Question 1:

fill missing rows and columns with zero to the original matrix so that the
new matrix has n’ elements in each rows and columns, where n’ is a power of
2. And then apply Strassen’s Matrix multiplication algorithm. Since n’ < 2n,
The time complexity is

$$\Theta((2n)^{\log_27}) = \Theta(2^{\log_27 \cdot \log_27}) = \Theta(n^{\log_27})$$

Question 2:

Solution 1:
since 3n − 1 is not a power of 2, expand B(x) so that
$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{3n-1} x^{3n-1} + 0 x^{3n} + \ldots + 0 x^{4n-1}.$$  
1. Split B(x) into odd part and even part, so that

$$B_{even}(x) = b_0 + b_2 x + b_4 x^2 + \ldots + b_{4n-2} x^{4n-2}$$
$$B_{odd}(x) = b_1 + b_3 x + b_5 x^2 + \ldots + b_{4n-1} x^{4n-2}.$$  
and

$$B(x) = B_{even}(x) + x B_{odd}(x)$$

2. Do similar operation to A(x).

3. We should evaluate A(x) and B(x) at 4n different points so that we can solve c_0, c_1, ..., c_{4n-1}. then make w = cos(2 * pi/4n) + i * sin(2 * pi/4n), and evaluate A(x) and B(x) at 1, w, w^2, ..., w^{4n-1}, and calculate A(x)*B(x).

4. Solve the FFT

Solution 2:

$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{3n-1} x^{3n-1}$$

$$= B_1(x) + x^n B_2(x) + x^{2n} B_3(x)$$

where

$$B_1(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1}$$
$$B_2(x) = b_n + b_{n+1} x + \ldots + b_{2n-1} x^{n-1}$$
$$B_3(x) = b_{2n} + b_{2n+1} x + \ldots + b_{3n-1} x^{n-1}$$
Then \( A(x)B(x) = A(x)B_1(x) + x^n A(x)B_2(x) + x^{2n} A(x)B_3(x) \), since \( n \) is a power of 2, and \( B_1(x), B_2(x) \) and \( B_3(x) \) all have \( n \) elements, it is okay to apply FFT algorithm on \( A(x)B_1(x), A(x)B_2(x) \) and \( A(x)B_3(x) \) seperately. And then combine them to get the final answer.

**Question 3:**

Let \( V_0 = V \);

while there are \( v \) in \( V_0 \) to be examined:

- if \( V_0 - \{v\} \) is a vertex cover;
  \[ V_0 = V_0 - \{v\} \]

return \( V_0 \);

You have to do \( ||V|| \) iteration, and during each iteration, the algorithm need to test \( ||E|| \) edge to see whether it is a vertex cover, So the total time complexity is \( O(||V|| ||E||) \)

PS: all other reasonable answers will be accepted.

**Question 4:**

It can be proved using following example: Black edges and Red edges are two different matchings, and \( ||Black|| > ||Red|| \), but if you try to add any black edge into red edges, the outcome fail to maintain the property of matching. And thus it violate the exchange property for matroid.

PS: all other reasonable answers will be accepted.

**Question 5:**

Solution 1: Solve it by Divide-Conquer (It is not a dynamic programming example strictly).

\[
pow(a, n) \begin{align*}
  &\text{if } n == 0: \\
  &\quad \text{return } 1 \\
  &\text{if } n \text{ is odd:} \\
  &\quad \text{return } a * pow(a, (n - 1/2))^2 \\
  &\text{else:} \\
  &\quad \text{return } pow(a, n/2)^2;
\end{align*}
\]

Time Complexity is \( O(\log n) \)

Solution 2:
\textit{pow}(a, n)
\begin{verbatim}
result = 1;
for i from 1 to n:
    result = result * a:
\end{verbatim}

Time complexity is $O(n)$