**UNIVERSIDADE DE SÃO PAULO**
**ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES**
**ACH2002 - INTRODUÇÃO À ANÁLISE DE ALGORITMOS**

ARTHUR CALHARES DE ALMEIDA SANCHEZ
JULIANNE AMANDA DE SÁ LAURINDO

# FILA DE PRIORIDADES COM *HEAP*

**SÃO PAULO**
**2020**

ARTHUR CALHARES DE ALMEIDA SANCHEZ
JULIANNE AMANDA DE SÁ LAURINDO

# FILA DE PRIORIDADES COM *HEAP*

Trabalho de alunos de Graduação do 2.º semestre de 2020 com objetivo de compor a nota da disciplina de Introdução à Análise de Algoritmos.
Orientadora: Prof.ª Karina Valdivia Delgado

**SÃO PAULO**
**2020**

# SUMÁRIO

# 1. ANÁLISE DOS PSEUDOCÓDIGOS

Nesta seção, serão analisados os pseudocódigos das funções *heapIncreaseKey()* e *heapInsert()*. Na implementação utilizada por este trabalho, ambas as funções chamam uma função auxiliar, *construirMaxHeap()*, que por sua vez chama *maxHeapify()*, analisada durante as aulas da disciplina Introdução à Análise de Algoritmos como $O(\lg n)$. A figura a seguir descreve o pseudocódigo de *construirMaxHeap()*:

```
1   construirMaxHeap(A, n) {
2       começo = (n / 2) - 1                          // O(1)
3       para (i = começo; i >= 0; i = i - 1) {        // O(n/2)
4           maxHeapify(A, n, i)                       // n/2 * O(lg n)
5       }
6   }
7   // Resultado: O(n*lg n)
```

Por chamar $O(\lg n)$ um total de $\frac{n}{2}$ vezes, então essa função pode ser considerada $O(n \lg n)$, como visto na imagem.

### 1.1. *heapIncreaseKey()*

Para modificar a prioridade de um elemento do *heap*, a função *heapIncreaseKey()* realiza tal alteração e em seguida chama *construirMaxHeap()*, que reorganiza a estrutura.

```
1   heapIncreaseKey(A, n, i, p) {
2       A[i].prioridade = p                           // O(1)
3       construirMaxHeap(A, n)                        // O(n*lg n)
4   }
5   // Resultado: O(n*lg n)
```

Como a alteração de um campo possui tempo constante, então o consumo de tempo desse algoritmo é igual ao da função nele chamada: $O(n \lg n)$.

### 1.2. *heapInsert()*

Para poder inserir um elemento no *heap* que caracteriza a fila de prioridades, a função *heapInsert()* primeiramente analisa se há espaço no arranjo, nesta implementação definido com até 4.000 casas, bem como se a prioridade não excede 900.000 ou é menor que 0. Caso positivo em alguma dessas condições, a execução do algoritmo é interrompida. Caso

contrário, é analisado se o *heap* está vazio ou não, e, caso contrário, há uma chamada da função *construirMaxHeap()* para que a estrutura seja reordenada.

```
1   heapInsert(A, n, i, p) {
2       se (n >= 4000 OU p < 0 OU p > 900000) {
3           retornar falso                                  // O(1)
4       } senão {
5           se (n é igual a 0) {
6               A[0].identificador = i                      // O(1)
7               A[0].prioridade = p                         // O(1)
8               n = n+1                                     // O(1)
9           } senão {
10              A[n].identificador = i                      // O(1)
11              A[n].prioridade = p                         // O(1)
12              n = n+1                                     // O(1)
13              construirMaxHeap(A, n)                      // O(n*lg n)
14          }
15          retornar verdadeiro                             // O(1)
16      }
17  }
18  // Resultado: O(n*lg n)
```

Novamente, como a maioria das operações é de comparação ou atribuição, então possuem tempo constante, com exceção da chamada de *construirMaxHeap()*. Logo, essa função é também $O(n \lg n)$.

# 2. DIFICULDADES ENCONTRADAS

A implementação das funções solicitadas pela proposta deste trabalho, realizada na linguagem de programação C, não apresentou grandes dificuldades. É possível, entretanto, destacar o desenvolvimento da função *main()*, que comanda a entrada e saída, como uma das partes mais complexas, em especial quando é chamada a opção 3, relativa ao *heapInsert()*. Isso porque essa opção exige uma segunda entrada de dois números de tipos diferentes separados apenas por espaço, ou seja, na mesma linha, de modo que se optou pelo uso da função *fgets()*, que por sua vez converte a linha inteira em uma *String*; foi necessário então separá-la no espaço em duas novas strings, em seguida convertidas uma para *int* e outra para *float*.