# Lista 3 - Complexidade

ESTRUTURA DE DADOS I – Pedro Nuno Moura

Monitor: Celio Ferreira Camara Júnior

1) Diga a ordem de complexidade de cada um dos trechos de código a seguir:

a.

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    a = a + rand();
}
for (j = 0; j < M; j++) {
    b = b + rand();
}
```

b.

```
int a = 0;
for (i = 0; i < N; i++) {
    for (j = N; j > i; j--) {
        a = a + i + j;
    }
}
```

c.

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```

2) Assuma que cada uma das expressões abaixo represente o tempo de processamento *T(n)* gasto por um algoritmo para resolver um problema cuja entrada possui tamanho *n*. Diga o termo dominante e a complexidade justa Big-Oh em cada expressão.

| | Expressão | Termo dominante | $O()$ |
|---|---|---|---|
| a | $5 + 0.001n^5 + 0.025n$ | | |
| b | $500n + 100n^{0.5} + 50n\log_{10} n$ | | |
| c | $0.3n + 5n^{1.5} + 2.5\ n^{1.75}$ | | |

| | | | |
|---|---|---|---|
| d | $n^2 \log_2 n + n(\log_2 n)^2$ | | |
| e | $n \log_3 n + n \log_2 n$ | | |
| f | $3 \log_8 n + \log_2 \log_2 \log_2 n$ | | |
| g | $100n + 0.01n^2$ | | |
| h | $0.01n + 100n^2$ | | |
| i | $2n + n^{0.5} + 0.5n^{1.25}$ | | |
| j | $0.01n \log_2 n + n(\log_2 n)^2$ | | |
| k | $100n \log_3 n + n^3 + 100n$ | | |
| l | $0.003 \log_4 n + \log_2 \log_2 n$ | | |

3) A seguir estão três implementações usando lógicas diferentes para realizar a tarefa de encontrar um elemento em um vetor ordenado. Diga qual a complexidade de cada método e explique como chegou a ela.

i)

```
private int retornaPosicao(int[] vetor, int desejado)
{
	int contador = 0;
	while(contador < vetor.length)
	{
		if(vetor[contador] == desejado)
		{
			return contador;
		}
	contador++;
	}
	return Integer.MIN_VALUE;
}
```

ii)

```
private int retornaPosicao(int[] vetor, int desejado)
{
	for(int contador = 0; contador < vetor.length; contador++)
	{
		if(vetor[contador] == desejado)
		{
			return contador;
		}
	}
	return Integer.MIN_VALUE;
}
```

iii)

```
private int retornaPosicao(int[] vetor, int desejado)
{
	int low = 0;
	int high = vetor.length - 1;
	while (low <= high)
	{
		int mid = (low + high) / 2;
		if (vetor[mid] < desejado)
		{
			low = mid + 1;
		}
		else if (vetor[mid] > desejado)
		{
			high = mid - 1;
		}
		else if (vetor[mid] == desejado)
		{
			return mid;
		}
	}
	return Integer.MIN_VALUE;
```

```
}
```

4) Implemente um método com a melhor complexidade possível para conseguir a quantidade de inteiros pares de uma pilha. Use a classe Pilha aprendida em sala de aula que possui os métodos p*op* e *push* padrões, além de atributos padrões. Você possui liberdade para criar novos métodos e/ou atributos e alterar os métodos *pop* e *push* já existentes.

5) O método abaixo implementa em Java o algoritmo de ordenação *Selection Sort*. Mostre qual é a complexidade assintótica de tal método, exibindo também a conta que levou a tal complexidade.

```
public void selectionSort(int vetor[], int n)
{
      int i,j;
      for (j = 0; j < n-1; j++)
      {
            int iMin = j;

            for (i = j+1; i < n; i++)
            {
                  if (vetor[i] < vetor[iMin])
                  {
                        iMin = i;
                  }
            }

            if(iMin != j)
            {
                  int temp = vetor[j];
                  vetor[j] = vetor[iMin];
                  vetor[iMin] = temp;
            }
      }
}
```