Lista de Exercícios 2 (Noções de Complexidade)

João Lucas Ribeiro – 2019005856

1. Faça uma pesquisa e escreva um exemplo de algoritmo para cada uma das classes abaixo:

(a) O(1): ordem constante.

Se cada linha de comando do algoritmo for O(1), logo, o algoritmo em si será O(1). Isto é, se a quantidade de comandos se limitar a uma constante definida, não importa o tamanho do problema, a ordem será constante. Por exemplo, em um algoritmo com laço "for" do tipo:

```
for (i = 0; i < 10; i++) {   //Ou seja, O(1), logo, toda a repetição é O(1)
      x = x + v;
      printf ("%d", x);
}
```

Como se tem um "i < k", com k definido a um valor fixo (no exemplo, k = 10), onde k não é f(n), o algoritmo terá um tempo de execução sempre constante O(g(n)).

(b) O(log(n)): ordem logarítmica.

Considera-se um algoritmo de repetição multiplicativa, em que cada iteração atualiza o controle mediante uma multiplicação ou divisão. Por exemplo:

```
Limite = 1;                  // o número de iterações depende
while (limite <= n)          // de n; limite vai dobrando a cada
{                            // iteração; depois de k iterações, limite = 2k e
      trecho com O(1)        // k = log2 limite; como o valor
      limite = limite * 2;   // máximo de limite é n, então
}                            // o trecho é O(log2n) = O(log n)
```

(c) O(n): ordem linear.

Como algoritmo desta ordem, pode-se considerar uma chamada de procedimento, que pode ser resolvida considerando-se que o procedimento também tem um algoritmo com sua própria complexidade. Também pode se tratar de uma chamada recursiva. Embora não haja um método único para esta avaliação, em geral a complexidade de um algoritmo recursivo será função de componentes como: a complexidade da base e do núcleo da solução e a profundidade da recursão (número de vezes que o procedimento é invocado recursivamente). Como um exemplo, considere o algoritmo do cálculo fatorial a seguir:

```
int fatorial (int n){
    if (n==0){
        return 1; // Base
    else{
        return n * fatorial (n - 1); //Núcleo
    }
}
```

A redução do problema se faz de uma em uma unidade, a cada reinvocação do procedimento, a partir de n, até alcançar n = 0. Logo, a profundidade da recursão é igual a n. Nesse caso, conclui-se que o algoritmo tem um tempo T(n) = n*1 + 1 = O(n).

(d) O(n . log(n)): ordem log linear.

Se o número de iterações é função de n, pela regra do produto, a complexidade da repetição é representada pela complexidade do corpo multiplicada pela função que descreve o número de iterações. Como exemplo disso, pode-se considerar o algoritmo abaixo:

```
for (i = 0; i < k * n; i++){   // o trecho é O(f(n)*g(n)), no caso
    trecho com O(log n)  // O(k*n*log n), ou seja: O(n log n)
}
```

(e) O(n2): ordem quadrática.

As repetições aninhadas, como num laço "for" duplo, constituem um algoritmo de ordem quadrática. Por exemplo:

```
for (i = 0; i < n; i++){       // o trecho é O(f(n)*g(n)), no caso
    for (j=0; j < n; j++){  // g(n)=n*1 (laço interno); logo,
        trecho com O(1)  // O(n*n), ou seja: O(n²)
    }
}
```

(f) $O(n^3)$: ordem cúbica.

Sob a perspectiva do item anterior, como as repetições são contadas em k níveis (desde que todos os índices dependam do tamanho do problema), a complexidade da estrutura aninhada será da ordem de n^k. Para um algoritmo adquirir grau 3, considera-se um "laço aninhado triplo", com n^k, k = 3. Considere o algoritmo abaixo como exemplo:

```
for (IndExt =1; IndExt <= n ; IndExt++){  // o laço mediano é
    for (IndMed = IndExt; IndMed <=n; IndMed++){  // executado
        for (IndInt = 1; IndInt <= IndMed; IndInt++){  // n+n-1+n-2+...
            trecho com O(1)  // +2+1=(n²+n)/2 vezes;
        }
    }
}
```

Aqui, o laço interno será executado no máximo n vezes. Logo, tem-se $O((n^2+n)*n)$, ou seja: $O(n^3)$.

(g) $O(2^n)$: ordem exponencial.

Corresponde a algoritmos que utilizam força-bruta na solução de problemas – tempo de execução exponencial. Algoritmos com esta performance são impraticáveis. Sempre que N dobra, o tempo de execução é elevado ao quadrado. Algoritmos combinatórios são exemplos disso, e pode-se citar o algoritmo solução para o "Problema do Caixeiro Viajante", cujo código pode ser encontrado no link abaixo, em Linguagem Python:

<https://github.com/vitornascimento95/AlgoritmoGenetico/blob/master/Algoritmo.py>

(h) $O(n!)$: ordem fatorial.

A função de complexidade é uma função fatorial do tamanho da entrada do algoritmo. Exemplo: $f(n) = O(n!)$. $f(n) = O(n!)$. Um exemplo de algoritmo fatorial é o algoritmo para gerar todas as permutações de um vetor de inteiros, cujo código pode ser encontrado no link abaixo, em Linguagem C:

< https://gist.github.com/marcoscastro/60f8f82298212e267021>