Chapter 5
The Divide-and-Conquer Strategy

A simple example
- finding the maximum of a set S of n numbers

A general divide-and-conquer algorithm
Step 1: If the problem size is small, solve this problem directly; otherwise, split the original problem into 2 sub-problems with equal sizes.
Step 2: Recursively solve these 2 sub-problems by applying this algorithm.
Step 3: Merge the solutions of the 2 sub-problems into a solution of the original problem.
Time complexity of the general algorithm

- Time complexity:
  \[ T(n) = \begin{cases} 
  2T(n/2) + S(n) + M(n) & n \geq c \\
  b & n < c 
  \end{cases} \]

where \( S(n) \): time for splitting
\( M(n) \): time for merging
\( b \): a constant
\( c \): a constant
- e.g. Binary search
- e.g. quick sort
- e.g. merge sort

2-D maxima finding problem

- **Def:** A point \((x_1, y_1)\) dominates \((x_2, y_2)\) if \(x_1 > x_2\) and \(y_1 > y_2\). A point is called a maxima if no other point dominates it
- **Straightforward method:** Compare every pair of points.

Time complexity:
\( O(n^2) \)

Divide-and-conquer for maxima finding

The maximal points of \( S_L \) and \( S_R \)

The algorithm:
- **Input:** A set of \( n \) planar points.
- **Output:** The maximal points of \( S \).

Step 1: If \( S \) contains only one point, return it as the maxima. Otherwise, find a line \( L \) perpendicular to the X-axis which separates the set of points into two subsets \( S_L \) and \( S_R \), each of which consisting of \( n/2 \) points.

Step 2: Recursively find the maximal points of \( S_L \) and \( S_R \).

Step 3: Find the largest y-value of \( S_R \). Project the maximal points of \( S_L \) onto \( L \). Discard each of the maximal points of \( S_L \) if its y-value is less than the largest y-value of \( S_R \).
### The closest pair problem

- **Given a set S of n points, find a pair of points which are closest together.**

  - **1-D version:** Solved by sorting
  - **2-D version**
    - **Time complexity:** \( O(n \log n) \)

```
\begin{align*}
\text{Step 1: } & O(n) \\
\text{Step 2: } & 2T(n/2) \\
\text{Step 3: } & O(n)
\end{align*}
```

Assume \( n = 2^k \)
\[ T(n) = O(n \log n) \]

---

**at most 6 points in area A:**

---

The algorithm:
- **Input:** A set of \( n \) planar points.
- **Output:** The distance between two closest points.

**Step 1:** Sort points in \( S \) according to their y-values and x-values.

**Step 2:** If \( S \) contains only two points, return infinity as their distance.

**Step 3:** Find a median line \( L \) perpendicular to the X-axis to divide \( S \) into two subsets, with equal sizes, \( S_L \) and \( S_R \).

**Step 4:** Recursively apply Step 2 and Step 3 to solve the closest pair problems of \( S_L \) and \( S_R \). Let \( d_L(d_R) \) denote the distance between the closest pair in \( S_L \) (\( S_R \)). Let \( d = \min(d_L, d_R) \).
Step 5: For a point $P$ in the half-slab bounded by $L-d$ and $L$, let its $y$-value be denoted as $y_P$. For each such $P$, find all points in the half-slab bounded by $L$ and $L+d$ whose $y$-value fall within $y_P-d$ and $y_P+d$. If the distance $d'$ between $P$ and a point in the other half-slab is less than $d$, let $d=d'$. The final value of $d$ is the answer.

- Time complexity: $O(n \log n)$

Step 1: $O(n \log n)$

Steps 2–5:

$$T(n)=\begin{cases} 2T(n/2)+O(n)+O(n) & , n > 1 \\ 1 & , n = 1 \end{cases}$$

$\Rightarrow T(n) = O(n \log n)$

The convex hull problem

- The convex hull of a set of planar points is the smallest convex polygon containing all of the points.

- The divide-and-conquer strategy to solve the problem:

1. Select an interior point $p$.
2. There are 3 sequences of points which have increasing polar angles with respect to $p$.
   (1) $g, h, i, j, k$
   (2) $a, b, c, d$
   (3) $f, e$
3. Merge these 3 sequences into 1 sequence: $g, h, a, b, f, c, e, d, i, j, k$.
4. Apply Graham scan to examine the points one by one and eliminate the points which cause reflexive angles.

(See the example on the next page.)
Divide-and-conquer for convex hull

- Input: A set $S$ of planar points
- Output: A convex hull for $S$

Step 1: If $S$ contains no more than five points, use exhaustive searching to find the convex hull and return.

Step 2: Find a median line perpendicular to the X-axis which divides $S$ into $S_L$ and $S_R$; $S_L$ lies to the left of $S_R$.

Step 3: Recursively construct convex hulls for $S_L$ and $S_R$. Denote these convex hulls by Hull($S_L$) and Hull($S_R$) respectively.

- Step 4: Apply the merging procedure to merge Hull($S_L$) and Hull($S_R$) together to form a convex hull.

- Time complexity:
  $T(n) = 2T(n/2) + O(n) = O(n \log n)$

The Voronoi diagram problem

- e.g. The Voronoi diagram for three points

Each $L_i$ is the perpendicular bisector of the line.
Definition of Voronoi diagrams

- **Def**: Given two points $P_i, P_j \in S$, let $H(P_i, P_j)$ denote the half plane containing $P_i$. The Voronoi polygon associated with $P_i$ is defined as

$$V(i) = \bigcap_{i \neq j} H(P_i, P_j)$$

- Given a set of $n$ points, the Voronoi diagram consists of all the Voronoi polygons of these points.

- The vertices of the Voronoi diagram are called Voronoi points and its segments are called Voronoi edges.

Example for constructing Voronoi diagrams

- Divide the points into two parts.
Merging two Voronoi diagrams

- Merging along the piecewise linear hyperplane

The final Voronoi diagram

- After merging

Divide-and-conquer for Voronoi diagram

- Input: A set S of n planar points.
- Output: The Voronoi diagram of S.

Step 1: If S contains only one point, return.
Step 2: Find a median line L perpendicular to the X-axis which divides S into $S_L$ and $S_R$ such that $S_L$ ($S_R$) lies to the left(right) of L and the sizes of $S_L$ and $S_R$ are equal.

Step 3: Construct Voronoi diagrams of $S_L$ and $S_R$ recursively. Denote these Voronoi diagrams by $VD(S_L)$ and $VD(S_R)$.
Step 4: Construct a dividing piece-wise linear hyperplane HP which is the locus of points simultaneously closest to a point in $S_L$ and a point in $S_R$. Discard all segments of $VD(S_L)$ which lie to the right of HP and all segments of $VD(S_R)$ that lie to the left of HP. The resulting graph is the Voronoi diagram of S.

(See details on the next page.)
Mergeing Two Voronoi Diagrams into One Voronoi Diagram

- **Input:** (a) \( S_L \) and \( S_R \) where \( S_L \) and \( S_R \) are divided by a perpendicular line \( L \).
  
  (b) \( \text{VD}(S_L) \) and \( \text{VD}(S_R) \).

- **Output:** \( \text{VD}(S) \) where \( S = S_L \cap S_R \)

**Step 1:** Find the convex hulls of \( S_L \) and \( S_R \), denoted as \( \text{Hull}(S_L) \) and \( \text{Hull}(S_R) \), respectively. (A special algorithm for finding a convex hull in this case will by given later.)

**Step 2:** Find segments \( \overline{P_aP_b} \) and \( \overline{P_cP_d} \) which join \( \text{HULL}(S_L) \) and \( \text{HULL}(S_R) \) into a convex hull (\( P_a \) and \( P_c \) belong to \( S_L \) and \( P_b \) and \( P_d \) belong to \( S_R \) Assume that \( \overline{P_aP_c} \) lies above \( \overline{P_bP_d} \). Let \( x = a, y = b, \) \( SG = \overline{P_aP_b} \), and \( HP = \emptyset \).

**Step 3:** Find the perpendicular bisector of \( SG \).
Denote it by \( BS \). Let \( HP = HP \cup \{BS\} \). If \( SG = \overline{P_aP_b} \), go to Step 5; otherwise, go to Step 4.

**Step 4:** The ray from \( \text{VD}(S_L) \) and \( \text{VD}(S_R) \) which \( BS \) first intersects with must be a perpendicular bisector of either \( \overline{P_aP_b} \) or \( \overline{P_cP_d} \) for some \( z \). If this ray is the perpendicular bisector of \( \overline{P_aP_b} \), then let \( SG = \overline{P_aP_b} \); otherwise, let \( SG = \overline{P_cP_d} \). Go to Step 3.

**Step 5:** Discard the edges of \( \text{VD}(S_L) \) which extend to the right of \( HP \) and discard the edges of \( \text{VD}(S_R) \) which extend to the left of \( HP \). The resulting graph is the Voronoi diagram of \( S = S_L \cup S_R \).

Properties of Voronoi Diagrams

- **Def:** Given a point \( P \) and a set \( S \) of points, the distance between \( P \) and \( S \) is the distance between \( P \) and \( \Pi \), which is the nearest neighbor of \( P \) in \( S \).

- The HP obtained from the above algorithm is the locus of points which keep equal distances to \( S_L \) and \( S_R \).

- The HP is monotonic in \( y \).
Number of Voronoi edges

- Number of edges of a Voronoi diagram ≤ 3n - 6, where n is number of points.
- Reasoning:
  i. Number of edges of a planar graph with n vertices ≤ 3n - 6.
  ii. A Delaunay triangulation is a planar graph.
  iii. Edges in Delaunay triangulation ← → edges in Voronoi diagram.

Construct a convex hull from a Voronoi diagram

- After a Voronoi diagram is constructed, a convex hull can be found in $O(n)$ time.

Number of Voronoi vertices

- Number of Voronoi vertices ≤ 2n - 4.
- Reasoning:
  i. Let $F$, $E$ and $V$ denote number of faces, edges and vertices in a planar graph.
     Euler’s relation: $F = E - V + 2$.
  ii. In a Delaunay triangulation,
      $V = n, E ≤ 3n - 6$
      $\Rightarrow F = E - V + 2 ≤ 3n - 6 - n + 2 = 2n - 4$.

Construct Convex Hull from Voronoi diagram

Step 1: Find an infinite ray by examining all Voronoi edges.
Step 2: Let $P_i$ be the point to the left of the infinite ray. $P_i$ is a convex hull vertex.
   Examine the Voronoi polygon of $P_i$ to find the next infinite ray.
Step 3: Repeat Step 2 until we return to the starting ray.
Time complexity

- Time complexity for merging 2 Voronoi diagrams:
  - Total: $O(n)$
  - Step 1: $O(n)$
  - Step 2: $O(n)$
  - Step 3 - Step 5: $O(n)$
    (at most $3n - 6$ edges in $VD(S_L)$ and $VD(S_R)$ and
    at most $n$ segments in HP)
- Time complexity for constructing a Voronoi diagram: $O(n \log n)$
  because $T(n) = 2T(n/2) + O(n) = O(n \log n)$

Lower bound

- The lower bound of the Voronoi diagram problem is $\Omega(n \log n)$.
  sorting $\propto$ Voronoi diagram problem

Applications of Voronoi diagrams

- Euclidean nearest neighbor searching

Fast Fourier transform (FFT)

- Fourier transform
  $A(f) = \int_{-\infty}^{\infty} a(t) e^{2\pi if t} dt$

- Inverse Fourier transform
  $a(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} A(f) e^{-2\pi if t} df$

- Discrete Fourier transform (DFT)
  Given $a_0, a_1, \ldots, a_{n-1}$
  $A_j = \sum_{0 \leq k < n} a_k e^{2\pi i j k / n}, 0 \leq j \leq n-1$

- Inverse DFT
  $a_k = \frac{1}{n} \sum_{0 \leq j < n} A_j e^{2\pi i j k / n}, 0 \leq k \leq n-1$

The Voronoi diagram for a set of points on a straight line

The Fourier transform and inverse Fourier transform

Inverse DFT of the Fourier transform

Applications of Voronoi diagrams

Euclidean nearest neighbor searching

Fast Fourier transform (FFT)

The lower bound of the Voronoi diagram problem is $\Omega(n \log n)$.

Applications of Voronoi diagrams

Euclidean nearest neighbor searching

Fast Fourier transform (FFT)
DFT and waveform

- Any periodic waveform can be decomposed into the linear sum of sinusoid functions (sine or cosine).

\[ f(t) = \cos(2\pi(7)t) + 3\cos(2\pi(15)t) \]

See the left part:

An application of the FFT — polynomial multiplication

- Polynomial multiplication:
  \[ f(x) = \sum_{j=0}^{\frac{N}{2}-1} a_j x^j, \quad g(x) = \sum_{i=0}^{\frac{N}{2}-1} c_i x^i \quad h(x) = f(x) \cdot g(x) \]

- The straightforward product requires \( O(n^2) \) time.

- DFT notations:
  \[ f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \]
  Let \( b_j = f\left(w^j\right) \), \( 0 \leq j \leq n-1 \), \( w^n = 1 \)
  \( \{b_0, b_1, \ldots, b_{n-1}\} \) is the DFT of \( \{a_0, a_1, \ldots, a_{n-1}\} \).
  \[ h(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1} \]
  \( a_i = \frac{1}{n} h\left(w^{-i}\right) \), \( 0 \leq i \leq n-1 \)
  \( \{a_0, a_1, \ldots, a_{n-1}\} \) is the inverse DFT of \( \{b_0, b_1, \ldots, b_{n-1}\} \).

Fast polynomial multiplication

1. Let \( N \) be the smallest integer that \( N=2^q \) and \( N \geq 2n-1 \).
2. Compute FFT of \( \{a_0, a_1, \ldots, a_N\} \).
3. Compute FFT of \( \{c_0, c_1, \ldots, c_N\} \).
4. Compute \( f(w^j) \cdot g(w^j) \), \( 0 \leq j \leq N-1 \), \( w = e^{2\pi i/N} \)
5. Let \( h(w^j) = f(w^j) \cdot g(w^j) \)
   - Compute inverse DFT of \( \{h(w^0), h(w^1), \ldots, h(w^{n-1})\} \).
   - The resulting sequence of numbers are the coefficients of \( h(x) \).

- Time complexity: \( O(N\log N) = O(n\log n) \), \( N < 4n \).

FFT algorithm

- A straightforward method to calculate DFT requires \( O(n^2) \) time.
- DFT can be solved by the divide-and-conquer strategy (FFT).
  - Let \( w = e^{2\pi i/n} \), i.e. \( w^n = 1 \), \( w^2 = -1 \).
  \[ e^{i\theta} = \cos \theta + i \sin \theta \]
  \[ A_j = \sum_{0 \leq k < n} a_k e^{2\pi i j k/n} : 0 \leq j \leq n-1 \]
  \[ = \sum_{0 \leq k < n} a_k w^{jk} \]
  - e.g. \( n = 8 \), let \( e^{2\pi i/8} = w = e^{\pi i/4} \)
FFT algorithm when \( n=4 \)

- \( n =4, \) let \( e^{2\pi i/4} = w = e^{\pi i/2} \) (\( w^4 = 1, \ w^2 = -1 \))
- \( A_0 = a_0 + a_1 + a_2 + a_3 \)
- \( A_1 = a_0 + a_1w + a_2w^2 + a_3w^3 \)
- \( A_2 = a_0 + a_1w^2 + a_2w^4 + a_3w^6 \)
  \[= a_0 + a_1w^2 + a_2w^4 + a_3w^6\]
- \( A_3 = a_0 + a_1w^3 + a_2w^6 + a_3w^9 \)
  \[= a_0 + a_1w^3 + a_2w^6 + a_3w^9\]

- Rewrite as:
  \( A_0 = a_0 + a_2 + (a_1 + a_3) \)
  \( A_2 = a_0 + a_2w^4 + (a_1w^2 + a_3w^6) \)
  \[= a_0 + a_2 - (a_1 + a_3)\]
- When we calculate \( A_0 \), we shall calculate \((a_0 + a_2)\) and \((a_1 + a_3)\). Later, \( A_2 \) can be easily found.
- Similarly,
  \( A_1 = (a_0 + a_2w^2) + a_1w + a_3w^3 \)
  \( A_3 = (a_0 + a_2w^2) + (a_1w^3 + a_3w^9) \)
  \[= a_0 + a_2w^2 - (a_1w + a_3w^3)\]

FFT algorithm when \( n=8 \)

- \( n = 8, \) let \( e^{2\pi i/8} = w = e^{\pi i/4} \) (\( w^8 = 1, \ w^4 = -1 \))
- \( A_0 = a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 \)
- \( A_4 = a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 \)
- \( A_1 = a_0 + a_1w + a_2w^2 + a_3w^3 + a_4w^4 + a_5w^5 + a_6w^6 + a_7w^7 \)
- \( A_3 = a_0 + a_1w^3 + a_2w^6 + a_3w^9 \)
- \( A_5 = a_0 + a_1w^5 + a_2w^{10} + a_3w^{13} + a_4w^{17} + a_5w^{21} + a_6w^{25} + a_7w^{29} \)
- \( A_7 = a_0 + a_1w^7 + a_2w^{14} + a_3w^{21} + a_4w^{28} + a_5w^{35} + a_6w^{42} + a_7w^{49} \)

- After reordering, we have
  \( A_0 = (a_0 + a_2 + a_4 + a_6) + (a_1 + a_3 + a_5 + a_7) \)
  \( A_4 = (a_0 + a_2 + a_4 + a_6) - (a_1 + a_3 + a_5 + a_7) \)
  \( A_1 = (a_0 + a_2w^2 + a_4w^4 + a_6w^6) + w(a_1 + a_3w^2 + a_5w^4 + a_7w^6) \)
  \( A_3 = (a_0 + a_2w^2 + a_4w^4 + a_6w^6) - w(a_1 + a_3w^2 + a_5w^4 + a_7w^6) \)
  \( A_5 = (a_0 + a_2w^4 + a_4w^8 + a_6w^{12}) + w^2(a_1 + a_3w^4 + a_5w^8 + a_7w^{12}) \)
  \( A_7 = (a_0 + a_2w^4 + a_4w^8 + a_6w^{12}) - w^2(a_1 + a_3w^4 + a_5w^8 + a_7w^{12}) \)
  \( A_2 = (a_0 + a_2w^4 + a_4w^8 + a_6w^{12}) + w^3(a_1 + a_3w^4 + a_5w^8 + a_7w^{12}) \)
  \( A_6 = (a_0 + a_2w^4 + a_4w^8 + a_6w^{12}) - w^3(a_1 + a_3w^4 + a_5w^8 + a_7w^{12}) \)

- Rewrite as:
  \( A_0 = B_0 + C_0 \)
  \( A_4 = B_0 - C_0 \)
  \( A_1 = B_1 + C_1 \)
  \( A_5 = B_1 - C_1 \)
  \( A_2 = B_2 + C_2 \)
  \( A_6 = B_2 - C_2 \)
  \( A_3 = B_3 + C_3 \)
  \( A_7 = B_3 - C_3 \)
\[ x = w^2 = e^{2\pi i/4} = e^{\pi i/2} \quad (x^4 = 1, \ x^2 = -1) \]

We can recursively apply the same method to calculate \( B_j \)'s and \( C_j \)'s.

\[
\begin{align*}
B_0 &= a_0 + a_2 + a_4 + a_6 \\
B_1 &= a_0 + a_2 w^2 + a_4 w^4 + a_6 w^6 \\
&= a_0 + a_2 x + a_4 x^2 + a_6 x^3 \\
B_2 &= a_0 + a_2 w^4 + a_4 w^8 + a_6 w^{12} \\
&= a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6 \\
B_3 &= a_0 + a_2 w^6 + a_4 w^{12} + a_6 w^{18} \\
&= a_0 + a_2 x^3 + a_4 x^6 + a_6 x^9
\end{align*}
\]

Thus, \( \{B_0, B_1, B_2, B_3\} \) is the DFT of \( \{a_0, a_1, a_2, a_3\} \).

**General FFT**

- In general, let \( w = e^{2\pi i/n} \) (assume \( n \) is even)
  \( (w^n = 1, \ w^{n/2} = -1) \)

\[
\begin{align*}
A_j &= a_0 + a_1 w^j + a_2 w^{2j} + \ldots + a_{n-1} w^{(n-1)j} \\
&= \{a_0 + a_2 w^2 + a_4 w^4 + \ldots + a_{n-2} w^{(n-2)j}\} + \{a_1 w + a_3 w^3 + \ldots + a_{n-1} w^{(n-1)j}\} \\
&= B_j + C_j \\
A_j + n/2 &= a_0 + a_1 w^{j + n/2} + a_2 w^{2j + n} + a_3 w^{3j + 3n/2} + \ldots + a_{n-1} w^{(n-1)j + (n(n-1)/2)} \\
&= a_0 + a_2 w^{2j} + a_4 w^{3j} + \ldots + a_{n-2} w^{(n-2)j} + a_{n-1} w^{(n-1)j} \\
&= B_j - C_j
\end{align*}
\]

**Divide-and-conquer (FFT)**

- Input: \( a_0, a_1, \ldots, a_{n-1}, n = 2^k \)
- Output: \( A_j, j=0, 1, 2, \ldots, n-1, \)
  where \( A_j = \sum_{0 \leq k < n} a_k e^{2\pi ijk/n} \)

**Step 1:** If \( n=2 \), compute

\[
\begin{align*}
A_0 &= a_0 + a_1, \\
A_1 &= a_0 - a_1, \\
\text{and return.}
\end{align*}
\]

**Step 2:** Divide each \( a_j, 0 \leq j \leq n/2 - 1 \) into two sequences: \( O_j \) and \( E_j \), where \( O_j(E_j) \) consists of odd-numbered (even-numbered) terms of \( A_j \).

**Step 3:** Recursively calculate the sums of terms in \( O_j \) and \( E_j \). Denote the sum of terms of \( O_j \) and \( E_j \) by \( B_j \) and \( C_j \), respectively.

**Step 4:** Compute \( A_j \) by the following formula:

\[
\begin{align*}
A_j &= B_j + C_j \text{ for } 0 \leq j \leq n/2 - 1 \\
A_{j + n/2} &= B_j - C_j \text{ for } 0 \leq j \leq n/2 - 1.
\end{align*}
\]

- Time complexity:
  \[ T(n) = 2T(n/2) + O(n) = O(n \log n) \]
Matrix multiplication

- Let A, B and C be $n \times n$ matrices
  
  \[ C = AB \]
  
  \[ C(i, j) = \sum_{k=1}^{n} A(i, k)B(k, j) \]

- The straightforward method to perform a matrix multiplication requires $O(n^3)$ time.

Divide-and-conquer approach

- \[ C = AB \]
  
  \[
  \begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
  \end{bmatrix} =
  \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
  \end{bmatrix}
  \begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
  \end{bmatrix}
  \]

  \[
  C_{11} = A_{11}B_{11} + A_{12}B_{21} \\
  C_{12} = A_{11}B_{12} + A_{12}B_{22} \\
  C_{21} = A_{21}B_{11} + A_{22}B_{21} \\
  C_{22} = A_{21}B_{12} + A_{22}B_{22}
  \]

- Time complexity:
  
  \[ T(n) = \begin{cases} 
  b & n \leq 2 \\
  8T(n/2)+cn^2 & n > 2
  \end{cases} \]

  We get $T(n) = O(n^3)$

---

Strassen’s matrix multiplication

- \[ P = (A_{11} + A_{22})(B_{11} + B_{22}) \]
- \[ Q = (A_{21} + A_{22})B_{11} \]
- \[ R = A_{11}(B_{12} - B_{22}) \]
- \[ S = A_{22}(B_{21} - B_{11}) \]
- \[ T = (A_{11} + A_{12})B_{22} \]
- \[ U = (A_{21} - A_{11})(B_{11} + B_{12}) \]
- \[ V = (A_{12} - A_{22})(B_{21} + B_{22}) \]

  \[
  C_{11} = P + S - T + V \\
  C_{12} = R + T \\
  C_{21} = Q + S \\
  C_{22} = P + R - Q + U
  \]

Time complexity

- 7 multiplications and 18 additions or subtractions

- Time complexity:
  
  \[ T(n) = \begin{cases} 
  b & n \leq 2 \\
  7T(n/2)+an^2 & n > 2
  \end{cases} \]

  \[ T(n) = an^2 + 7T(n/2) \]
  
  \[ = an^2 + 7(a(n/2)^2 + 7T(n/4)) \]
  
  \[ = an^2 + (7/4)an^2 + 7^2T(n/4) \]
  
  \[ = \ldots \]
  
  \[ = an^2(1 + 7/4 + (7/4)^2 + \ldots +(7/4)^k+7^kT(1)) \]
  
  \[ \leq cn^2(7/4)^{log_{24}7/4} + \log_{24}7^k \]
  
  \[ = cn^2 \log_{24}7 + \log_{24}7^k \]
  
  \[ = O(n^2 \log 27) \]
  
  \[ \cong O(n^2.81) \]