

# 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`
  - (b) `set`
  - (c) `multimap`
  - (d) `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:    $p \leftarrow i/2$  ▷  $p$  é o índice do nó pai de  $i$   
3:   if  $i > 1$  and  $value(i) > value(p)$  then  
4:     SWAP( $i, p$ ) ▷ 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++.

```
1 class Heap:
2
3     def __init__(self):
4
5         self.N = 0
6         self.xs = [0]
7
8
9     def add(self, x):
10
11         self.N += 1
12
13         if len(self.xs) > self.N:
14             self.xs[self.N] = x
15         else:
16             self.xs.append(x)
17
18         self._swin(self.N)
19
20
21     def empty(self):
22
23         return self.N == 0
24
25
26     def extract_max(self):
27         M = self._top()
28         self._pop()
29         return M
30
31
32     def _top(self):
33
34         return self.xs[self.N]
35
36
37     def _pop(self):
38
39         self.xs[1], self.xs[self.N] = self.xs[self.N], self.xs[1]
40         del self.xs[-1]
41         self.N -= 1
42         self._sink(1)
43
44
45     def __str__(self):
46
47         if len(self.xs[1:]) > 0:
48             return ','.join([str(x) for x in self.xs[1:]])
49         else:
50             return ''
51
52
53     def _parent(self, i):
54
55         return i/2
56
57
58     def _left(self, i):
59
60         return 2*i
61
62
63     def _right(self, i):
64
65         return 2*i + 1
66
67
68     def _swin(self, i):
69
70         p = self._parent(i)
71
```

```

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)

```