Algorithms and Lab (CSE130)

More on Divide and Conquer Approach

Muhammad Tariq Mahmood

tariq@koreatech.ac.kr
School of Computer Science and Engineering

Note: These notes are prepared from the following resources.

- (main text)Foundations of Algorithms, by Richard Neapolitan and Kumarss Naimipour
- Python Algorithm (파이썬 알고리즘) by Y.K. Choi (2021) (Korean)
- Introduction to the Design and Analysis of Algorithms by Anany Levitin
- Introduction to Algorithms, by By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
- https://www.geeksforgeeks.org

Contents

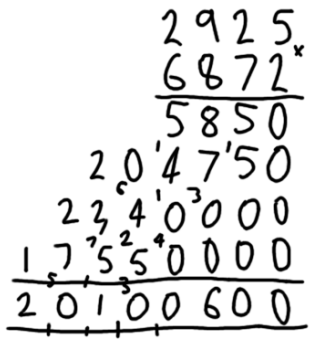
Multiplication of Large Integers

- We are given two integers u and v. Both have a length of length n digits. We want to find the product uv of these two numbers x and y. i.e. $uv = u * v$
- The size of the problem is n. The more digits in u and v the harder the problem.
- Some applications such as cryptography, require manipulation of integers that are over 100 decimal digits long.
- Since such integers are too long to fit in a single word (memory) of a computer, so they require special treatment.
- This practical need requires investigations of algorithms for efficient manipulation of large integers.
- **divide-and-conquer** approach can lead to an efficient manipulation of large integers.

Multiplication of Large Integers (cont...)

The grade-school algorithm

- Algorithm: Steps in grade-school algorithm
  1. Break the second number into units, tens, hundreds, thousands etc.
  2. Start with the units. Multiply the units from the second number by each digit in the first number.
  3. Do the same with the tens but add a zero before your answer.
  4. Do the same with the hundreds but add two zeros before your answer.
  5. Continue through the digits of the second number adding an additional zero before your answer each time.
  6. Sum all these partial multiplications to get the final answer.
- Complexity : $T(n) = 4 \times n^2 \in \Theta(n^2)$



- Let's run through the grade-school algorithm on two four digits integers. 2925×6872.

Multiplication of Large Integers (cont...)

Karatsuba's algorithm

- The key to understanding Karatsuba's multiplication algorithm is remembering that a digit can be split into 2 smaller digits
- Examples

$$\underbrace{567832}_{6\ digits} = \underbrace{567}_{3\ digits} \times 10^3 + \underbrace{832}_{3\ digits}$$

$$\underbrace{9423723}_{7\ digits} = \underbrace{9423}_{4\ digits} \times 10^3 + \underbrace{723}_{3\ digits}$$

- In general, if n is the number of digits in the integer u, we will split the integer into two integers, one with [n/2] and the other with [n/2], as follows:

$$\underbrace{u}_{n\ digits} = \underbrace{x}_{\lceil \frac{n}{2} \rceil\ digits} \times 10^m + \underbrace{y}_{\lfloor \frac{n}{2} \rfloor\ digits}$$

where, the exponent m of 10 is given by $m = \lfloor \frac{n}{2} \rfloor$

- If we have two n-digit integers

$$\begin{cases} \underbrace{u}_{n\ digits} = \underbrace{x}_{\lceil \frac{n}{2} \rceil\ digits} \times 10^m + \underbrace{y}_{\lfloor \frac{n}{2} \rfloor\ digits} \\ \underbrace{v}_{n\ digits} = \underbrace{w}_{\lceil \frac{n}{2} \rceil\ digits} \times 10^m + \underbrace{z}_{\lfloor \frac{n}{2} \rfloor\ digits} \end{cases}$$

the the product of uv can be written as

$$uv = (x \times 10^m + y)(w \times 10^m + z)$$
$$uv = xw \times 10^{2m} + (xz + wy) \times 10^m + yz$$

Multiplication of Large Integers (cont...)

- Note that there are four multiplications xw, xz, wy, yz, we want to reduce the number of multiplications
- Let

$$\begin{aligned} r &= (x+y)(w+z) \\ r &= xw + (xz + wy) + yz \\ xz + wy &= r - xw - yz \end{aligned}$$

- Now, we have three multiplications $\underbrace{(x+y)(w+z)}_{r}$, xw, yz. Thus, the product of two big integers can be computed as

$$uv = xw \times 10^{2m} + (r - xw - yz) \times 10^{m} + yz$$

- The algorithm is based on idea of using divide-and-conquer by splitting an n-digit integer into two integers of approximately n/2 digits.

Multiplication of Large Integers (cont...)

Algorithm for multiplication of large integers

- **Problem:** Multiply two large integers, u and v.
- **Inputs:** large integers u and v.
- **Outputs:** the product of u and v.
- **Complexity Analysis:**

$$\begin{cases} T(1) = 1, & n = 1 \\ T(n) = 3T\left(\frac{n}{2}\right) + 4n, & n > 1 \end{cases}$$

Using Master Theorem

$$a = 3,\ b = 2,\ k = 1, b^k = 2,$$
$$a > b^k \ \ i.e.\ (case - 3)$$

$$T(n) \in \Theta\left(n^{\lg_b(a)}\right) = \Theta\left(n^{\lg_2(3)}\right) \approx \Theta\left(n^{1.58}\right)$$

- **Pseudo-code**

```
1:  procedure PROD( u, v)
2:      x, y, w, z, r, p, q, n, m
3:      n = max(digits in u, digits in v)
4:      if ((u == 0||v == 0)) then
5:          return 0
6:      else
7:          if (n <= threshold) then
8:              return u × v obtained in the usual way
9:          else
10:             m = [n/2]
11:             x = u  divide  10^m
12:             y = u  rem  10^m
13:             w = v  divide  10^m
14:             z = v  rem  10^m
15:             r = prod(x + y, w + z)
16:             p = prod(x, w)
17:             q = prod(y, z)
18:             return p × 10^2m + (r − p − q) × 10^m + q
19:         end if
20:     end if
21: end procedure
```

Matrix multiplication

- A matrix is a two-dimensional array of numbers, called the **entries** of the matrix. A matrix with m rows and n columns is called an $m \times n$ matrix. The values m and n are referred to as the **dimensions** of the matrix. A **square** matrix is one in which the number of rows and columns are equal (i.e. $m = n$).

- Consider two matrices are being multiplied $C_{2\times 2} = A_{2\times 2} \times B_{2\times 2}$

$$\begin{bmatrix} c_{11} & c_{21} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{21} \\ b_{21} & b_{22} \end{bmatrix}$$

- The product $C_{2\times 2} = A_{2\times 2} \times B_{2\times 2}$ is given by

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

- Example

$$\underbrace{\begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} 2\times 5 + 3\times 6 & 2\times 7 + 3\times 8 \\ 4\times 5 + 1\times 6 & 4\times 7 + 1\times 8 \end{bmatrix}}_{A\times B}$$

$$= \underbrace{\begin{bmatrix} 28 & 38 \\ 26 & 36 \end{bmatrix}}_{A\times B}$$

- Note that it requires eight multiplications and four additions.

- for larger matrices, $C_{n\times n} = A_{n\times n} \times B_{n\times n}$ is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Matrix multiplication (cont...)

- **brute-force algorithm**

```
1:  procedure matrixMultuply(A[][],B[][])
2:      integer i, j, k
3:      number C[][]
4:      for (i = 1; i <= n; i++) do
5:          for (j = 1; j <= n; j++) do
6:              C[i][j] = 0
7:          end for
8:      end for
9:      for (i = 1; i <= n; i++) do
10:         for (j = 1; j <= n; j++) do
11:             for (k = 1; k <= n; k++) do
12:                 C[i][j] = C[i][j] + A[i][k] * B[k][j]
13:             end for
14:         end for
15:     end for
16:     return C
17: end procedure
```

Complexity Function:

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n}\left(\sum_{j=1}^{n} 1\right) + \left(\sum_{i=1}^{n}\left(\sum_{j=1}^{n}\left(\sum_{k=1}^{n} 1\right)\right)\right) + 1 \\
&= \sum_{i=1}^{n} n + \sum_{i=1}^{n}\left(\sum_{j=1}^{n} n\right) + 1 \\
&= n^2 + \sum_{i=1}^{n}\left(\sum_{j=1}^{n} n\right) + 1 \\
&= n^2 + \sum_{i=1}^{n} n^2 + 1 \\
&= n^2 + n^3 + 1 \\
&= n^3 + n^2 + 1 \in \Theta(n^3)
\end{aligned}
$$

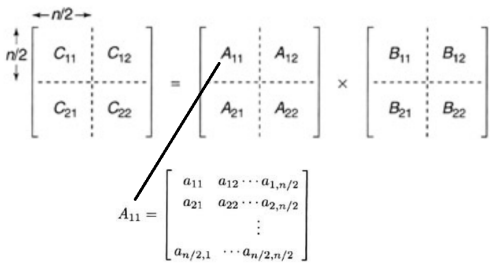
Matrix multiplication (cont...)

- In 1969, Strassen published an algorithm whose time complexity is better than cubic in terms of both multiplications and additions/subtractions.
- Theorem Let A and B be two square $n \times n$ matrices, where n is even. Let A_{11}, A_{12}, A_{21}, and A_{22} represent the four $\frac{n}{2} \times \frac{n}{2}$ submatrices of A that correspond to its four **quadrants**.
- For example, A_{11} consists of rows 1 through $n/2$ of A whose entries are restricted to columns 1 through $n/2$. Similarly, A_{12} consists of rows 1 through $n/2$ of A whose entries are restricted to columns $n/2+1$ through n. Finally, A_{21} and A_{22} represent the bottom half of A.
- In Strassen's method, first we divide each matrix (A and B) into four sub-matrices with sizes $\left(\frac{n}{2} \times \frac{n}{2}\right)$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

- Example

$$\begin{pmatrix} \begin{pmatrix} 2 & -1 \\ 4 & -2 \end{pmatrix} & \begin{pmatrix} -1 & 3 \\ 5 & 0 \end{pmatrix} \\ \begin{pmatrix} 4 & 1 \\ -3 & 0 \end{pmatrix} & \begin{pmatrix} 2 & 6 \\ -2 & 3 \end{pmatrix} \end{pmatrix}$$



Matrix multiplication (cont...)

- The principal insight of the algorithm lies in the discovery that we can find the product C of two 2×2 matrices A and B with just seven multiplications as opposed to the eight required by the brute-force algorithm

- For two matrix multiplication $C_{2\times2} = A_{2\times2} \times B_{2\times2}$

$$\begin{bmatrix} c_{11} & c_{21} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{21} \\ b_{21} & b_{22} \end{bmatrix}$$

Strassen determined that if we have

$$\begin{aligned} m_1 &= (a_{11} + a_{22}) \times (b_{11} + b_{22}) \\ m_2 &= (a_{21} + a_{22}) \times b_{11} \\ m_3 &= a_{11} \times (b_{12} - b_{22}) \\ m_4 &= a_{22} \times (b_{21} - b_{11}) \\ m_5 &= (a_{11} + a_{12}) \times b_{22} \\ m_6 &= (a_{21} - a_{11}) \times (b_{11} + b_{12}) \\ m_7 &= (a_{12} - a_{22}) \times (b_{21} + b_{22}) \end{aligned}$$

- Then the product C is computed as

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

where

$$\begin{aligned} c_{11} &= m_1 + m_4 - m_5 + m_7 \\ c_{12} &= m_3 + m_5 \\ c_{21} &= m_2 + m_4 \\ c_{22} &= m_1 + m_3 - m_2 + m_6 \end{aligned}$$

- Note that, to multiply two 2×2 matrices, Strassen's algorithm makes seven multiplications and 18 additions/subtractions, whereas the brute-force algorithm requires eight multiplications and four additions.

Matrix multiplication (cont...)

- **Strassen's Algorithm (Pseudo-code)**
  - **Problem:** Determine the product of two $n \times n$ matrices where n is a power of 2.
  - **Inputs:** an integer n that is a power of 2, and two nxn matrices A and B.
  - **Outputs:** the product C of A and B.

```
1:  procedure STRASSEN(integer n, matrix A, matrix B)
2:      matrix C
3:      if (n <= threshold) then
4:          compute C = A × B using the standard algorithm
5:      else
6:          divide A into four sub matrices A11, A12, A21, A22 having sizes n/2
7:          divide B into four sub matrices B11, B12, B21, B22, having sizes n/2
8:          M1 = strassen(n/2, (A11 + A22), (B11 + B22))      ▷ Compute M1 matrix
9:          M2 = strassen(n/2, (A21 + A22), B11)              ▷ Compute M2 matrix
10:         M3 = strassen(n/2, A11, (B12 − B22))              ▷ Compute M3 matrix
11:         M4 = strassen(n/2, A22, (B21 − B11))              ▷ Compute M4 matrix
12:         M5 = strassen(n/2, (A11 + A22), B22)              ▷ Compute M5 matrix
13:         M6 = strassen(n/2, (A21 − A11), (B11 + B12)       ▷ Compute M6 matrix
14:         M7 = strassen(n/2, (A12 − A22), (B21 + B22))      ▷ Compute M7 matrix
15:         C11 = (M1 + M4 − M5 + M7)                         ▷ Compute C11 matrix
16:         C12 = (M3 + M5)                                   ▷ Compute C12 matrix
17:         C21 = (M2 + M4                                    ▷ Compute C21 matrix
18:         C22 = (M1 + M3 − M2 + M6)                         ▷ Compute C22 matrix
19:         Compute C matrix from C11, C12, C21, C22
20:     end if
21:     return C
22: end procedure
```

- **Worse-Case Time Complexity Analysis of Strassen's Algorithm**

  - Time Complexity for **No. of multiplication operations**
    - Complexity Function

$$\begin{cases} T(1) = 1, & n = 1 \\ T(n) = 7T\left(\frac{n}{2}\right), & n > 1 \end{cases}$$

    - Solution

$$T(n) = n^{\lg 7}$$

    - Complexity in terms of asymptomatic notations

$$T(n) \in \Theta\left(n^{2.81}\right)$$

  - Time Complexity for **No. of Subtraction/Addition operations**
    - Complexity Function

$$T(n) = \begin{cases} T(1) = 0, & n = 1 \\ 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2, & n > 1 \end{cases}$$

    - Solution (Please see Appendix B.20)

$$T(n) = 6n^{\lg 7} - 6n^2$$

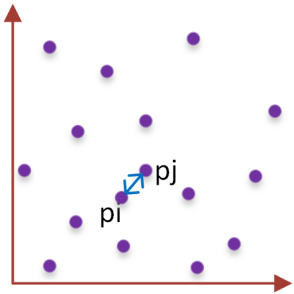
    - Complexity in terms of asymptomatic notations

$$T(n) \in \Theta\left(n^{2.81}\right)$$

Closest-Pair Problem

- The closest-pair problem calls for finding the two closest points in a set of n points in d-dimensions.
- For simplicity, we consider the two-dimensional case of the closest-pair problem.
- The distance between two points $p_i(x_i, y_i), p_j(x_j, y_j)$ is the standard Euclidean distance.

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$



- Fundamental problem in many applications as well as a key step in many algorithms.
  - Dynamic minimum spanning trees
  - Straight skeletons and roof design
  - Collision detection applications
  - Hierarchical clustering
  - Robot Motion Planning
  - Traveling salesman heuristics
  - Greedy matching

Closest-Pair Problem (cont...)

- A brute-force algorithm for the closest pair problem
  - Input: A list P of n ($n \geq 2$) points $p_1(x_1, y_1), ..., p_n(x_n, y_n)$
  - Output: The distance d between the closest pair of points
  - The brute-force algorithm:

```
1: procedure CP( P[])
2:     d = ∞
3:     for (i = 1; i <= n − 1; i + +) do
4:         for (j = i + 1; j <= n; j + +) do
5:             d = min {d, sqrt(x_i − x_j)^2 + (y_i − y_j)^2}
6:         end for
7:     end for
8:     return d
9: end procedure
```

  - Complexity Analysis

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 \\
&= \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\
&= n(n-1) - \frac{n(n-1)}{2} \\
&= \frac{n(n-1)}{2} \in \Theta(n^2)
\end{aligned}
$$

- The closest-pair problem can be posed in the k-dimensional space, in which the Euclidean distance between two points $p_i(x_1, x_2, ..., x_k), p_j(x_1, x_2, ..., x_k)$ is defined as

$$
d(p_i, p_j) = \sqrt{\sum_{s=1}^{k} (x_s^{(i)} - x_s^{(j)})^2}
$$

Closest-Pair Problem (cont...)

- A brute-force algorithm for the closest pair problem in k-dimensional space
- Input: A list P of n ($n \geq 2$) points $p_1(x_1^{(1)}, x_2^{(1)}, ..., x_k^{(1)}), ..., p_n(x_1^{(n)}, x_2^{(n)}, ..., x_k^{(n)})$
- Output: The distance d between the closest pair of points

  - The brute-force algorithm:

```
1:  procedure CP( P[])
2:      for (i = 1; i <= n − 1; i + +) do
3:          for (j = i + 1; j <= n; j + +) do
4:              dn = 0
5:              for (s = 1; s <= k; s + +) do
6:                  dn = dn + (x_s^(i) − x_s^(j))^2
7:              end for
8:              d = min {d, sqrt(dn)}
9:          end for
10:     end for
11:     return d
12: end procedure
```

  - Complexity Analysis

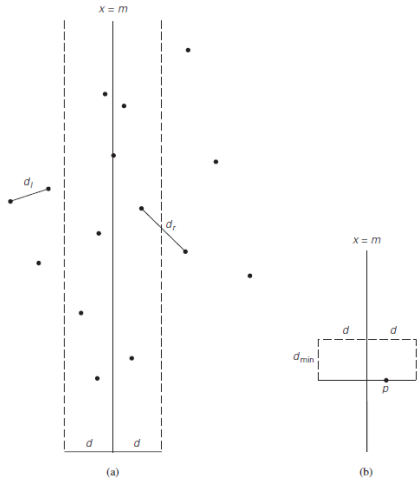
$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{s=1}^{k} 1 \\
&= k \sum_{i=1}^{n-1} (n-i) = k \left[\sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \right] \\
&= k \left[n(n-1) - \frac{n(n-1)}{2} \right] \\
&= k \left[\frac{n(n-1)}{2} \right] \in \Theta(kn^2)
\end{aligned}
$$

- Let us discuss more sophisticated and more efficient algorithm for these problem, which is based on the divide-and-conquer technique.

Closest-Pair Problem (cont...)

Closest-Pair by divide-and-conquer

- For the sake of simplicity, we assume that the points are distinct. We also assume that the points are ordered in nondecreasing order of their x coordinate (P list) and y coordinate (Q list).
- we can divide the points into two subsets P_l and P_r of $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$ points, respectively, by drawing a vertical line through the median m of their x coordinates so that $\lceil \frac{n}{2} \rceil$ points lie to the left of or on the line itself, and $\lfloor \frac{n}{2} \rfloor$ points lie to the right of or on the line.
- Let d_l and d_r be the smallest distances between pairs of points in P_l and P_r , respectively, and let $d = min(d_l, d_r)$.
- Geometrically, Let $p(x, y)$ be a point on this list. For a point $p'(x', y')$ to have a chance to be closer to p than d_{min}, p' must belong to the rectangle shown in Figure b.



- Algorithm

Closest-Pair Problem (cont...)

1: **procedure** CPDQ($P[], Q[]$)
2: **if** (if n $\leq$ 3) **then** return the minimal distance
3: **else**
4: copy the first $\lceil \frac{n}{2} \rceil$ points of P to array P_l
5: copy the first $\lceil \frac{n}{2} \rceil$ points of Q to array Q_l
6: copy the first $\lfloor \frac{n}{2} \rfloor$ points of P to array P_r
7: copy the first $\lfloor \frac{n}{2} \rfloor$ points of Q to array Q_r
8: $d_l = CPDQ(P_l, Q_l)$
9: $d_r = CPDQ(P_r, Q_r)$
10: $d_{min} = d\{d_l, d_r\}$
11: $m = P[\frac{n}{1} - 1].x$
12: copy all the points of Q for which $|x - m| < d$ into array $S[0..num-1]$
13: $dminsq = d^2$
14: **for** (i=0 to num−2) **do**
15: k=i+1
16: **while** ($k \leq num-1$ *and* $(S[k].y - S[i].y)^2 < dminsq$) **do**
17: $dminsq = min((S[k].x - S[i].x)^2 + (S[k].y - S[i].y)^2, dminsq)$
18: $k = k + 1$
19: **end while**
20: **end for**
21: **end if**
22: **return** $sqrt(dminsq)$
23: **end procedure**

Closest-Pair Problem (cont...)

- Complexity

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 2T(\frac{n}{2}) + \Theta(n) & n > 2 \end{cases}$$

where $\Theta(n)$ is the complexity of Strip processing (merge part)

- Using Master Theorem $a = 2, b = 2, k = 1 \Rightarrow a = b^k$

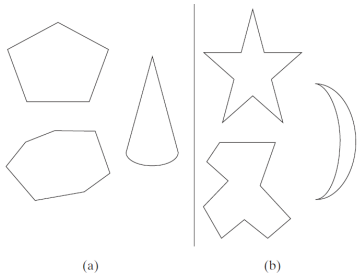
$$T(n) \in \Theta(nlogn)$$

- The necessity to presort input points does not change the overall efficiency class if sorting is done by a $O(nlogn)$ algorithm such as mergesort or quick sort.

- In fact, this is the best efficiency class one can achieve, because it has been proved that any algorithm for this problem must be in $\Omega(nlogn)$

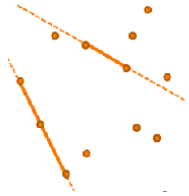
Convex-Hull Problem

- Convex: A set of points (finite or infinite) in the plane is called convex if for any two points p and q in the set, the entire line segment with the endpoints at p and q belongs to the set. (a) Convex sets. (b) not convex.
- The convex hull of a set S of points is the smallest convex set containing S. (The "smallest" requirement means that the convex hull of S must be a subset of any convex set containing S.)
- The convex-hull problem is the problem of constructing the convex hull for a given set S of n points.
- Example: The convex hull for the set of eight points is the convex polygon with vertices at p_1, p_5, p_6, p_7, p_3.



Convex-Hull (cont...)

- An extreme point of a convex set is a point of this set that is not a middle point of any line segment with endpoints in the set. For example, the extreme points of a triangle are its three vertices, the extreme points of a circle are all the points of its circumference



- The straight line through two points $(x_1, y_1), (x_2, y_2)$ in the coordinate plane can be defined by the equation

$$ax + bx - c = 0, \text{ where } a = y_2 - y_1, \quad b = x_2 - x_1, \quad c = x_1 y_2 - y_1 x_2$$

- if $q_1(x_1, y_1)$, $q_2(x_2, y_2)$, and $q_3(x_3, y_3)$ are three arbitrary points in the Cartesian plane, then the area of the triangle $\triangle q_1 q_2 q_3$ is equal to one-half of the magnitude of the determinant

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1 y_2 + x_3 y_1 + x_2 y_3 - x_3 y_2 - x_2 y_1 - x_1 y_3$$

- The sign of this expression is positive if and only if the point $q_3 = (x_3, y_3)$ is to the left of the line $\vec{q_1 q_2}$.

Convex-Hull (cont...)

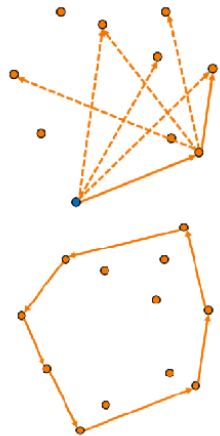
A brute-force algorithm for the convex hull problem

- Input: A list P of n $(n\ 2)$ points $p_1(x_1, y_1), ..., p_n(x_n, y_n)$

- Output: Set of points (Convex Hull)

```
1:  procedure CH(P[])
2:      for each point p_i do
3:          for each point p_j ≠ p_i do
4:              for each point p_k ≠ p_i ≠ p_j do
5:                  if P_k is not (left or on) (p_i, p_j) then
6:                      (p_i, p_j) is not extreme
7:                  else
8:                      (p_i, p_j) include in Convex Hull
9:                  end if
10:             end for
11:         end for
12:     end for
13: end procedure
```

- Complexity $\Theta(n^3)$

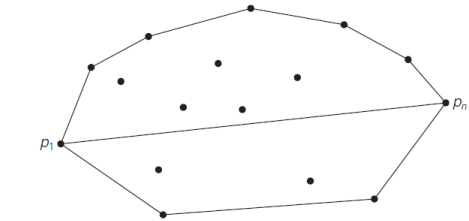


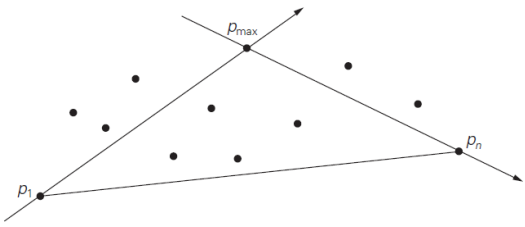
- Better algorithms: Gift Wrapping, Quick Hull, Graham's Algorithm

Convex-Hull (cont...)

A Divide and Conquer based algorithm for the convex hull problem

- Algorithm Quick Hull
  1. Sort the set of points S by x-coordinates.
  2. the leftmost point p_1 and the rightmost point p_n are two distinct extreme points of the set's convex hull
  3. Divide the entire set S into two sets $S1$ and S_2 related to upper and lower hulls
  4. The algorithm identifies point p_{max} in $S1$, which is the farthest from the line $\overrightarrow{p_1 p_n}$
  5. Then the algorithm identifies all the points of set S_1 that are to the left of the line $\overrightarrow{p_1 p_{max}}$; these are the points that will make up the set $S_{1,1}$.
  6. Similarly, the algorithm identifies all the points of set S_1 that are to the left of the line $\overrightarrow{p_n p_{max}}$; these are the points that will make up the set $S_{1,2}$.
  7. the algorithm can continue constructing the upper hulls of $p_1 \cup S_{1,1} \cup p_{max}$ and $p_{max} \cup S_{1,2} \cup p_n$ recursively and then simply concatenate them to get the upper hull of the entire set $p_1 \cup S_1 \cup p_n$.
- Complexity $T(n) \in \Theta(nlogn)$

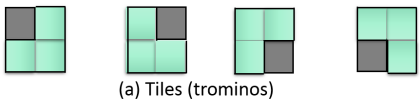




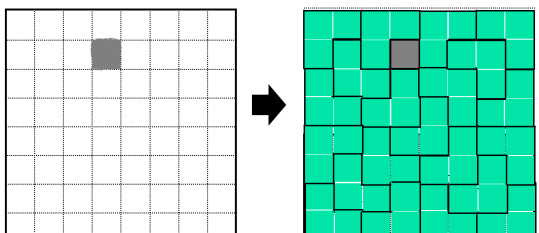
Tromino Puzzle

Problem definition

- A **tromino** is a group of three unit squares arranged in **L-shaped** (as shown in the Figure (a)).
- Consider the following tiling Problem:
  The input is an $2^n \times 2^n$ array of unit squares where n is the a positive integer of power 2, with one forbidden square on the array Figure (b).
- The output is a tiling of the array as shown in Figure (c). that satisfy the following conditions:
  1. Every unit square other than the input square is covered by a tromino.
  2. No tromino covers the input square.
  3. No two tromino overlap.
  4. No tromino extends beyond the board



(a) Tiles (trominos)

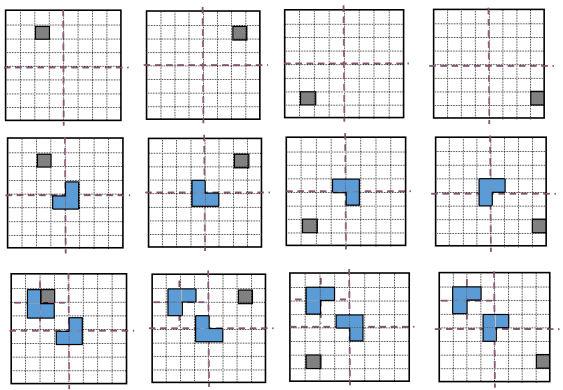


(b) Board with nxn squares (c) Tiling of the board

Tromino Puzzle (cont...)

Divide and conquer approach for Tromino Puzzle

- To solve the problem of Tromino puzzle, divide and conquer approach can be applied
- Suppose a board with a missing square of size $n \times n$.
- First, the board of size $2^n \times 2^n$is divided into four disjoint subboards of sizes $2^{n-1} \times 2^{n-1}$
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.
- Continue to reduce the size of the original problem, so that we eventually get to the 2×2 boards which we know how to solve (Base case)
- The subboard containing the missing square is recursively tiled first



Tromino Puzzle (cont...)

- **Algorithm: Tromino Puzzle(Tiling problem) (Pseudo-code)**

```
1: procedure TILING(integer n , integer x_location, integer y_location)
2:     if (n == 2) then                                          ▷ Base Case
3:         place one tromino
4:         return
5:     else                                                      ▷ Recursive Cases
6:         put the first tile at center with appropriate facing
7:         tiling(n/2, x1, y1)                                   ▷ tile the first quadrant
8:         tiling(n/2, x2, y2)                                   ▷ tile the second quadrant
9:         tiling(n/2, x3, y3)                                   ▷ tile the third quadrant
10:        tiling(n/2, x4, y4)                                   ▷ tile the forth quadrant
11:    end if
12: end procedure
```

- **Complexity**

$$T(n) = \begin{cases} T(2) = 1, & n = 2 \\ 4T\left(\frac{n}{2}\right) + 1, & n > 2 \end{cases}$$

$$a = 4,\ b = 2,\ k = 0,\ b^k = 1 \quad a > b^k \quad (case - 3)$$

$$T(n) \in \Theta\left(n^{\lg_b(a)}\right) = \Theta\left(n^{\lg_2(4)}\right) = \Theta\left(n^2\right)$$

- Master Theorem

$$\begin{cases} T(n) \in \Theta\left(n^k\right), & if\ \ a < b^k \quad (case - 1) \\ T(n) \in \Theta\left(n^k \lg n\right), & if\ \ a = b^k \quad (case - 2) \\ T(n) \in \Theta\left(n^{\log_b a}\right), & if\ \ a > b^k \quad (case - 2) \end{cases}$$

Summary

1. Multiplication of Large Integers
2. Matrix multiplication
3. Closest-Pair Problem
4. Convex-Hull Problem
5. Tromino Puzzle