Algorithm Design Strategy

"Divide and Conquer"

More examples of Divide and Conquer

- Review
  - Divide & Conquer Concept
  - Binary search
  - Merge sort

- More examples
  - Quicksort
  - Tromino tiling
  - Finding closest pair of points
  - Matrix Multiplication Algorithm

Divide and Conquer Approach

- Concept: D&C is a general strategy for algorithm design

It involves three steps:

(1) Divide an instance of a problem into one or more smaller instances

(2) Conquer (solve) each of the smaller instances. Unless a smaller instance is sufficiently small, use recursion to do this

(3) If necessary, combine the solutions to the smaller instances to obtain the solution to the original instances (e.g., Merge sort)

- Approach:
  - Recursion (Top-down approach)

Quicksort

```
quicksort(L)
{
   if (length(L) < 2) return L
   else
   {
         pick some x in L        // x is the pivot element
         L1 = { y in L : y < x }
         L2 = { y in L : y > x }
         L3 = { y in L : y = x }
         quicksort(L1)
         quicksort(L2)
         return concatenation of L1, L3, and L2
   }
}
```

Source: http://www.ics.uci.edu/~eppstein/161/960118.html

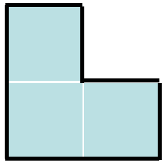
Time Complexity of Quicksort

- $T(n) = T(n_1) + T(n_2) + n\text{-}1$
  - n_1 = length of L_1, n_2 = length of L_2
  - $n_1 + n_2 = n - 1$ if all the elements in L are unique
- Best case
  - L is divided into two halves at every recursion
  - $T(n) = 2T(n/2) + n\text{-}1 = \Theta(n\lg n)$
- Average case
  - $\Theta(n\lg n)$
  - See the textbook for a proof
- Worst case
  - $T(n) = T(0) + T(n\text{-}1) + n\text{-}1$ when L is already sorted (As a result, the pivot is always the smallest)
  - So, $T(n) = \Theta(n^2)$
  - Why is it called quicksort then?

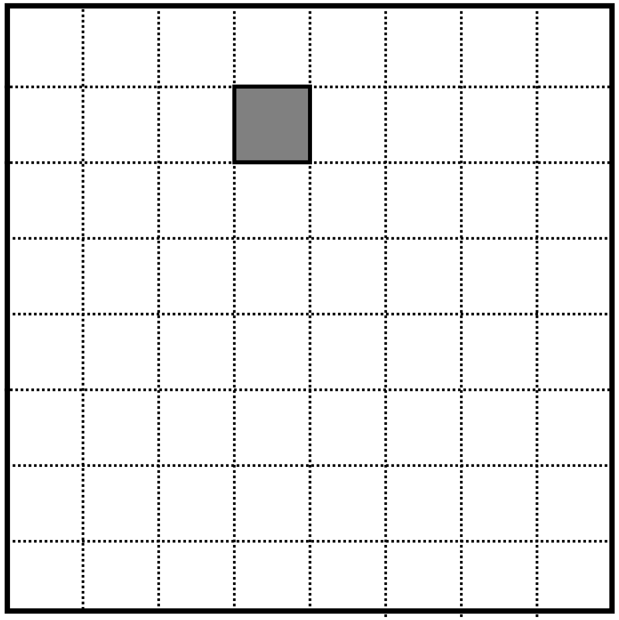
How to make Quicksort efficient

- Randomization
  - Randomly pick an element in L as the pivot
  - So, each item has the same probability of 1/n to be selected as the pivot
  - Avoid the worst case (what is the worst case in quicksort?)
  - O(nlgn) in the worst case
    - c.f., See blackboard for a proof (optional – not required)

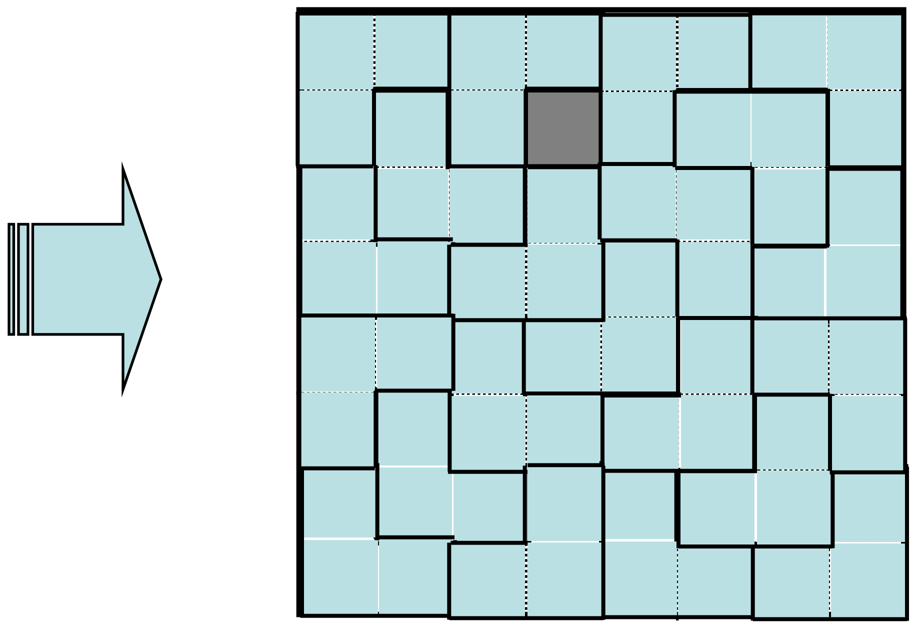
Tromino Tiling

A tromino tile: 

And a nxn (2^kx2^k) board with a hole:



A tiling of the board with trominos:

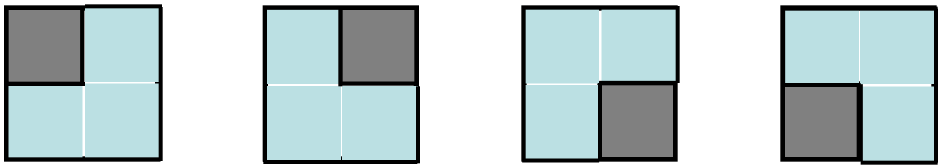


Solvable?

- Total number of squares:
  - $n * n - 1 = 2^k * 2^k - 1 = 2^{2k} - 1 = 4^k - 1$
- Size of one tromino: 3
- So, $4^k - 1$ should be divisible by 3
- Proof by induction
  - Basis: If $n = 2$, 4^1-1 is divisible by 3
  - Induction hypothesis: Assume $4^{k-1} - 1$ is divisible by 3
  - Induction step: $4^k - 1 = 4(4^{k-1} - 1) + 3$

Tiling: Trivial Case (n = 2)

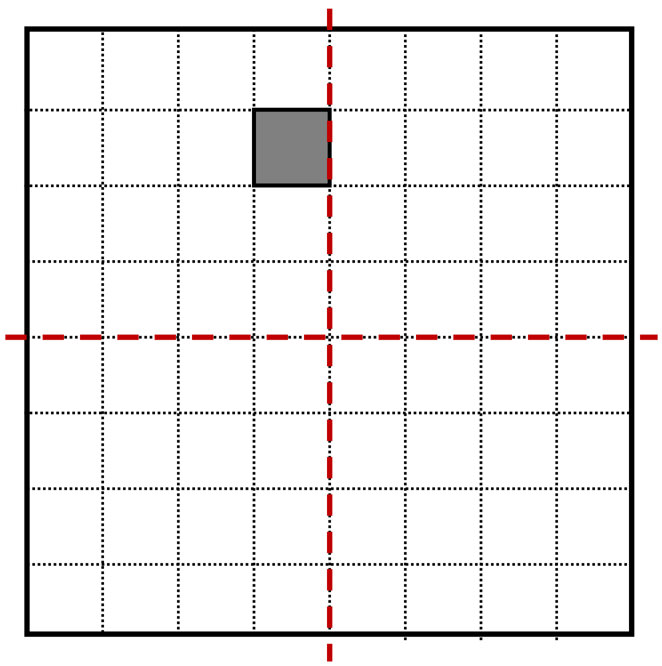
- Trivial case (n = 2): tiling a 2x2 board with a hole:



- Idea – try somehow to reduce the size of the original problem, so that we eventually get to the 2x2 boards which we know how to solve…

Tiling: Dividing the Problem

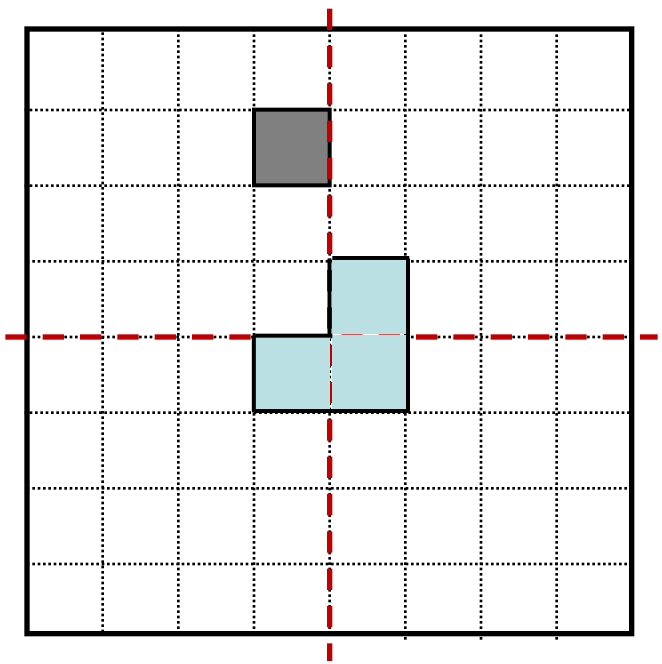
- To get smaller square boards, let's divide the original board into four boards



- Great! We have one problem of the size (n/2)x(n/2)
- But: The other three problems are not similar to the original problems – they do not have holes!

Tiling: Dividing the Problem

- Idea: insert one tromino at the center to get three *imaginary* holes in each of the three smaller boards



- Now we have four boards with holes of the size (n/2)x(n/2)
- Keep doing this division, until we get the 2x2 boards with holes – we know how to tile those

Tiling: Algorithm

INPUT: *n* – the board size (2^kx2^k board where n =2^k), *L* – location of the hole.
OUTPUT: tiling of the board

```
Tile(k, L)
   if k = 1 then
      Trivial case
      Tile with one tromino
      return
   Divide the board into four equal-sized boards
   Place one tromino at the center to cut out 3 additional
    holes (orientation based on where existing hole, L, is)
   Let L1, L2, L3, L4 denote the positions of the 4 holes
   Tile(k-1, L1)
   Tile(k-1, L2)
   Tile(k-1, L3)
   Tile(k-1, L4)
```

Tiling: Divide-and-Conquer

- Tiling is a divide-and-conquer algorithm:
  - Just do it trivially if the board is 2x2, else:
  - **Divide** the board into four smaller boards (introduce holes at the corners of the three smaller boards to make them look like original problems)
  - **Conquer** using the same algorithm recursively
  - **Combine** by placing a single tromino in the center to cover the three introduced holes

Time Complexity of Tromino Tiling

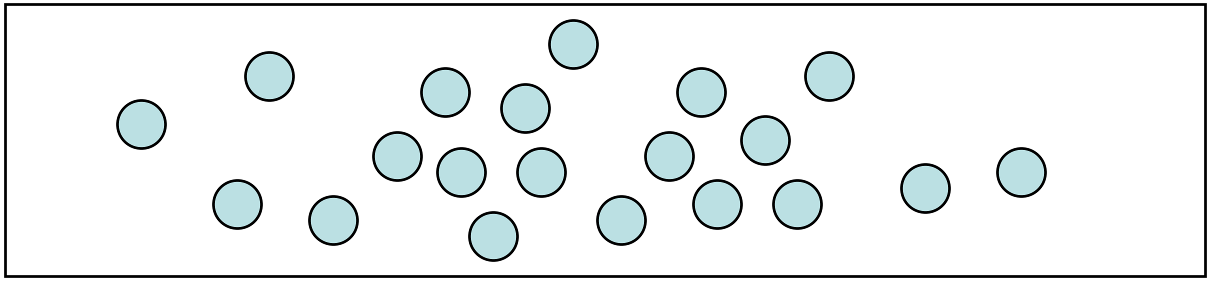
- $T(n) = 4T(n/2) + 1$
- $T(2) = 1$
- $O(n^2)$ by master theorem (or solve it iteratively)

Closest Pair Problem

- <u>Finding the closest pair of points</u>

  - Find the closest pair of points in a set of points. The set consists of points in two dimension plane

  - Given a set P of N points, find p, q $\in$ P, such that the distance d(p, q) is minimum



- Application:
  - Traffic control systems: A system for controlling air or sea traffic might need to know which two vehicles are too close in order to detect potential collisions.
  - Computational geometry

Brute force algorithm

Input: set S of points
Output: closest pair of points

```
min_distance = infinity
for each point x in S
    for each point y in S
        if x ≠ y and distance(x,y) < min_distance
        {
            min_distance = dist(x,y)
            closest_pair = (x,y)
        }
```

Time Complexity: $O(n^2)$

1 Dimension Closest Pair Problem

- Brute-force algorithm: Find all the distances D(p, q) and find the minimum distance
  - Time complexity: $O(n^2)$
- 1D problem can be solved in O(nlgn) via sorting
- But, sorting does not generalize to higher dimensions
- Let's develop a divide & conquer algorithm for 2D problem.

2D Closest Pair Problem

- <u>Finding the closest pair of points</u>

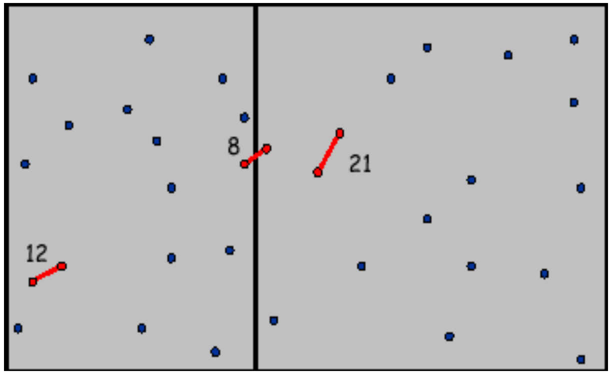
  - The "closest pair" refers to the pair of points in the set that has the smallest Euclidean distance,

  Distance between points p1=(x1,y1) and p2=(x2,y2)

$$D(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

  - If there are two identical points in the set, then the closest pair distance in the set will obviously be zero.

2D Closest Pair Problem



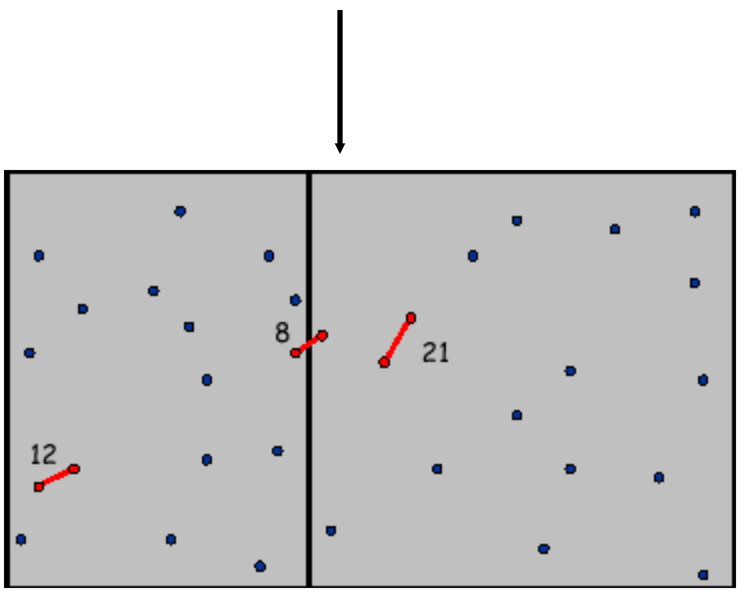
- Finding the closest pair of points

- Divide: Sort the points by x-coordinate; draw vertical line to have roughly n/2 points on each side
- Conquer: Find closest pair in each side recursively
- Combine: Find closest pair with one point in each side
- Return: best of three solutions

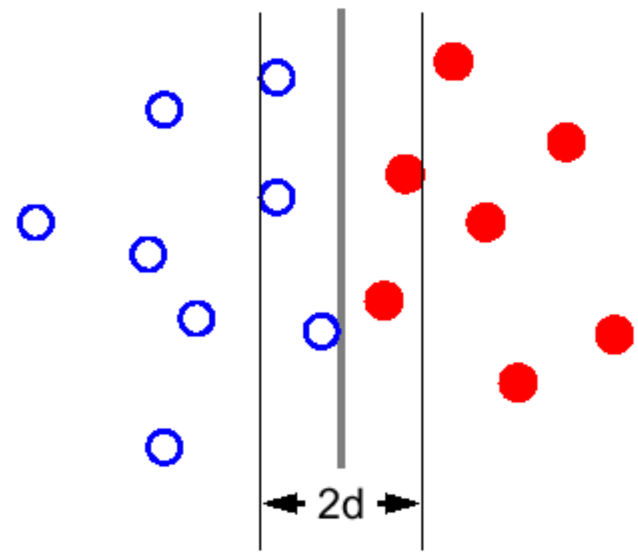
Key observation

Find the closest pair in a strip of width 2d

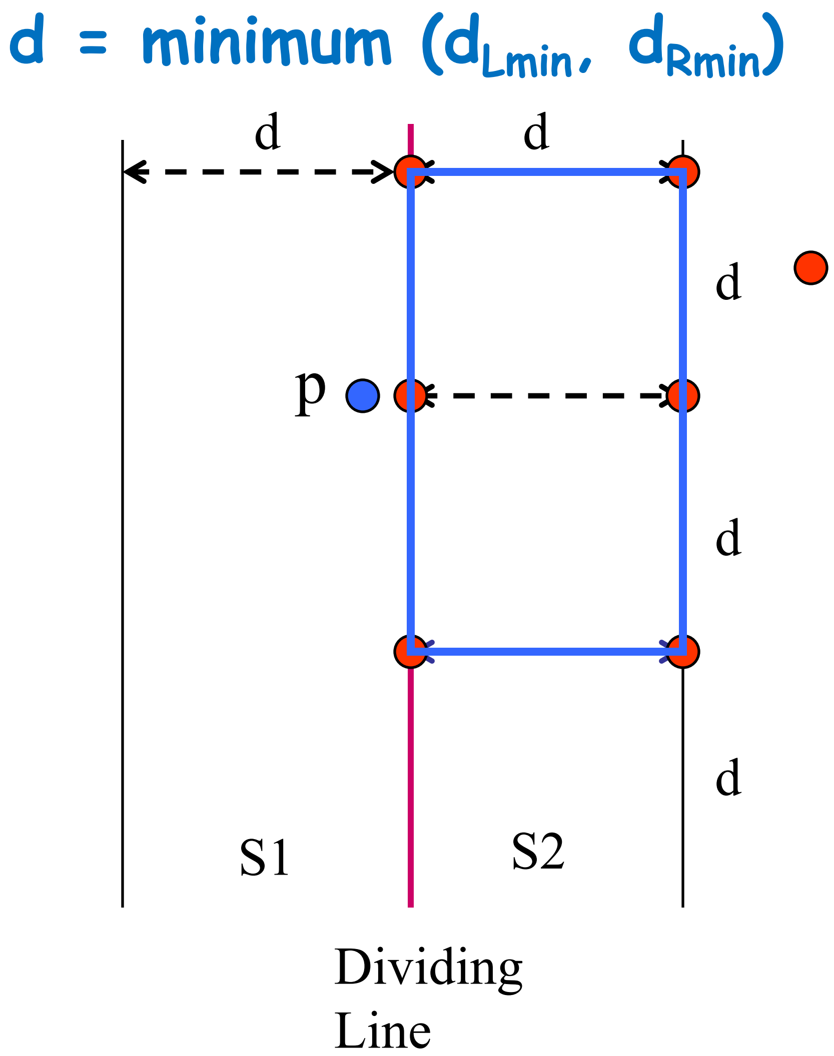
Example: d= min(12, 21)



- Find the closest (○, ●) pair in a strip of width 2d, knowing that no (○, ○) or (●, ●) pair is closer than d.



2D Closest Pair



For each point p on the left of the dividing line, we have to compare d to the distances between p and the points in the blue rectangle.

Note there must be no point inside the blue rectangle, because d = minimum (dLmin, dRmin). Thus, for each point p, we only have to consider 6 points – 6 red circles on the blue rectangles.

Also, note that there can be no red point between two red points on the blue box.

So, we need at most 6*n/2 distance comparisons

2D Closest Pair

- Time Complexity
  - $T(n) = 2T(n/2) + \theta(n) = \theta(n\lg n)$
  - Solve this recurrence equation yourself by applying the iterative method

Strassen's matrix multiplication algorithm

- Example: 2 by 2 matrix multiplication

m1 = (a11 + a22)(b11+b22)
m2 = (a21 + a22) b11
m3 = a11 (b12 – b22)
m4 = a22 (b21 – b11)
m5 = (a11 + a12) b22
m6 = (a21 – a11) (b11+b12)
m7 = (a12 – a22)(b21 + b22)

$$C = \begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix} = \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} \times \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix}$$

$$C = \begin{bmatrix} m1 + m4 - m5 + m7 & m3 + m5 \\ m2 + m4 & m1 + m3 - m2 + m6 \end{bmatrix}$$

Example 4 (cont'd)

- Strassen's matrix multiplication algorithm

  - partition n*n matrix into sub-matrices n/2 * n/2 (assume n is power of 2)
  - apply the seven basic calculations:

M1 = (A11 + A22)(B11+B22)
M2 = (A21 + A22) B11
M3 = A11 (B12 – B22)
M4 = A22 (B21 – B11)
M5 = (A11 + A12) B22
M6 = (A21 – A11) (B11+B12)
M7 = (A12 – A22)(B21 + B22)

$$C = \begin{bmatrix} C11 & C12 \\ C21 & C22 \end{bmatrix} = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} \times \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix}$$

$$C = \begin{bmatrix} M1 + M4 - M5 + M7 & M3 + M5 \\ M2 + M4 & M1 + M3 - M2 + M6 \end{bmatrix}$$

Strassen's matrix multiplication example

$$C=\begin{bmatrix} C11 & C12 \\ C21 & C22 \end{bmatrix}=\begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix}\times\begin{bmatrix} 5 & 5 & 6 & 6 \\ 5 & 5 & 6 & 6 \\ 7 & 7 & 8 & 8 \\ 7 & 7 & 8 & 8 \end{bmatrix}$$

$$M1=(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}+\begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix})\times(\begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}+\begin{bmatrix} 8 & 8 \\ 8 & 8 \end{bmatrix})$$

$$C=\begin{bmatrix} M1+M4-M5+M7 & M3+M5 \\ M2+M4 & M1+M3-M2+M6 \end{bmatrix}$$

Strassen's matrix multiplication algorithm

- Strassen's matrix multiplication:

```
input: n, A, B;
output: C

strassen_matrix_multiply(int n, A, B, C)
{
    divide A into A11, A12, A21, A22;
    divide B into B11, B12, B21, B22;

    strassen_matrix_multiply(n/2, A11+A22, B11+B22, M1);
    strassen_matrix_multiply(n/2, A21+A22, B11, M2);
     ......
     strassen_matrix_multiply(n/2, A12-A22, B21+B22, M7);

     Compose C11, ..., C22 by M1, ..., M7;

 }
```

Time Complexity

- Strassen's matrix multiplication algorithm

  - Every case Time complexity

    Multiplications: $T(n) = 7\ T(n/2);$
    $T(1) = 1$

    $T(n) = n^{lg7} \in \Theta(n^{2.81})$
    (while brute-force approach: $T(n) = \Theta(n^3)$)

    Additions/Subtractions: $T(n) = 7\ T(n/2) + 18\ (n/2)^2$
    $T(1) = 0$

    $T(n) = O(n^{2.81})$ by Master theorem ($a=7$, $b=2$, $k=2$)

Combine the D&C algorithm with other simple algorithm

- <u>Switching point</u>

- Recursive method may have no advantage for small n compared to an alternative, non-recursive, algorithm

- Recursive algorithm requires a fair amount of overhead

- Stop the dividing process at a certain switching point (or threshold) for recursive algorithm, and then use the alternative algorithm

- <u>Example</u>

  - Use the standard matrix multiplication to multiply two 2 *2 matrices

When not to use D&C algorithm

- Two cases:

- An instance of size n is divided into two or more instances each almost of size n

- An instance of size n is divided into almost n instances of size n/c (c is constant)

Fibonacci number: Divide & Conquer based on recursion is a poor choice!