Questão 1 - Determine a complexidade de tempo no pior caso do algoritmo Merge Sort ao realizar a ordenação de strings com tamanho máximo de `'s'` caracteres. Lembre-se que existe uma complexidade temporal adicional atrelada à comparação e atribuição envolvendo strings. Considere a complexidade do pior caso de cada um desses operadores.

Questão 2 - Resgate em seu material da disciplina de Teoria de Grafos (TEG) um algoritmo para encontrar um ciclo em um grafo direcionado (caso não tenha este algoritmo pesquise e implemente-o). Analise a complexidade de tempo e espaço para o melhor e pior caso deste algoritmo em relação ao número de vértices V, caso a implementação utilize uma matriz de adjacências, ou em relação ao número de arestas E e vértices V, caso a implementação utilize uma lista de adjacências.

Questão 3 - Considere uma aplicação na qual é necessário armazenar e consultar números inteiros grandes (BigInt) de k dígitos. Optou-se por utilizar um algoritmo de Hash com um vetor de h posições, usando endereçamento aberto com listas encadeadas para cada posição da estrutura. Utilizou-se uma função hash de complexidade de tempo $\Theta(k)$ e complexidade de espaço $\Theta(1)$. Responda:

a) Qual seria a complexidade de tempo e espaço, no melhor e pior caso, para realizar uma operação de inserção, sabendo que já foram incluídos h registros? Explique em quais circunstâncias ocorre o melhor e pior caso.

b) Qual seria a complexidade de tempo para realizar b operações de consulta, sabendo que já foram inseridos p registros? Explique em quais circunstâncias ocorre o melhor e pior caso.

OBS: pensar e descrever o melhor e pior caso nestes exercícios pode ajudar a elaborar a complexidade final do algoritmo.

Questão 4 - Responda as seguintes questões sobre inteiros grandes (Bigint):

a) Considerando a multiplicação entre inteiros grandes (n) de k bits (ambos), quais são o menor e o maior tamanho possível do valor resultante desta operação? Apresente o resultado em relação ao número de dígitos do valor resultante (k) e em relação ao seu valor numérico (n).

b) Considerando a exponenciação entre inteiros grandes de k bits – i.e. repetidas multiplicações – diga qual é o maior tamanho possível em relação ao número de dígitos (k) do valor resultante desta operação.

Questão 5 - Com relação a primalidade e ao algoritmo de RSA, responda as seguintes questões:

a) Seria computacionalmente viável utilizar o algoritmo do Crivo de Erastótenes para verificar se os valores p e q gerados aleatoriamente são primos? Calcule a complexidade de tempo de tal algoritmo a fim de justificar sua resposta.

b) Sabe-se que em um algoritmo simples que tenta fatorar uma chave pública de RSA, por ataque de força bruta, é necessário testar a primalidade de p e q, dois primos grandes. Analise o algoritmo apresentado a seguir, que recebe um valor n e tenta fatorá-lo em p e q. Diga qual a complexidade de tempo, no pior caso, para tal algoritmo, em termos do seu valor real (n) e em termos do seu número de bits (k). Considere as seguintes complexidades de tempo para as demais operações utilizadas no algoritmo:

`sqrt()`: O($k^2 \log k$) / `soma()`: O(k) / `ehDivisor()`: O(k^2) / `print()`: O(k)
operador <= (menor ou igual): O(k) / operador / (divisão): O(k^2)

c) Seria interessante usar este algoritmo fatorador apresentado no quadro abaixo para determinar a primalidade de $\boldsymbol{p}$ e $\boldsymbol{q}$, sabendo que usualmente estes números possuem 1024 bits? Justifique sua resposta considerando a complexidade de tempo em relação ao valor absoluto (n).

OBS: por comparação, lembre-se que um bubble sort se torna muito lento quando são ordenados mais do que 1.000.000 ($\approx 2^{20}$) elementos.

```
void fatorador( BigInt n ){
   BigInt aux = sqrt( n );
   for(BigInt p=2; p<=aux; p.soma(1)){
      if( n.ehDivisor(p) ){
         q = n/p;
         p.print();
         q.print();
    }
    return res;
 }
```

TEOREMA MESTRE:

$\mathbf{f(n)} = \boldsymbol{\Theta}\left(\boldsymbol{n}^{\log_b a}\right) \rightarrow \boldsymbol{T(n)} = \boldsymbol{\Theta}\left(\boldsymbol{n}^{\log_b a} \times \log \boldsymbol{n}\right)$

$\mathbf{f(n)} = \mathbf{O}\left(\boldsymbol{n}^{\log_b a - \varepsilon}\right) \rightarrow \boldsymbol{T(n)} = \boldsymbol{\Theta}\left(\boldsymbol{n}^{\log_b a}\right)$

$\mathbf{f(n)} = \boldsymbol{\Omega}\left(\boldsymbol{n}^{\log_b a + \varepsilon}\right) \rightarrow \boldsymbol{T(n)} = \boldsymbol{\Theta}\left(\boldsymbol{f(n)}\right)$

se $\boldsymbol{af(n/b)} \leq \boldsymbol{cf(n)}$ com c<1