Exercícios Teóricos – u01: Noções de Complexidade

Catarina F. M. Castro (803531) – AEDs II

Exercícios Iniciais

1-

a) $2^0 = 1$
b) $2^1 = 2$
c) $2^2 = 4$
d) $2^3 = 8$
e) $2^4 = 16$
f) $2^5 = 32$
g) $2^6 = 64$
h) $2^7 = 128$
i) $2^8 = 256$
j) $2^9 = 512$
k) $2^{10} = 1024$
l) $2^{11} = 2048$

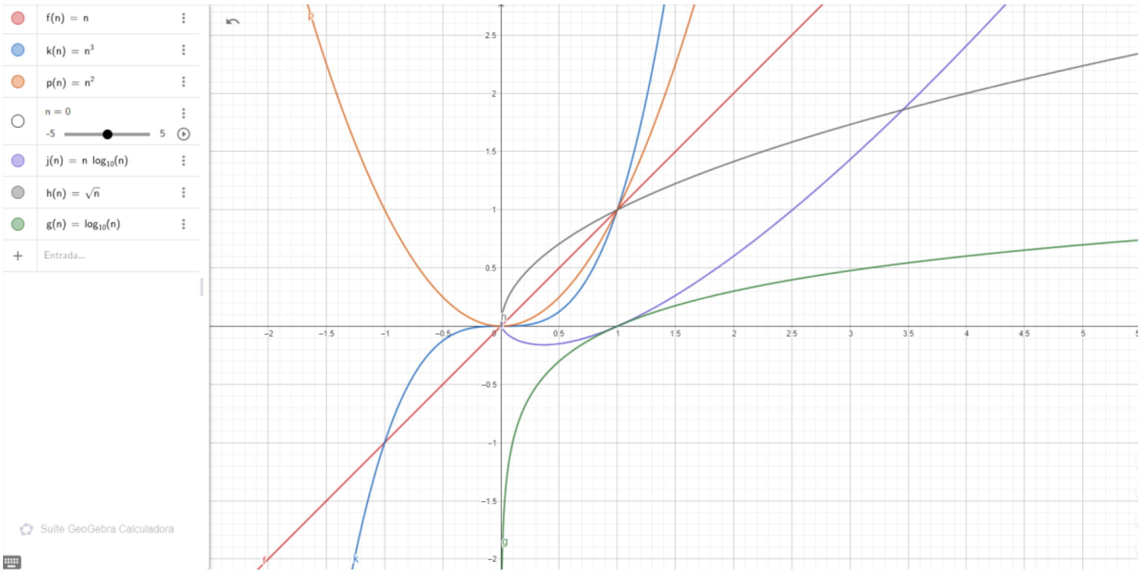
2-

a) $\log(2048) = 11$
b) $\log(1024) = 10$
c) $\log(512) = 9$
d) $\log(256) = 8$
e) $\log(128) = 7$
f) $\log(64) = 6$
g) $\log(32) = 5$
h) $\log(16) = 4$
i) $\log(8) = 3$
j) $\log(4) = 2$
k) $\log(2) = 1$
l) $\log(1) = 0$

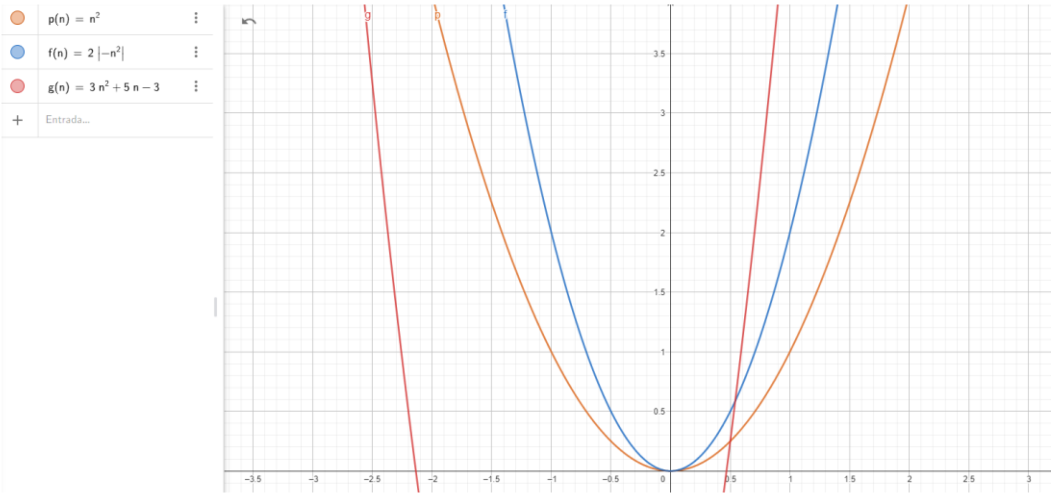
3-

a) $\lceil 4{,}01 \rceil = 5$
b) $\lfloor 4{,}01 \rfloor = 4$
c) $\lceil 4{,}99 \rceil = 5$
d) $\lfloor 4{,}99 \rfloor = 4$
e) $\lceil \lg(16) \rceil = 4$
f) $\lfloor \lg(16) \rfloor = 4$
g) $\lg(17) = 4{,}087$
h) $\lceil \lg(17) \rceil = 5$
i) $\lfloor \lg(17) \rfloor = 4$
j) $\lg(15) = 3{,}907$
k) $\lceil \lg(15) \rceil = 4$
l) $\lfloor \lg(15) \rfloor = 3$

4- Plotar os gráficos das respectivas funções



5- Plotar os gráficos das respectivas funções



Contagem de Operações

Estrutura sequencial e condicional

1-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
a--;
a -= 3;
a = a - 2;
```

R: 3 subtrações

2-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
if (a - 5 < b - 3){
        i--;
        --b;
        a -= 3;
} else {
        j--;
}
```

R: 3 subtrações no melhor caso, e 5 no pior

3-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
if (a - 5 < b - 3 || c - 1 < d - 3){
        i--;
        --b;
        a -= 3;
} else {
        j--;
}
```

R: 5 subtrações no melhor caso e 7 subtrações no pior caso.

Estrutura de repetição simples

4-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < 4; i++){
        a--;
}
```

R: 4 subtrações

5-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
        a--;
        b--;
}
```

Observação: Sua resposta deve ser em função de ***n***

R: $2n$ subtrações

6-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 3; i < n; i++){
        a--;
}
```

R: $n - 3$ subtrações

7-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
int i = 0, b = 10;

while (i < 3){
        i++;
        b--;
}
```

R: 3 subtrações

8-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
int i = 0, b = 10;

do {
        i++;
        b--;
} while (i < 3);
```

R: 3 subtrações

Estrutura de repetição dupla

9-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < 3; i++){
        for (int j = 0; j < 2; j++){
                a--;
        }
}
```

R: $3 * 2 * 1 = 5$ subtrações

Estrutura de repetição com custo logarítmico

10-

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = n; i > 0; i /= 2){
        a *= 2;
}
```

R: Como são realizadas divisões sucessivas, o código realiza $\lg(n) + 1$ multiplicações

Mais exercícios resolvidos

11-

- Faça um método que receba um número inteiro n e efetue o **número de subtrações** pedido em:
  a) $3n + 2n^2$
  b) $5n + 4n^3$
  c) $\lg(n) + n$
  d) $2n^3 + 5$
  e) $2n^4 + 2n^2 + n/2$
  f) $\lg(n) + 5 \lg(n)$

a)

```
...
i = 0;
while (i < n){
        i++;
        a--; b--; c--;
}
for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
                a--; b--;
        }
}
```

b)

```
…
i = 0;
while (i < n){
      i++;
      a--; b--; c--; d--; e--;
}
for (i = 0; i < n; i++){
      for (j = 0; j < n; j++){
            for (k = 0; k < n; k++, a--, b--, c--, d--);
      }
}
```

c)

```
…
i = 1;
while (i < n){
      i*= 2;
      a--;
}
for (i = 0; i < n; i++){
      a--;
}
```

d)

```
…
for (i = 0; i < n; i++){
      for (j = 0; j < n; j++){
            for (k = 0; k < n; k++, a--, b--);
      }
}
a--; b--; c--; d--; e--;
```

e)

```
…
for (i = 0; i < n; i++){
      for (j = 0; j < n; j++){
            a--; b--;
            for (k = 0; k < n; k++){
                  for (l = 0; l < n; l++, c--; d--);
                  }
            }
      if (i % 2 == 0) e--;
}
```

f)

```
…
i = 1;
while (i < n){
      i*= 2;
```

```
        a--; b--; c--; d--; e--; f--;
}
```

Funções de Complexidade

1-

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];

for (int i = 1; i < n; i++){
        if (min > array[i]){
                min = array[i];
        }
}
```

Nesse código, a operação relevante é a comparação entre elementos de um array, que será executada $T(n) = n - 1$ vezes em todos os casos (visto que não há melhor ou pior caso).
Dessa forma, o algoritmo pode ser considerado como ótimo, visto que é preciso fazer sempre o mesmo número de operações para garantir a resposta.

2-

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
        if (array[i] == x){
                resp = true;
                i = n;
        }
}
```

No melhor caso, no qual o elemento desejado está na primeira posição, são realizados $t(n) = 1$ comparações ente elementos do array.
Já na pior situação, são realizadas $t(n) = n$ comparações. Isso ocorreria quando o elemento buscado está na última posição ou não está no array.

3-

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa binária no melhor e no pior caso

```
boolean resp = false;
int dir = n - 1, esq = 0, meio, diferença;
while (esq <= dir) {
        meio = (esq + dir) / 2;
        diferença = (x - array[meio]);
        if (diferenca == 0){
                resp = true;
                esq = n;
        } else if (diferença > 0){
                esq = meio + 1;
        } else {
                dir = meio - 1;
        }
}
```


No melhor caso, quando o elemento desejado está na posição [(esq+dir)/2], a função de

complexidade da pesquisa é $t(n) = 1$.
Já, no pior caso, a função é $t(n) = \lg(n)$, visto que são feitas divisões sucessivas do escopo em cada iteração. Esse caso acontece quando o elemento ou está na primeira/ultima posição do array ou não está contido.

4-

- Explique porque o Algoritmo de Seleção realiza $m(n) = 3n - 3$ movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {
        int menor = i;
        for (int j = (i + 1); j < n; j++){
                if (array[menor] > array[j]){
                        menor = j;
                }
        }
        swap(menor, i);
}
```

```
void swap(int a, int b) {
   int temp = array[a];
   array[a] = array[b];
   array[b] = temp;
}
```

Como o laço se repete $(n-1)$ vezes, e cada loop executa um *swap* que realiza 3 movimentações, são feitas então $t(n) = 3 * (n-1) = 3n - 3$ movimentações de registros.

5-

- Modifique o código do Algoritmo de Seleção para que ele contabilize o número de movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {
        int menor = i;
        for (int j = (i + 1); j < n; j++){
                if (array[menor] > array[j]){
                        menor = j;
                }
        }
        swap(menor, i);
}
```

As alterações estão marcadas de amarelo.

```
int mov = 0;
for (int i = 0; i < (n - 1); i++) {
      int menor = i;
      for (int j = (i + 1); j < n; j++){
            if (array[menor] > array[j]){
            menor = j;
            }
      }
      swap(menor, i);
      mov += 3;
}
printf("Teoria: " + (3*n - 3));
printf("Prática: " + mov);
```

6-

- Explique porque o Algoritmo de Seleção realiza $c(n) = \frac{n^2}{2} - \frac{n}{2}$ comparações entre registros

```
for (int i = 0; i < (n - 1); i++) {
    int menor = i;
    for (int j = (i + 1); j < n; j++){
        if (array[menor] > array[j]){
            menor = j;
        }
    }
    swap(menor, i);
}
```

Como as comparações desejadas estão sendo feitas no *IF*, com o laço externo se repetindo $(n - 2)$ vezes e o laço interno se repetindo $n - (n - 1)$ vezes, temos que:

$$\boldsymbol{c(n) = \Sigma_{i=0}^{n-2}(n - i - 1)}$$

Notação Θ

1-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
    if (rand() % 2 == 0){
        a--;
        b--;
    } else {
        c--;
    }
}
```

No melhor caso, o código realiza $f(n) = n$, ou seja, $\Theta(n)$. Já no pior caso, são realizadas $f(n) = 2n$, logo, $\Theta(n)$.

2-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
int i = 10;
while (i >= 7){
    i--;
}
```

R: 3 subtrações.

3-

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 5; i >= 2; i--){
    a--;
}
```

R: 4 subtrações.

4-

- Calcule o número de subtrações que o código abaixo realiza:

```
...
for (int i = 0; i < 5; i++){
        if (i % 2 == 0){
                a--;
                b--;
        } else {
                c--;
        }
}
```

R: 8 subtrações

5-

- Calcule o número de subtrações que o código abaixo realiza:

```
...
int i = 10, b = 10;
while (i > 0){
        b--;
        i = i >> 1;
}
i = 0;
while (i < 15){
        b--;
        i += 2;
}
```

R: 12 subtrações

6-

- Calcule o número de multiplicações que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
        for (int j = 0; j < n - 3; j++){
                a *= 2;
        }
}
```

R: $n(n-3)$ multiplicações

7-

- Calcule o número de multiplicações que o código abaixo realiza:

```
...
for (int i = n - 7; i >= 1; i--){
        for (int j = 0; j < n; j++){
                a *= 2;
        }
}
```

R: $(n-7)n$ multiplicações

8-

- Calcule o número de multiplicações que o código abaixo realiza:

```
...
for (int i = n - 7; i >= 1; i--){
        for (int j = n - 7; j >= 1; j--){
                a *= 2;
        }
}
```

R: $(n-7)^2$ multiplicações.

9-

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = n; i > 1; i /= 2){
        a *= 2;
}
```

R: $piso(\lg(n)) + 1$ multiplicações.

10-

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = n + 1; i > 0; i /= 2)
        a *= 2;
}
```

R: $piso(\lg(n)) + 1$ multiplicações.

11-

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = 1; i < n; i *= 2)
        a *= 2;
}
```

R: $teto(\lg(n))$ multiplicações.

12-

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = 1; i <= n; i*= 2)
        a *= 2;
}
```

R: $teto(\lg(n) + 1)$ multiplicações.

13-

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = n+4; i > 0; i >>= 1){
        a *= 2;
}
```

R: $piso(\lg(n + 4)) + 1$ multiplicações.