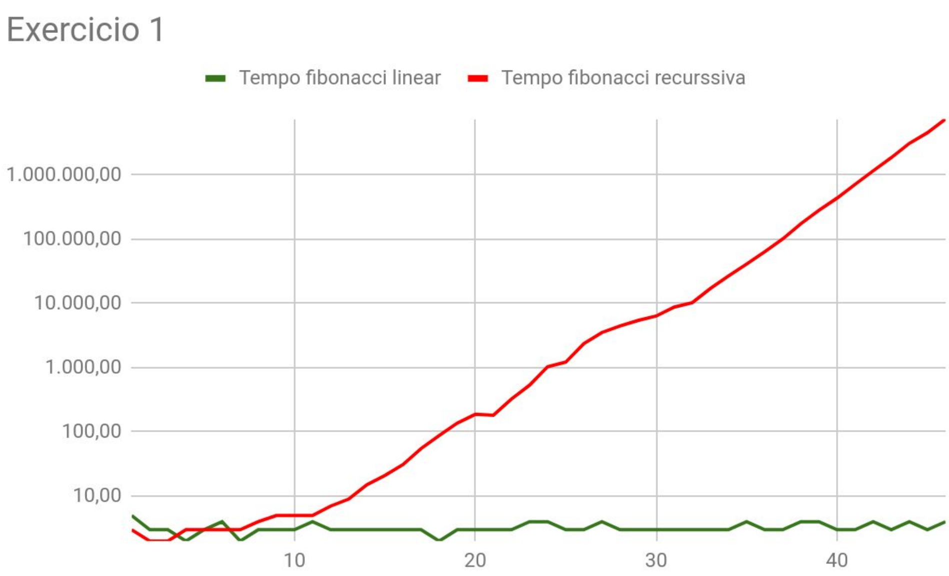
UDESC - Universidade do Estado de Santa Catarina **Data:** 16/12/2020
Aluno: Menderson Nicolau Costa
Disciplina: Complexidade de Algoritmos

Lista de Exercicios 1

1)
A função foi implementada em C, a saída da função exporta no arquivo saida_ex1.csv os tempos em milissegundos do cálculo do fatorial de “n” com as funções sequencial e recursiva, com “n” variando de 0 a 45. O gráfico a seguir foi gerado com os valores da saída na aplicação online Google Sheets.



2)
```
unsigned int potencia (unsigned int b, unsigned int e)
{
	unsigned int r;		//O(1)
	if (e == 0)		//O(1)
		return 1;	//O(1)
	r = potencia(b, e/2);	//Análise A)
	if (e % 2 == 0)		//O(1)
		return r*r;	//O(1)
	else
		return r*r*b;	//O(1)
}
```

Análise A: Cada chamada recursiva executa 5 vezes O(1) até o seu retorno, logo:
$T(n)=T(n/2) + 5$
$T(n/2)=T(n/2^2) + 5$
$T(n/2^2)=T(n/2^3) + 5$
…

$T(n/2^{l-1})=T(n/2^l) + 5$
$n/2^l=1$
$n=2^l$
$l=\log_2 n$
$T(n)=(T(1) + 5)*\log_2 n$
$T(n) = 6*\log_2 n$
Complexidade de tempo: **O(log n)**.
Complexidade de espaço: A pilha será incrementada 1 vez a cada chamada de recursão, dessa forma a complexidade de espaço também será **O(log n)**.

3)
a) T(n) = T(n/2) + n
T(1) = 1
$T(n) = T(n/2) + n$
$T(n/2) = T(n/2^2) + n/2 + n$
$T(n/2^2)= T(n/2^3) +n/2^2 + n/2 + n$
$T(n/2^3)= T(n/2^4) +n/2^3 +n/2^2 + n/2 + n$
…
$T(n/2^{l-1}) = T(n/2^l) + n/2^{l-1} + \ldots +n/2^3 +n/2^2 + n/2 + n$
$n/2^l=1$ ->base
$n=2^l$
$l=\log_2 n$
$T(n)= T(1) + n/2^{l-1} + \ldots +n/2^3 +n/2^2 + n/2 + n = 1+n(1+1)$
Resp: $T(n)= 1 + 2n$

b) T(n) = 2T(n - 1) + n
T(1) = 1
$T(n) = 2T(n - 1) + n-1$
$2T(n - 1)=2^2T(n-2)+2(n-1)$
$2^2T(n-2)=2^3T(n-3)+2^2(n-2)$
$2^3T(n-3)=2^4(n-4)+2^3(n-3)$
…
$2^{l-1}T(n-(l-1)=2^lT(n-l)+2^{l-1}(n-(l-1))$

$n-l=1$ ->base
$n=l+1$
$l=n-1$

$T(n) = n + 2(n-1) + 2^2(n-2) + 2^3(n-3) + \ldots + 2^{l-1}(2) + 2^l1$

$T(n)= \sum_{i=0}^{n-1} 2^i(n-1)$
$= \sum_{i=0}^{n-1} (2^in - 2^ii)$
$= \sum_{i=0}^{n-1} 2^in - \sum_{i=0}^{n-1} 2^ii$

$=n \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} 2^i i$

$=n(2^n-1) = 2^n n - n = \mathbf{2^{n+1} - n}$

$=(n*2^n-3*2^n+4)/2 = n*2^{n-1}-3*2^{n-1}+2$

logo,

$T(n)= \sum_{i=0}^{n-1} 2^i(n-1)= n \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} 2^i i = 2^{n+1} - n - (n*2^{n-1}-3*2^{n-1}+2) = 2^{n-1} + 3*2^{n-1}-n-2 = \mathbf{2^{n+1}-n-2}$

Resp: 2^{n+1}-n-2

c) T(n) = 4T(n/2) + n

T(1) = 1

$T(n) = 4T(n/2) + n$

$4T(n/2)=4^2T(n/2^2)+4n/2$

$4^2T(n/2^2)=4^3T(n/2^3) + 4^2n/2^2$

$4^3T(n/2^3) = 4^4T(n/2^4) + 4^3n/2^3$

…

$4^{l-1}T(n/2^{l-1}) = 4^lT(n/2^l) + 4^{l-1}n/2^{l-1}$

$n/2^l = 1$-> base

$n = 2^l$

$l = \log_2 n$

$T(n) = n + 4n/2 + 4^2/n^2 + 4^3/n^3 + … + 4^{l-1}/2^l + 4^l*1$

$T(n) = 4^0n/2^0 + 4n/2 + 4^2/n^2 + 4^3/n^3 + … + 4^{l-1}/2^l + (4^l*n)/2^l$

$T(n) = \sum_{i=0}^{l} (4^i*n)/2^i$

$T(n) = n* \sum_{i=0}^{l} 4^i*/2^i = n* \sum_{i=0}^{l} (4/2)^i = n* \sum_{i=0}^{l} 2^i$

Achando a fórmula fechada pelo método da perturbação:

$T(n) = n* \sum_{i=0}^{l} 2^i = n(2^{l+1} - 1) = n(2*2^l - 1) = n*(2n - 1) = \mathbf{2n^2 - n}$

Resp: T(n) = 2n² – n

d) T(n) = T(n/2) + $\log_2$ n

T(1) = 1

$T(n/2^0) = T(n/2^1) + \log_2 n$

$T(n/2^1) = T(n/2^2) + \log_2 n/2$

$T(n/2^2) = T(n/2^3) + \log_2 n/2^2$

...

$T(n/2^{l-1}) = T(n/2^l) + \log_2 n/2^{l-1}$

$n/2^l = 1$-> base

$n = 2^l$

$l = \log_2 n$

$T(n) = \log_2 n + \log_2 n/2 + \log_2 n/2^2 + ... + \log_2 n/2^{l-1} + 1$

$T(n) = \log_2 n + (\log_2 n - \log_2 2) + (\log_2 n - \log_2 2^2) + … + (\log_2 n - \log_2 2^{l-1}) + 1$

$T(n) = \log_2 n * l - (1 + 2 +3 + … + l - 1) + 1$

$T(n) = \log_2 n * \log_2 n - (1 + 2 +3 + … + l - 1) + 1$

$T(n) = \log^2_2 n - \sum_{i=0}^{l-1} i + 1$

$T(n) = \log^2_2 n - ((l-1)*(l-1+1))/2 + 1$

$T(n) = \log^2_2 n - ((l-1) * l)/2 + 1$

$T(n) = \log^2_2 n - (l^2 - l)/2 + 1$

$\mathbf{T(n) = \log^2_2 n - (\log^2_2 n - \log_2 n)/2 + 1}$

4) Usando indução matemática prove que a solução da relação de recorrência:
$T(n) = 4T(n/2) + n$
$T(1) = 1$
é
$T(n) = 2n^2 – n$, para n = 2 k e k ≥ 1.

Base para k=1
$T(2) = 4T(1) + 2$
$T(2) = 4*1+2 = 6$
Usando a fórmula:
$T(2) = 2*2^2-n = 2*4-2 = 6$

Passo indutivo:
Hipótese:
$T(2^k) = 2*(2^k)^2 - 2^k$
Passo:
$T(2^{k+1}) = 2*(2^{k+1})^2 - 2^{k+1}$
Na relação de recorrência:
$T(2^{k+1}) = 4T(2^k) + 2^{k+1}$
$T(2^{k+1}) = 4*(2*(2^k)^2-2^k) + 2^{k+1}$
$2*(2^{k+1})^2 - 2^{k+1} = 4*(2*(2^k)^2-2^k) + 2^{k+1}$
$2*(2^k*2)^2 - 2^{k+1} = 4*(2*2^{2k}-2^k) + 2^{k+1}$
$2*2^{2k}*2^2 - 2^{k+1} = 4*2*2^{2k} - 4*2^k + 2^{k+1}$
$\cancel{4*2*2^{2k}} - 2^{k+1} = \cancel{4*2*2^{2k}} - 4*2^k + 2^{k+1}$ (Dividindo ambos os lados por $4*2*2^{2k}$)
$- 2^{k+1} = - 4*2^k + 2^{k+1}$
$- 2*2^k = - 4*2^k + 2*2^k$ (Somando)
$- 2*2^k = - 2*2^k$ Provado por indução

5)Algoritmo em exercicios.c
- Inserção no Início: O(1) – A pilha sempre aponta para seu inicio, dessa forma inserção no início não é necessário percorrer a pilha, então possui complexidade O(1). A criação de um novo nó tem complexidade O(1). Após, um elemento é inserido na primeira posição. O algoritmo funciona da mesma forma no caso de a lista estar inicialmente vazia.

- Inserção no Fim: O(n) – A complexidade é proporcional à quantidade de elementos presentes na pilha, sendo o tamanho da pilha o “n”, enquanto i->ptr != NULL é executado o algoritmo com complexidade O(1), então para buscar o último elemento da lista, são executadas “n” vezes. Logo complexidade O(n).

6)

$T(n) = 2T(n/2)+1$

$T(1)=1$

$2T(n/2) = 2^2T(n/2^2)+1$

$2^2T(n/2^2) = 2^2T(n/2^3)+1$

…

$2^{l-1}T(n/2^{l-1}) = 2^{l-1}T(n/2^l)+1$

$n/2^l = 1$-> base

$n = 2^l$

$l = \log_2 n$

$T(n)=1 + l*1$

$T(n)=1 + \log_2 n * 1 =$ $\mathbf{\log_2 n +1}$

Complexidade de Tempo: **O(log n)**

Complexidade de Espaço: **O(log n)**

Essa complexidade faz sentido, uma vez que a árvore está balanceada e a altura dela será $\log_2 n$.

A implementação da versão não recursiva está disponível no arquivo codigo.c