1. Após encontrarmos a fórmula fechada para as recorrências T_C e T_D, temos:

$$
\begin{aligned}
T_A(n) &= n\sqrt{n} \\
T_B(n) &= n \log n \\
T_C(n) &= \sqrt{n} \log n \quad (\text{primeiro caso do Teorema Mestre simplificado}) \\
T_D(n) &= n^2 \qquad\qquad\qquad\quad (\text{primeiro caso do Teorema Mestre geral})
\end{aligned}
$$

A menor delas é T_C, pois nenhuma das outras três é $\Omega(T_C)$:

- T_A não é $\Omega(T_C)$. Assumindo que é verdade, teríamos

$$
\begin{aligned}
n\sqrt{n} &\leq c\sqrt{n} \log n \\
n &\leq c \log n \\
\frac{n}{\log n} &\leq c
\end{aligned}
$$

  O que é claramente falso, já que $\log n$ é sempre menor que n para para $n > 1$ e não podemos limitar por uma constante.
- T_B não é $\Omega(T_C)$. Assumindo que é verdade, teríamos:

$$
\begin{aligned}
n \log n &\leq c\sqrt{n} \log n \\
n &\leq c\sqrt{n} \\
\frac{n}{\sqrt{n}} &\leq c \\
\sqrt{n} &\leq c
\end{aligned}
$$

  O que é claramente falso, já que não podemos limitar por uma constante.
- T_D não é $\Omega(T_C)$. Assumindo que é verdade, teríamos:

$$
\begin{aligned}
n^2 &\leq c\sqrt{n} \log n \quad (\text{majorando } \log n \text{ por } \sqrt{n}) \\
n^2 &\leq cn \\
n &\leq c
\end{aligned}
$$

  O que é claramente falso, já que não podemos limitar por uma constante.
- Portanto, devemos escolher o algoritmo C, já que ele é assintoticamente o de menor complexidade

2. De acordo com o teorema mestre, já que $a > b^d (7 > 2^2)$ temos que:

$$T(A) = n^{\log_b a} = n^{\log_2 7}$$

Para o algoritmo A', temos que $b^d = 4^2 = 16$. De acordo com o teorema mestre:

- Para $a > 16$, a complexidade de $T(A') = n^{\log_b a} = n^{\log_4 a}$. Para que $T(A') < T(A)$, temos que

$$\begin{aligned}
n^{\log_4 a} &< n^{\log_2 7} \\
\log_4 a &< \log_2 7 \\
\frac{\log_2 a}{\log_2 4} &< \log_2 7 \qquad (\text{mudança de base}) \\
\frac{\log_2 a}{2} &< \log_2 7 \\
\log_2 a &< 2\log_2 7 \\
\log_2 a &< \log_2 7^2 \\
a &< 7^2
\end{aligned}$$

  Portanto, $a < 49$. O maior inteiro menor que 49 é o 48.

3. **InsertionSort:** A_1 é o melhor caso e A_2 é o pior caso. No melhor caso, o número de comporações é $\Omega(n)$, e no pior caso, $O(n^2)$, portanto estamos no caso ii.
   MergeSort: A complexidade do MergeSort é $\Theta(n \log n)$, portanto em ambos os casos o número de comparações é similar e estamos no caso iii.
   QuickSort: Em ambos os casos estamos no pior caso, pois particiona vai gerar um subvetor de tamanho zero e um subvetor de tamanho $n - 1$ a cada iteração. A complexidade do QuickSort no pior caso é $O(n^2)$, e estamos no caso iii.
   HeapSort A complexidade do HeapSort é $\Theta(n \log n)$, portanto em ambos os casos o número de comparações é similar e estamos no caso iii.
4. No início do algoritmo, temos que $m = a$ (que é um inteiro positivo pela pré-condição). Também é válido que $a = bx + m$ pois $x = 0$, e $a = m = a$.
   Seja x_i e m_i o valor de x e m no final do passo i, respectivamente. No final do passo $i + 1$, temos que $x_{i+1} = x_i + 1$ (linha 5) e $m_{i+1} = m_i - b$ (linha 6). Dessa maneira, assumindo que a invariante de laço é verdadeira, temos que:

$$\begin{aligned}
a_{i+1} &= b(x_{i+1}) + m_{i+1} \\
&= b(x_i + 1) + m_i - b \\
&= bx_i + b + m_i - b \\
&= bx_i + m_i = a_i
\end{aligned}$$

Além disso, temos que $m_i \geq b$ (senão não teriamos entrado no laço), portanto $m_{i+1} > 0$. Portanto a invariante de laço se mantém. Como m_i sempre decresce, o laço termina quando $m_i < b$, sendo $i = x$ o número de vezes que o laço é executado. Observe também que x é número de vezes que subtraímos b de a (portanto o resultado da divisão inteira de a por b), e $m = a - bx$ o resto da divisão inteira.

5. O melhor caso é quando o grafo é completamente desconexo e não tem nenhuma aresta, somente vértices. Nesse caso, a busca verifica um único nó, e como ele não tem nenhum vizinho, a busca termina. No caso da lista de adjacências, a lista de adjacência a partir daquele nó está vazia, portanto o algoritmo é $O(1)$. No caso da matriz de adajacência, precisaríamos verificar uma linha inteira da matriz para constantar que ele não tem adjacências, portanto $O(|V|)$.