Estruturas Lineares

Parte A

1. **Complete a sentença:** Em uma estrutura não-linear, os elementos são organizados de forma _____.
2. Qual é a complexidade das operações de inserção/remoção/busca na estrutura `map` do C++?
3. Das estruturas disponíveis na API do C++ listadas abaixo, quais delas são implementadas usando árvores binárias de busca balanceadas?
   - ☐ `map`
   - ☐ `set`
   - ☐ `bitset`
   - ☐ `priority_queue`
4. Descreva brevemente, com um exemplo de uso, cada uma das estruturas de C++ abaixo:
   (a) `map`
   (a) `set`
   (a) `multimap`
   (a) `multiset`
5. Quais estruturas em C++ são implementadas usando o conceito de *hash*? Qual é a complexidade média esperada para as operações de inserção e remoção?
6. Defina o conceito de iteradores em C++ e dê um exemplo de uso em uma estrutura não-linear.
7. Declare uma `priority_queue` de elementos do tipo `double` que mantenha, em seu topo, o menor dentre os elementos já inseridos.

Parte B

8. Qual é a diferença entre uma *heap* e uma árvore binária de busca? Qual operação da árvore binária de busca não se aplica às *heaps*?
9. Qual é a diferença entre os três algoritmos de ordenação da API do C++, a saber: `sort`, `partial_sort` e `stable_sort`? Determine a complexidade assintótica de cada um deles.
10. Em competições de programação que permitem ao participante "*hackear*" o código de outros participantes é desaconselhado o uso das estruturas `unordered_set` e `unordered_map`. Justifique o porquê e apresente uma alternativa à estas estruturas.
11. A propriedade de uma *heap* binária pode ser violada caso exista algum nó cujo valor seja maior do que o valor de seu nó pai. O algoritmo abaixo restaura a propriedade da *heap*:

Algoritmo 1 Restauração da propriedade da *heap* binária

Entrada: O índice i do nó a ser corrigido.
Saída: Ao final do procedimento, a propriedade da *heap* está garantida.

```
1: procedure SWIN(i)
2:     p ← i/2                          ▷ p é o índice do nó pai de i
3:     if i > 1 and value(i) > value(p) then
4:         SWAP(i, j)                   ▷ Troca os valores de i e p
5:         SWIN(p)
6:     end if
7: end procedure
```

Implemente o pseudocódigo acima em C++.

12. O algoritmo abaixo implementa uma `heap` em Python. Reimplemente o mesmo código em C++.

```
class Heap:

    def __init__(self):

        self.N = 0
        self.xs = [0]


    def add(self, x):

        self.N += 1

        if len(self.xs) > self.N:
            self.xs[self.N] = x
        else:
            self.xs.append(x)

        self._swin(self.N)


    def empty(self):

        return self.N == 0


    def extract_max(self):
        M = self._top()
        self._pop()
        return M


    def _top(self):

        return self.xs[self.N]


    def _pop(self):

        self.xs[1], self.xs[self.N] = self.xs[self.N], self.xs[1]
        del self.xs[-1]
        self.N -= 1
        self._sink(1)


    def __str__(self):

        if len(self.xs[1:]) > 0:
            return ','.join([str(x) for x in self.xs[1:]])
        else:
            return ''


    def _parent(self, i):

        return i/2


    def _left(self, i):

        return 2*i


    def _right(self, i):

        return 2*i + 1


    def _swin(self, i):

        p = self._parent(i)

```

```
72          if i > 1 and self.xs[i] > self.xs[p]:
73              self.xs[i], self.xs[p] = self.xs[p], self.xs[i]
74              self._swin(p)
75
76      def _sink(self, i):
77
78          L = self._left(i)
79          R = self._right(i)
80
81          if R <= self.N and self.xs[R] > self.xs[L]:
82              M = R
83          else:
84              M = L
85
86          if M <= self.N and self.xs[M] > self.xs[i]:
87              self.xs[i], self.xs[M] = self.xs[M], self.xs[i]
88              self._sink(M)
```