Divide-and-Conquer
Finding the closest pair of points
The Closest pair problem

- **Input:** a set of $n$ points in the plane (2D)
- **Output:** a pair of closest point
  - Find pair $p$ and $q$ such that $\text{dist}(p, q)$ is minimum over all pairs

\[
\text{dist}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}
\]
The Closest pair problem: Solution 1: Brute-Force

- **Try all pairs**
  - Find the distances between all pairs of points
  - Take the minimum of distances
- **Runtime:** $\Theta(n^2)$
The Closest pair problem: Solution 2: Recursive

- Assume you have the solution for \( n-1 \rightarrow r_1 \)
- Compute minimum distance from point \( n \) to all the \( n-1 \) points \( \rightarrow r_2 \)
- Minimum distance = \( \min(r_1, r_2) \)

**Runtime**: \( T(n) = T(n-1) + n - 1 = \Theta(n^2) \)
The Closest pair problem: One-dimensional points

- **Input:** n points on the real line
- If the points are sorted, then two closest points are neighbor to each other
- **Algorithm:**
  - Sort the points
  - examine the points from left to right to find the minimum distance
- **Runtime:** $O(n \log n) + O(n)$
The Closest pair: Divide and Conquer: Idea

- **Divide** the set into two equal parts: Left half, L, right half, R each with \( n/2 \) points
- **Conquer**: Recursively find the closest pair in L and R
  - Suppose they are \( d_1 \) and \( d_2 \) and let \( d = \min(d_1, d_2) \)
- **Combine**: find closest pairs crossing line \( m \)
- Return \( \min(d_1, d_2, d_{\text{cross}}) \)

**Runtime**: \( T(n) = 2 \ T(n/2) + \text{Time for combine} \)
The Closest pair: Divide and Conquer: Idea

- Suppose $d$ is the minimum distance of all pairs in $L$ or $R$
- **Observation 1:**
  - If the closest pair crosses $m$, they must be within $2d$-wide strip around $m$. Why?

Runtime: $T(n)=2T(n/2) + O(n^2)$

In the worst case all points will be within the strip

$$d = \min(d_1, d_2)$$
The Closest pair: Divide and Conquer: Idea

- If p in L (within the strip) and q in R (within the strip) and dist(p, q) < d, then q must be in the green area.
  - In other words |y_q - y_p| < d
- Why?

\[ d = \min(d_1, d_2) \]
The Closest pair: Divide and Conquer: Idea

- How many points can be in the green area?
  - at most 8 points. Why?
- Implication: for each $p$ within the strip we can find in constant time if there is a point $q$ on the other side so that $\text{dist}(p,q) < d$
How many points can be in a $2d \times d$ rectangle?

$d = \min(d_1, d_2)$

How many points can be in the green area?
How many points can be in a 2dxd rectangle?

No two points lies in the same d/2 x d/2 square.

Proof, if so, their distance would be less than d.

$d$ is set to be the minimum of the closest pairs on L and R.
Closest pair: Divide and Conquer: Combine step

- **Sort** the points within the strip according to their y-coordinates
- For each point p, check the distance to the next 7 points
  - to check if \( \exists \) q so that \( \text{dist}(p, q) < d \)
- Similar to one-dimensional problem

- Now the combine step is \( O(n) \)

**When to sort the points according to the y-coordinates?**

\[ d = \min(d_1, d_2) \]
Closest pair: Divide and Conquer Algorithm

- Sort all the points according to their x-coordinates
- Sort all the points according to their y-coordinates

- **Divide:**
  - Divide the plane by the vertical line that bisects the set into L and R
  - If several points lie on the vertical line, we divide them arbitrarily

- **Conquer:**
  - find the minimum distance in each part
  - Let $d_1$ be the minimum distance in L and $d_2$ be the minimum distance in R

- **Combine:**
  - $d = \min(d_1, d_2)$
  - Check to see if there is a point in L with a distance $< d$ to a point in R
  - Return the overall minimum

**Runtime:** $T(n) = 2 \ T(n/2) + O(n)$
Closest pair: Divide and Conquer Algorithm: Pseudocode

Sort the points according to their x-coordinates
Sort the points according to their y-coordinates

def closest_Pair(points):
    #Input: points (a set of n points in the plane)
    #Output: d (the distance between the two closest points in the set)
    if len(points) <= 3:
        return brute_force(points)
    mid = len(n) / 2
    P1 = points[0:mid]
    P2 = points[mid:len(n)]
    d1 = closest_pair(P1)
    d2 = closest_pair(P2)
    d = min(d1, d2)

    Points_to_check: Scan through the list of points sorted according to y-coordinates and include those that are within the strip

    for i = 0 to len(points_to_check):
        for j = 0 to 7:
            t = distance(points_to_check[i], points_to_check[i+j])
            if t < d:
                d = t

    return d
Divide-and-Conquer Algorithm

Integer Multiplication
Integer Multiplication

- **Input:**
  - 2 n-digit integers, X, Y

- **Output:**
  - Z = X . Y
Integer Multiplication: brute-force (grade school) method

Example:

\[
\begin{array}{c}
\times \\
2981 \\
1234 \\
\hline \\
11924 \\
8943 \rightarrow \text{One shift} \\
5962 \rightarrow 2 \text{ shifts} \\
2981 \rightarrow 3 \text{ shifts} \\
\hline \\
3678554
\end{array}
\]

Runtime for: $O(n^2)$

- We are counting number of shift, addition, and multiplication operations. Each one of these operations are constant time
Divide-and-Conquer: First Algorithm

- Example:
  - Suppose we want to multiply two 2-digit integers: 32, 45
  - We can multiply four 1-digit integers and then use add/shift to obtain the result
Divide-and-Conquer: First Algorithm

- Divide each number into n/2 digits integer
- Recursively multiply n/2 digits numbers
- Merge solutions: by shift and addition operations
Divide-and-Conquer: First Algorithm

- $X = ab$, $Y = cd$, $X = a^{\frac{n}{2}} + b$, $Y = c^{\frac{n}{2}} + d$
  - Each of $a$, $b$, $c$, $d$ is of size $n/2$
- $XY = (a^{\frac{n}{2}} + b) \cdot (c^{\frac{n}{2}} + d) = ac^{\frac{n}{n}} + (ad + bc)\cdot 10^{n/2} + bd$

<table>
<thead>
<tr>
<th>29 81</th>
<th>1234</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>shifts</td>
<td></td>
</tr>
<tr>
<td>29 x 12</td>
<td>4</td>
</tr>
<tr>
<td>29 x 34</td>
<td>2</td>
</tr>
<tr>
<td>81 x 12</td>
<td>2</td>
</tr>
<tr>
<td>81 x 34</td>
<td>0</td>
</tr>
</tbody>
</table>
Divide-and-Conquer: First Algorithm

- **Pseudocode:**

  ```plaintext
  function DC-Mult1(X, Y):
  
  Base Case:
  if (X or Y is single digit)
    return X.Y
  
  a = first n/2 digits of X
  b = last n/2 digits of X
  c = first n/2 digits of Y
  d = last n/2 digits of Y
  
  ac = DC-Mult(a, c)
  ad = DC-Mult(a, d)
  bc = DC-Mult(b, c)
  bd = DC-Mult(b, d)
  
  return ac*10^n + (ad + bc)*10^{n/2} + bd
  
  Shift: O(n)
  3 additions: O(n)
  ```

- **Runtime:** $4T(n/2) + O(n)$
- **Using Master Theorem:** $a=4$, $b=2$, $k=1$, $a^k > b^k = 2$

\[
T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)
\]
Divide-and-Conquer: Karatsuba (1960)

\[ XY = (a \ 10^{n/2} + b) \cdot (c \ 10^{n/2} + d) = ac \ 10^n + (ad + bc) \ 10^{n/2} + bd \]

**Observation:** We want to have sum of \( ad + bc \). We do not need \( ad \) or \( bc \) themselves.

\[(a + b) \times (c + d) = ac + (ad + bc) + bd \]

\[ad + bc = (a + b) \times (c + d) - ac - bd\]

Number of multiplications are 3:
- \( ac \)
- \( bd \)
- \((a + b) \times (c + d)\)

**Runtime:** \( T(n) = 3T(n/2) + O(n) = O(n^{\log_2{3}}) = O(n^{1.59})\)
Integer Multiplication

- **Exercise:** Generalize Karatsuba to do 5 size n/3 subproblems
  - Results in an algorithm with runtime $\Theta(n^{1.46})$

- What if $X$ has $n$ digits and $Y$ has $m$ digits and $n \gg m$
  - Break $X$ into $O(n/m)$ chunks, each having $m$ digits
  - Multiply each chunk by $Y$
  - Add up all products, taking into account the shifts

  Runtime: $o\left(\left(\frac{n}{m}\right)m^{\log_2 3}\right)$

- Asymptotically faster methods for larger $n$:
  - Schonhage & Strassen (1971): $O(n (\log n) (\log \log n))$ (used in practice)
  - Recent breakthrough (2019): $O(n \log n)$
Matrix Multiplication

**Input:** 2 nxn matrices A and B

**Output:** C = AxB

count operations {+, −, ×}
Matrix Multiplication

Brute-force Algorithm: \[ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = (a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}) \]
Matrix Multiplication

Pseudocode:

```plaintext
function MM(A, B):
    C = int[n][n];
    for i = 1 to n
        for j = 1 to n
            C[i][j] = 0
            for k = 1 to n
                C[i][j] += A[i][k].B[k][j]
    return C
```

Runtime: \(O(n^3)\)

- count number of arithmetic operations. Assume each operation \{+,-,\times\} takes constant time
Matrix Multiplication: Divide-and-Conquer

Divide matrices into blocks of size n/2

\[
\begin{array}{c|c}
A_{11} & A_{12} \\
\hline
A_{21} & A_{22}
\end{array}
\quad \times \quad
\begin{array}{c|c}
B_{11} & B_{12} \\
\hline
B_{21} & B_{22}
\end{array}
\quad = \quad
\begin{array}{c|c}
C_{11} & C_{12} \\
\hline
C_{21} & C_{22}
\end{array}
\]

Each one of \(A_{11}, \ldots, A_{22}, B_{11}, \ldots, B_{22}, C_{11}, \ldots, C_{22}\) are n/2 x n/2 matrices and + is matrix addition operation.

\[
\begin{align*}
C_{11} & = A_{11} \times B_{11} + A_{12} \times B_{21} \\
C_{12} & = A_{11} \times B_{12} + A_{12} \times B_{22} \\
C_{21} & = A_{21} \times B_{11} + A_{22} \times B_{21} \\
C_{22} & = A_{21} \times B_{12} + A_{22} \times B_{22}
\end{align*}
\]

\[
C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} = (A_{i1}B_{1j} + A_{i2}B_{2j} + \ldots + A_{in}B_{nj})
\]
Matrix Multiplication: Divide-and-Conquer

**Pseudocode:**

```plaintext
function DC1-MM(A, B)
    C_{11} = DC1-MM (A_{11} , B_{11}) + DC1-MM (A_{12} \times B_{21})
    C_{12} = DC1-MM (A_{11} \times B_{12}) + DC1-MM (A_{12}B_{22})
    C_{21} = DC1-MM (A_{21} , B_{11}) + DC1-MM (A_{22} \times B_{21})
    C_{22} = DC1-MM (A_{21} , B_{12}) + DC1-MM (A_{22} , B_{22})
    return C = C_{11} C_{21} C_{12} C_{22}
```

**Runtime:** $T(n) = 8 T(n/2) + c (n/2)^2$

Using Master Theorem: $a=8$, $b=2$, $k=2$, $a > b^k \rightarrow n^{\log_2 8} = O(n^3)$
Divide-and-Conquer: Strassen’s algorithm

- Like idea for integer multiplication, reduce the number of multiplications
- Define the following matrices and use them to compute \( C \)

\[
\begin{align*}
Z_1 &= A_{11} (B_{12} - B_{22}) \\
Z_2 &= (A_{11} + A_{12}) B_{22} \\
Z_3 &= (A_{21} + A_{22}) B_{11} \\
Z_4 &= A_{22} (B_{21} + B_{11}) \\
Z_5 &= (A_{11} + A_{22}) (B_{11} + B_{22}) \\
Z_6 &= (A_{12} - A_{22}) (B_{21} + B_{22}) \\
Z_7 &= (A_{11} - A_{21}) (B_{11} + B_{12}) \\
C_{11} &= Z_5 + Z_4 - Z_2 + Z_6 \\
C_{12} &= Z_1 + Z_2 \\
C_{21} &= Z_3 + Z_4 \\
C_{22} &= Z_5 + Z_1 - Z_3 + Z_7
\end{align*}
\]
Divide-and-Conquer: Strassen’s algorithm

- 8 additions
- 7 multiplications
- 10 additions

**Runtime:**

\[ T(n) = 7 \cdot T(n/2) + O(n^2) \]

Using Master Theorem: \( a=7, b=2, k=2, a=7, \ a > b^k = 4 \)

Therefore:

\[ T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7}) = \Theta(n^{2.807}) \]

**Exercise:** Try to solve the above recurrence relation using recursion tree method
The Centrality of Matrix Multiplication

- Suppose two $n \times n$ matrices can be multiplied in time $O(n^\omega)$: $2 \leq \omega \leq 3$
- Many problems can be solved in $O(n^\omega)$:
  - computing $\det A$
  - computing $A^{-1}$
- Many problems are at least as difficult as matrix multiplication
Reduction of matrix multiplication to matrix inversion

Suppose you want to multiply matrices $A$ and $B$ and you have an algorithm for matrix inversion:

$$M = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix} \quad M^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}$$
Reduction of matrix inversion to matrix multiplication

- If we can multiply two matrices fast, we can also invert them fast
- Suppose we want to find the inverse of Matrix M

\[
M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad M^{-1} = \begin{pmatrix} I_n & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I_n \end{pmatrix}
\]

Assuming \( A \) and \( S = D - CA^{-1}B \) are invertible

**Runtime:** \( T(n) = 2 \ T(n/2) + c \ (n/2)^\omega \)

Using Master Theorem, \( T(n) = O(n^\omega) \)
The Selection Problem
The selection problem

**Input:** a collection of $n$ elements and a number $i$

**Output:** return the $i$-th order statistics of the collection

**Definition.** For a collection of data, **$i$-th order statistics** is the $i$-th smallest value in the data set

- $i=1 \rightarrow$ minimum
- $i=n \rightarrow$ maximum
- $i= \left\lfloor \frac{n+1}{2} \right\rfloor$ or $\left\lceil \frac{n+1}{2} \right\rceil \rightarrow$ median
Selection: Brute-force solution

Sort and return the element at position $i$

**Worst-case running time** $= \Theta(n \lg n)$,

using merge sort or heapsort (not quicksort)
Selection Problem: Divide-and-Conquer

**Idea:** Use partitioning

(the same subroutine used in quicksort)

- **Runtime:** similar to quicksort, depends on the choice of the pivot
- **Best case:**
  \[ T(n) = T(n/2) + \Theta(n) = \Theta(n) \]
- **Worst case:**
  \[ T(n) = T(n-1) + \Theta(n) = \Theta(n^2) \]
- **Average Case:**
  \[ \Theta(n) \]
- **Randomized Algorithm:** Expected runtime:
  \[ \Theta(n) \]

---

def select(A, i):
    q = partition(A)
    if q == k:
        return A[q]
    else if q > i:
        return select(A[0 ... q-1] , i]
    else:
        return select(A[q+1 ... len(A)-1], i-q-1)
Selection Problem

- Works fast: linear expected time
- Excellent algorithm in practice
- But, the worst case is very bad: $\Theta(n^2)$.
- Question:
  - Is there a deterministic algorithm that runs in linear time in the worst case?
- Answer:
  - Yes, due to Blum, Floyd, Pratt, Rivest, and Tarjan [1973].

**IDEA : Generate a good pivot recursively**
Selection Problem: QuickSelect Algorithm

- Divide the input elements into blocks of five elements each.
  - The leftover elements are put into their own block
- Determine the median of each 5-element block
  - This can be done in constant time since the number of elements in each block is a constant
- Recursively determine the median of these medians
- Use that median as a pivot
- Make a recursive call on an input of size at most $7n/10$
| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $a_9$ |
1) Divide the n elements into blocks of 5 each
1) Divide the n elements into blocks of 5 each
2) Determine the median of each 5-element block
1) Divide the n elements into blocks of 5 each
2) Determine the median of each 5-element block
1) Divide the n elements into blocks of 5 each.
2) Determine the median of each 5-element block.
3) Recursively determine the median of these medians. Select that median as the pivot.
The pivot is larger than $\frac{3}{10}$ of the total elements

- Number of elements larger than pivot is at most $\frac{7}{10}$ of the total elements

The pivot is smaller than $\frac{3}{10}$ of the total elements

- Number of elements smaller than pivot is at most $\frac{7}{10}$ of the total elements
QuickSelect Algorithm: Runtime

\[ T(n) = T\left(\left\lfloor \frac{n}{5} \right\rfloor \right) + T\left(\left\lfloor \frac{7n}{10} \right\rfloor \right) + \Theta(n) \quad n > 50 \]

\[ T(n) = \Theta(1) \quad n \leq 50 \]

\[ T(n) \leq \left(\frac{n}{5}\right) + T\left(\left\lfloor \frac{7n}{10} \right\rfloor \right) + kn \quad n > 50 \]

\[ T(n) \leq k \quad n \leq 50 \]
- We guess $T(n) = O(n)$. Prove it by induction.
- Theorem. Prove for all $n \geq 1$ there exist $c > 0$, where $T(n) \leq cn$
- Base case: if $1 \leq n \leq 50 \rightarrow T(n) \leq c \leq cn$
- Inductive Step: Assume that the claim hold for all values less than $n$

$$
T(n) \leq T\left(\lfloor n/5 \rfloor\right) + T\left(\lfloor 7n/10 \rfloor\right) + kn
$$

$$
\leq c\left\lfloor n/5 \right\rfloor + c\left\lfloor 7n/10 \right\rfloor + kn
$$

$$
\leq c(n/5) + c(7n/10) + kn
$$

$$
= c(9n/10) + kn
$$

Which is true when $c \geq 10k$
Question

- What if we divide the data into group of 3? Do we still have a linear-time algorithm?
- How about dividing the data into group of 7? Do we still have a linear-time algorithm?