1. Uma fila F com N elementos é ordenada se os elementos quando removidos de F são $e_1, \ldots, e_N$ (nesta ordem) e tais que $e_i \leq e_{i+1}$, para todo $1 \leq i < N$. Isto é, uma fila é ordenada se quando seus elementos são todos removidos eles saem em ordem ascendente. Dada uma fila F, escreva um algoritmo que remova e reinsira elementos até que F se torne ordenada com espaço auxiliar constante acrescido de uma pilha auxiliar P (F e P devem ser manipuladas apenas através das funções enfileira/desenfileira/próximo/tamanho e empilha/desempilha/topo/tamanho, respectivamente).

1. função que remove e reinsere elementos em uma fila F até que ela se torne ordenada, utilizando uma pilha auxiliar P e com espaço auxiliar constante:

```
algoritmo OrdenarFila(F: Fila, P: Pilha)
  N ← tamanho(F) // número de elementos na fila F
  enquanto não estáOrdenada(F) faça
    maior ← -INF // assume um valor mínimo inicial para comparar
    para i de 1 até N faça
      elemento ← desenfileira(F) // remove o elemento da frente da fila F
      se elemento > maior então
        maior ← elemento // atualiza o maior elemento encontrado
      fim se
      empilha(P, elemento) // empilha o elemento na pilha P
    fim para
    enfileira(F, maior) // enfileira o maior elemento de volta na fila F
    enquanto tamanho(P) > 0 faça // desempilha e enfileira os elementos restantes na fila F
      elemento ← desempilha(P)
      enfileira(F, elemento)
    fim enquanto
  fim enquanto
fim algoritmo
```

A função **estaOrdenada(F)** é uma função auxiliar que verifica se a fila F está ordenada de acordo com a definição fornecida no enunciado, ou seja, se os elementos, quando removidos da fila, saem em ordem ascendente. A complexidade de tempo desse algoritmo depende do tamanho da fila F e do número de elementos N na fila. O pior caso ocorre quando a fila está totalmente desordenada, exigindo que todos os elementos sejam removidos e reinsiridos. Portanto, a complexidade de tempo é O(N^2), onde N é o número de elementos na fila.

PARA CASOS DO PROFESSOR PEDIR A FUNÇÃO **ESTAORDENADA(F)**

```
função estáOrdenada(F: Fila): Lógico
  anterior ← -INF // assume um valor mínimo inicial para comparar
  enquanto não estáVazia(F) faça
```

```
        elemento ← desenfileira(F) // remove o elemento da frente da fila F
        se elemento < anterior então
            retorna falso // se o elemento atual for menor que o anterior, a fila não está ordenada
        fim se
        anterior ← elemento // atualiza o valor anterior para o elemento atual
    fim enquanto
    retorna verdadeiro // se todos os elementos foram verificados e não houve violação de ordenação, a fila está ordenada
fim função
```

CÓDIGO COMPLETO:

```
função estáVazia(F: Fila): Lógico
    se tamanho(F) = 0 então // verifica se o tamanho da fila é igual a 0
        retorna verdadeiro // se for, a fila está vazia
    senão
        retorna falso // caso contrário, a fila não está vazia
    fim se
fim função

função estáOrdenada(F: Fila): Lógico
    anterior ← -INF // assume um valor mínimo inicial para comparar
    enquanto não estáVazia(F) faça
        elemento ← desenfileira(F) // remove o elemento da frente da fila F
        se elemento < anterior então
            retorna falso // se o elemento atual for menor que o anterior, a fila não está ordenada
        fim se
        anterior ← elemento // atualiza o valor anterior para o elemento atual
    fim enquanto
    retorna verdadeiro // se todos os elementos foram verificados e não houve violação de ordenação, a fila está ordenada
fim função

algoritmo OrdenarFila(F: Fila, P: Pilha)
    N ← tamanho(F) // número de elementos na fila F
    enquanto não estáOrdenada(F) faça
        maior ← -INF // assume um valor mínimo inicial para comparar
        para i de 1 até N faça
            elemento ← desenfileira(F) // remove o elemento da frente da fila F
            se elemento > maior então
```

```
            maior ← elemento // atualiza o maior elemento encontrado
        fim se
        empilha(P, elemento) // empilha o elemento na pilha P
    fim para
    enfileira(F, maior) // enfileira o maior elemento de volta na fila F
    enquanto tamanho(P) > 0 faça // desempilha e enfileira os elementos restantes na fila F
        elemento ← desempilha(P)
        enfileira(F, elemento)
    fim enquanto
  fim enquanto
fim algoritmo
```

2. Considere que as letras do seu nome completo, removendo-se as letras repetidas, foram usadas como elementos para serem inseridas em uma árvore binária de busca. Escreva em uma linha o seu nome completo em letra de forma, riscando todas as ocorrências de letras que se repetem, deixando apenas a primeira ocorrência de cada uma. Agora, desenhe abaixo a árvore de busca resultante da inserção das letras distintas em uma árvore binária de busca inicialmente vazia. Considere que uma letra é menor do que outra se vem antes no alfabeto. A ordem de inserção das letras é natural da leitura (da esquerda para direita).

2. ARGOS MAIA

A árvore de busca resultante terá a letra A como raiz, seguida pelas letras G e M como filhos da esquerda e direita, respectivamente. A letra O será inserida como filho esquerdo do nó G, a letra I será inserida como filho direito do nó G e como filho direito do nó M, e a letra S será inserida como filho direito do nó O. A letra A será inserida como filho direito do nó I e como filho direito do nó M.

3. Elabore os algoritmos solicitados abaixo (definição de Nó à direita):

a. **função** BuscaArv(T: ^Nó, x: Inteiro): Lógico
```
//assume T uma árvore
//retornar V se x é um elemento de T, ou F caso contrário
```

```
estrutura Nó:
  Elem: Inteiro
  Esq, Dir: ^Nó
```

b. **função** MaioresInternos(T: ^Nó, x: Inteiro): Inteiro
```
//assume T uma árvore
//retornar o número de elementos não folhas de T que são maiores que x
```

c. **função** NúmeroFolhasPares(T: ^Nó): Inteiro
```
//assume T uma árvore
//retornar o número de folhas de T associadas a elementos pares
```

d. **função** Média(T: ^Nó): Real
```
//assume T uma árvore
//retornar a média aritmética dos elementos associados a nós de T
```

3. a)

```
função BuscaArv(T: ^Nó, x: Inteiro): Lógico
  se T = NULO então
    retorne Falso
  senão se T^.Chave = x então
    retorne Verdadeiro
  senão se x < T^.Chave então
    retorne BuscaArv(T^.Esquerda, x)
  senão
    retorne BuscaArv(T^.Direita, x)
  fim se
fim função
```

A complexidade do algoritmo é O(h), onde h é a altura da árvore. No pior caso, em uma árvore desbalanceada, a complexidade pode ser O(n), onde n é o número de elementos na árvore.

b)

```
função NúmeroFolhasPares(T: ^Nó): Inteiro
  se T = NULO então
    retorne 0
  senão se T^.Chave mod 2 = 0 então
    retorne 1 + NúmeroFolhasPares(T^.Esquerda) + NúmeroFolhasPares(T^.Direita)
  senão
    retorne NúmeroFolhasPares(T^.Esquerda) + NúmeroFolhasPares(T^.Direita)
  fim se
fim função
```

A complexidade do algoritmo é O(n), onde n é o número de elementos na árvore. O algoritmo percorre todos os elementos da árvore para contar quantos são pares.

c)

```
função Média(T: ^Nó): Real
  se T = NULO então
    retorne 0
  senão
    retorne (T^.Chave + Média(T^.Esquerda) + Média(T^.Direita)) / (1 + Média(T^.Esquerda) + Média(T^.Direita))
  fim se
fim função
```

A complexidade do algoritmo é O(n), onde n é o número de elementos na árvore. O algoritmo percorre todos os elementos da árvore para calcular a média dos valores associados aos nós.