), E)
        M6 = strassen(add(A, B), H)
        M7 = strassen(D, sub(G, E))
        I = sub(sub(add(M2, M3), M6), M7)
        J = add(M4, M6)
        K = add(M5, M7)
        L = sub(sub(sub(M1, M3), M4), M5)
        return recompose(I, J, K, L)

Analysis of Strassen algorithm

\[
T(n) = \begin{cases} 
    c & \text{if } n = 1 \\
    7T(n/2) + dn^2 & \text{otherwise}
\end{cases}
\]

where \( c, d \) are constants.
Analysis of Strassen algorithm

\[
T(n) = 7T(n/2) + dn^2 \\
= 7(7T(n/4) + d(n/2)^2) + dn^2 \\
= 7^2T(n/4) + 7dn^2/4 + dn^2 \\
= 7^3T(n/8) + 7^2dn^2/4^2 + 7dn^2/4 + dn^2 \\
= 7^iT(n/2^i) + dn^2 \sum_{j=0}^{i-1} (7/4)^j \\
= 7^{\log n}T(1) + dn^2 \sum_{j=0}^{\log n-1} (7/4)^j \\
= en^{\log 7} + dn^2(\frac{(7/4)^{\log n} - 1}{7/4 - 1}) \\
= en^{\log 7} + \frac{4}{3}dn^2(\frac{n^{\log 7}}{n^2} - 1) \\
= O(n^{\log 7}) \\
\approx O(n^{2.8})
\]

Master Thorem case 1:
\[f(n) \in O(n^{\log_7 7 - \epsilon})?\]
\[dn^2 \in O(n^{2.8-\epsilon})?\] true for \(\epsilon = 0.5\)
Then \(T(n) \in \Theta(n^{\log_7 7})\)

Discussion

- There is a large constant hidden which makes Strassen impractical, unless the matrices are large (\(n>45\)) and dense
- For sparse matrices there are faster methods
- Strassen is not as *numerically stable* as the naïve
- Sub-matrices at each level consume space
- FYI: the current best algorithm for dense matrices runs in \(O(n^{2.376})\)
- Lower bound \(\Omega(n^2)\) [for dense matrices]
Closest Pair Problem

- Let \( P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n) \) be a set \( S \) of \( n \) points in the plane
- **Problem:** Find the two closest points in \( S \)
- **Assumptions:**
  - \( n \) is a power of two
  - points are ordered by their \( x \) coordinate (if not, we can sort them in \( O(n \log n) \) time)
Closest-Pair Problem: Brute-force

- Compute the distance between every pair of distinct points
- Return the indexes of the points for which the distance is the smallest

Time complexity?

Closest-Pair: Divide and Conquer

**Step 1.** Divide the points in $S$ into two subsets $L$ and $R$ by a vertical line $x = c$ so that half the points lie to the left or on the line and half the points lie to the right or on the line ($c$ is the median of the $x$ coord)
Closest-Pair: Divide and Conquer

**Step 2.** Find recursively the closest pairs for $L$ and $R$. Let $d_1, d_2$ be the distances of the two closest pairs.
Set $d = \min\{d_1, d_2\}$

Closest Pair:  Divide and Conquer

**Step 3.** Consider the vertical strip $2d$-wide centered at $x = c$. Let $Y$ be the subset of points in this vertical strip of width $2d$
Closest Pair: Divide and Conquer

• **Observation 1:** if a pair of points $p_L, p_R$ has distance less than $d$, both points of the pair **must** be within $Y$

![Diagram of observation 1]

Closest Pair: Divide and Conquer

**Observation 2:** Since all the points within $L$ are at least $d$ units apart, at most 4 points can reside within the $d \times d$ square (same is true for $R$)

![Diagram of observation 2]
Closest Pair: Divide and Conquer

**Proof:** Let’s suppose (for sake of contradiction) that five or more points are found in a square of size $d \times d$. Divide the square into four smaller squares of size $d/2 \times d/2$. At least one pair of points must fall within the same smaller square: these two points will be at a distance $d/\sqrt{2} < d$, which leads to a contradiction.

**Consequence:** At most 8 points can reside within the $d \times 2d$ rectangle, because on each side all points are at least $d$ unit apart.
Closest Pair: Divide and Conquer

**Step 4.** For each point \( p \) in \( Y \), try to find points in \( Y \) that are within \( d \) units of \( p \). Only 7 points in \( Y \) that follow \( p \) need to be considered.

![Diagram showing close pairs and coincident points](image)

**Closest pair in Python**

```python
def closestPair(xP, yP):
    n = len(xP)
    if n <= 3:
        return bruteForceClosestPair(xP)
    Xl = xP[:n//2]
    Xr = xP[n//2:]
    Yl, Yr = [], []
    median = Xl[-1].x
    for p in yP:
        if p.x <= median:
            Yl.append(p)
        else:
            Yr.append(p)
```

**Remark:** \( xP \) and \( yP \) is the same of input points \((x,y)\), but \( xP \) is sorted by \( x \) and \( yP \) is sorted by \( y \).

**Remark:** \( Xl \) is the first half of the points sorted by \( x \), and \( Xr \) is the second half.

**Remark:** \( Yl \) contains the points (sorted by \( y \)) which have a \( x \) coordinate smaller than the median.
dl, pairl = closestPair(Xl, Yl)

\[ \text{dr, pairr} = \text{closestPair(Xr, Yr)} \]

dm, pairm = (dl, pairl) if dl < dr else (dr, pairr)

\[ \text{st} = \{p \text{ for } p \text{ in } yP \text{ if } \text{abs}(p.x - \text{median}) < dm \} \]

\[ n_{st} = \text{len(st)} \]

\[ \text{closest} = (dm, pairm) \]

if n_st > 1:
    for i in range(n_st-1):
        for j in range(i+1, min(i+8, n_st)):
            if d(st[i], st[j]) < closest[0]:
                closest = (d(st[i], st[j]), (st[i], st[j]))

return closest

Remark: variable \( \text{st} \) contains the points in the strip \([\text{median}-dm, \text{median}+dm]\) sorted by \( y \)

Remark: \( d(x,y) \) returns the distance between \( x \) and \( y \)

Analysis of the Closest-Pair Algorithm

- We can keep the points in \( Y \) stored in increasing order of their \( y \) coordinates, which is maintained by merging during the execution of step 4
- We can process the points in \( Y \) sequentially in linear time
- Running time is described by \( T(n) = 2T(n/2) + O(n) \)
- By the Master Theorem, \( T(n) \) is \( O(n \log n) \)
Linear-time selection

- **Problem:** Select the $i$-th smallest element in an unsorted array of size $n$ (assume distinct elements)
- **Trivial solution:** sort $A$, select $A[i]$; time complexity is $O(n \log n)$

- Can we do it in linear time? Yes, thanks to Blum, Floyd, Pratt, Rivest, and Tarjan
Linear-time selection

Select \( (A, \text{start, end, } i) \) /* \( i \) is the \( i \)-th order statistic */

1. divide input array \( A \) into \( \lfloor n/5 \rfloor \) groups of size 5
   (and one leftover group if \( n \% 5 \) is not 0)
2. find the median of each group of size 5 by sorting
   the groups of 5 and then picking the middle element
3. call Select recursively to find \( x \), the median of the \( \lfloor n/5 \rfloor \) medians
4. partition array around \( x \), splitting it into two arrays
   \( L \) (elements smaller than \( x \)) and \( R \) (elements bigger than \( x \))

5. \( k \triangleq \lfloor |L| + 1 \rfloor \)
   if \( (i = k) \) then return \( x \)
   else if \( (i < k) \) then Select \( (L, i) \)
   else Select \( (R, i - k) \)

[r] means the ceiling (rounding to the next integer) of real number \( r \)

Python linear-time selection

```python
def selection(a, rank):
    n = len(a)
    if n <= 5:
        return rank_by_sorting(a, rank)
    medians = [rank_by_sorting(a[i:i+5], 3)
               for i in range(0, n-4, 5)]
    median = selection(medians, (len(medians) + 1) // 2)
    L, R = [], []
    for x in a:
        if x < median:
            L += [x]
        else:
            R += [x]
    if rank <= len(L):
        return selection(L, rank)
    else:
        return selection(R, rank - len(L))
```

Stefano Lonardi, UCR
Example

Let us run Select(A, 1, 28, 11), where

\[ A = \{12, 34, 0, 3, 22, 4, 17, 32, 3, 28, 43, 82, 25, 27, 34, 2, 19, 12, 5, 18, 20, 33, 16, 33, 21, 30, 3, 47\} \]

Note that the elements in this example are not distinct.

Example

First make groups of 5

<table>
<thead>
<tr>
<th>12</th>
<th>34</th>
<th>0</th>
<th>3</th>
<th>22</th>
<th>4</th>
<th>17</th>
<th>32</th>
<th>3</th>
<th>28</th>
<th>43</th>
<th>2</th>
<th>20</th>
<th>30</th>
<th>3</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>17</td>
<td>82</td>
<td>19</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>25</td>
<td>12</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td>5</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>28</td>
<td>34</td>
<td>18</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Example

Then find medians in each group

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>25</th>
<th>2</th>
<th>20</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td>5</td>
<td></td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>17</td>
<td>34</td>
<td>12</td>
<td>21</td>
<td>19</td>
<td>47</td>
</tr>
<tr>
<td>34</td>
<td>32</td>
<td>43</td>
<td>19</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>28</td>
<td>82</td>
<td>18</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
</tbody>
</table>

## Example

Then find median of medians

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>25</th>
<th>2</th>
<th>20</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td>5</td>
<td></td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>17</td>
<td>34</td>
<td>12</td>
<td>21</td>
<td>19</td>
<td>47</td>
</tr>
<tr>
<td>34</td>
<td>32</td>
<td>43</td>
<td>19</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>28</td>
<td>82</td>
<td>18</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
</tbody>
</table>

12, 12, 17, 21, 30, 34
Example

Use 17 as the pivot value and partition original array

\[
\begin{array}{cccccccc}
0 & 4 & 25 & 2 & 20 & \boxed{3} \\
3 & 3 & 27 & 5 & 16 & \boxed{30} \\
12 & 17 & 34 & 12 & 21 & \boxed{47} \\
34 & 32 & 43 & 19 & 33 & \\
22 & 28 & 82 & 18 & 33 & \\
\end{array}
\]

12, 12, 17, 21, 30, 34

Example

After partitioning

\[L = \{12, 0, 3, 4, 3, 2, 12, 5, 16, 3\}\]

\[L \text{ contains 10 elements smaller than 17}\]

\{17\} \text{ this is the 11-th smallest}\n
\[R = \{34, 22, 32, 28, 43, 82, 25, 27, 34, 19, 18, 20, 33, 33, 21, 30, 47\}\]

\[R \text{ contains 17 elements bigger than 17}\]
Linear-time selection

- Finding the median of medians guarantees that $x$ causes a “good split”
- At least a constant fraction of the $n$ elements $\leq x$ and a constant fraction $> x$
- **Analysis**: we need to find the worst case for the size of $L$ and $R$

Linear-time selection: analysis

**Observation**: At least 1/2 of the medians found in step 2 are greater than the median of medians $x$. So at least half of the $[n/5]$ groups contribute 3 elements that are bigger than $x$, except for the one group with less than 5 elements and the group with $x$ itself
Linear-time selection: analysis

- Therefore there are
  \[ 3\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \geq (3n/10) - 6 \]
  elements are \( > x \) (or \( < x \))

- So worst-case split has at most \((7n/10) + 6\)
  elements in “big” section of the problem, that is:
  \[ \max\{|L|,|R|\} < (7n/10) + 6 \]

Running Time:

1. \( O(n) \) (break into groups of 5)
2. \( O(n) \) (sorting 5 numbers and finding median is \( O(1) \) time)
3. \( T(\lceil n/5 \rceil) \) (recursive call to find median of medians)
4. \( O(n) \) (partition is linear time)
5. \( T(7n/10 + 6) \) (maximum size of subproblem)

Recurrence relation

\[
T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) \quad n > 80
\]

\[
= \Theta(n) \quad n \leq 80
\]
Linear-time select: Analysis

Fact: \( T(n) = T(\lfloor n/5 \rfloor) + T(7n/10 + 6) + O(n) \) is \( O(n) \)

Proof:

Base case: easy (omitted).  
\[
T(n) = T(\lfloor n/5 \rfloor) + T(7n/10 + 6) + O(n) \\
\leq c\lfloor n/5 \rfloor + c(7n/10 + 6) + O(n) \\
\leq c(n/5 + 1) + 7cn/10 + 6c + O(n) \\
= cn - [c(n/10 - 7) - dn] \\
\leq cn
\]

Inequality \( cn/10 - 7c - dn \geq 0 \) is equivalent to \( c \geq 10dn/(n-70) \) when \( n>70 \). We can assume that \( n \geq 140 \), so that \( n/(n-70) \geq 2 \). In that case, choosing \( c \geq 20d \) will satisfy the inequality (there is nothing special about choosing \( n \geq 140 \), a different choice of \( n > 70 \) will require to choose a different \( c \)).

Reading assignment on Chapter 2

- Mergesort (section 2.3)
- Binary Search (page 50, box)
- Integer Multiplication (Karatsuba, section 2.1)
- Matrix Multiplication (Strassen, section 2.5)
- Closest pair (problem 2.32)
- Medians (section 2.4 covers randomized)
- Skip FFT