Exercício 1) Analise a complexidade de tempo e espaço para o melhor e pior caso dos algoritmos apresentados no arquivo: `estruturas-dados.cpp`

Exercício 2) Determine a complexidade de tempo no pior caso do algoritmo Merge Sort ao realizar a ordenação de strings com tamanho máximo de 's' caracteres. (Dica: analise o algoritmo do Merge Sort repensando as complexidades dos operadores envolvendo strings: maior, menor, atribuição que neste caso são usados para string. Considere a complexidade do pior caso de cada um desses operadores).

Exercício 3) Considere uma aplicação na qual é necessário armazenar e consultar diversos nomes de pessoas de no máximo **s** caracteres. Optou-se por utilizar um algoritmo de Hash com um vetor de **n** posições, usando endereçamento aberto com listas encadeadas para cada posição da estrutura. Utilizou-se uma função hash de complexidade de tempo Θ(s) e complexidade de espaço Θ(1). Com base nestas informações 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 **p** registros? Explique em quais circunstâncias ocorre o melhor e pior caso. Considere **p < n**.

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. Novamente considere **p < n**.

Exercício 4) Se for preciso calcular o caminho mínimo entre dois vértices de um grafo esparso, com arestas que podem ter peso negativo, mas não apresentam um ciclo de valores negativos, qual seria o melhor algoritmo a se utilizar: Dijkstra, Bellman-Ford ou Floyd-Warshall? Justifique sua resposta com base nas complexidades destes algoritmos.

OBS: para a prova não é necessário decorar as complexidades. Neste caso você pode consultar o material, então não é necessário trazer a complexidade e detalhamento dos algoritmos.

Exercício 5) Analise a complexidade de tempo e espaço para o pior caso (dica: qual seria o divisor que conduz a mais repetições?) do algoritmo de divisão de números inteiros grandes. Apresente a complexidade de tempo em relação ao valor real de **a** (n) e também em relação ao número de bits de **a**. Por fim, diga qual a complexidade de espaço do algoritmo para o pior caso.

```
BigInt divisao(BigInt a, BigInt d){
    BigInt res = 0;
    while( a.ehMenor(d) ){
        res.soma(1);
        a.subtracao(d);
    }
    return res;
}
```

OBS: considere que as funções `ehMenor`, `soma` e `subtração` têm complexidade O(k), onde k é o número de dígitos do maior número envolvido na subtração.

Exercício 6) Analise o seguinte algoritmo de quebra da chave RSA por força bruta que recebe um valor **n** e tenta o fatorar em **p** e **q**, sendo que: $p \times q = n$ (p e $q \neq 1$). Diga qual a complexidade de tempo (pior caso) em termos do valor real em termos do seu número de bits (k), considerando as seguintes complexidades de tempo:

```
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;
 }
```

`sqrt()`: $O(k^3)$ / `soma()`: $O(k)$ / `ehDivisor()`: $O(k^2)$ / operador <=: $O(k)$
operador /: $O(k)$ / `print()`: $O(k)$
onde k é o número de bits do maior BigInt envolvido.