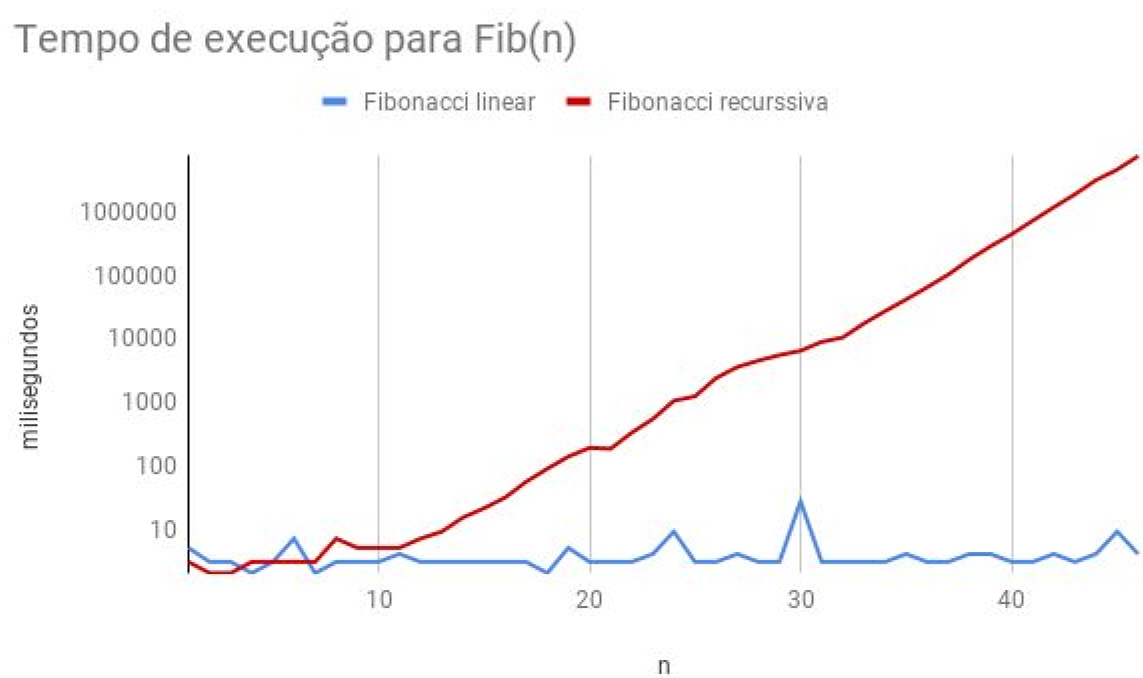
Complexidade de Algoritmos, 2020/2
Paulo Roberto Albuquerque

1) Implemente uma função que calcule o n-ésimo termo da sequência de Fibonacci com complexidade de tempo linear. Construa um gráfico comparando a execução da versão linear com a versão recursiva definida abaixo:

```
unsigned int fib (unsigned int n)
{
  if (n < 2)
    return n;
  return fib (n-2) + fib (n-1);
}
```

Resposta:
Implementação no arquivo **.c**
Para confecção do gráfico, foi-se criado um programa auxiliar para exportar resultados das duas versões de Fibonacci em um arquivo **.csv** e plotado usando o Google Sheets.



2) Para a função ***potência*** definida abaixo: mostre qual a relação de recorrência que descreve o tempo de execução da função. Resolva essa relação de recorrência. Calcule a complexidade de tempo e a complexidade espaço para essa função.

```
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);      (Chamada recursiva)
  if (e % 2 == 0)            O(1)
    return r*r;              O(1)
  else                       ↕ Esses dois não podem ocorrer numa mesma execução
    return r*r*b;            O(1)
}
```

Resposta:
Para calcular a **complexidade de tempo** devemos fazer:

$T(n) \quad = T(n/2^1) + 5$

$T(n/2^1) = T(n/2^2) + 5$

$T(n/2^2) = T(n/2^3) + 5$

$T(n/2^3) = T(n/2^4) + 5$

$\vdots$

$T(n/2^{L-1}) = T(n/2^L) + 5$

$n/2l^L = 1$

$n = 2^L$

$L = \log_2 n$

$T(n) = (T(1) + 5)*\log_2 n$

$\underline{T(n) = 6*\log_2 n}$ $\therefore$ Complexidade de Tempo = **O(log n)**

Para a complexidade de espaço, basta perceber que a pilha será incrementada <u>1</u> vez a cada chamada recursiva, portanto, também será **O(log n)**.

3) Resolva as relações de recorrência:

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

b) $T(n) = 2T(n-1) + n$
$T(1) = 1$

c) $T(n) = 4T(n/2) + n$
$T(1) = 1$

d) $T(n) = T(n/2) + \log 2n$
$T(1) = 1$

Resposta:
a)
$T(n) = T(n/2) + n$
$T(n/2) = T(n/2^2) + n/2 + n$
$T(n/2^2) = T(n/2^3) + n/2^2 + n/2 + n$
$T(n/2^3) = T(n/2^4) + n/2^3 + n/2^2 + n/2 + n$
$\vdots$
$T(n/2^{L-1}) = T(n/2^L) + n/2^{L-1} + \ldots + n/2^3 + n/2^2 + n/2 + n$
$n/2^L = 1 \rightarrow$ Base
$n = 2^L$
$L = \log_2 n$
$T(n) = T(1) + n/2^{L-1} + \ldots + n/2^3 + n/2^2 + n/2 + n = 1 + n(1 + 1)$
$T(n) = 1 + 2n$

b)
$T(n) = 2T(n - 1) + n$
$T(1) = 1$

$T(n) = 2T(n - 1) + n-1$
$2T(n - 1) = 2^2T(n - 2) + 2(n - 1)$
$2^2T(n - 2) = 2^3T(n - 3) + 2^2(n - 2)$
$2^3T(n - 3) = 2^4T(n - 4) + 2^3(n - 3)$
$\vdots$
$2^{L-1}T(n - (L - 1) = 2^LT(n - L) + 2^{L-1}(n -(L - 1))$
$n-L = 1 \rightarrow$ Base
$n = L + 1$
$L = n - 1$
$T(n) = n + 2(n - 1) + 2^2(n - 2) + 2^3(n - 3) + ... + 2^L -1(2) + 2^L 1$

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

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

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

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

$T(n) = n(2^n - 1) - (n.2^n - 3.2^2 + 4)/2$

$T(n) = 2^{n+1} - n - n.2^{n-1} - 3.2^{2-1} + 2$

$\therefore$

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

c)

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

$T(1) = 1$

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

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

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

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

$\vdots$

$4^{L-1}T(n/2^{L-1}) = 4^LT(n/2^L) + 4^{L-1}n/2^{L-1}$

$n/2^L = 1 \rightarrow$ Base

$n = 2^L$

$L = \log_2 n$

$T(n) = n + 4n/2 + 4^2/n2 + 4^3/n3 + ... + 4^L - 1/2^L + 4^L.1$

$T(n) = 4^0n/2^0 + 4n/2 + 4^2/n2 + 4^3/n3 + ... + 4^L - 1/2^L + 4^L.n/2^L$

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

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

$T(n) = n. \sum_{i=0}^{L} 2^i = n(2^{L+1} - 1) = n(2.2^L - 1) = n(2n - 1) = 2n^2 - n$

$T(n) = 2n^2 - n$

d)

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

$T(1) = 1$

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

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

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

$\vdots$

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

$n/2^L = 1 \rightarrow$ Base

$n = 2^L$

$L = \log_2 n$

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

$T(n) = \log_2 n + (\log_2 n - \log_2 2) + (\log_2 n - \log_2 2^2) + \ldots + (\log_2 n - \log_2 2^{L-1}) + 1$
$T(n) = \log_2 n * L - (1 + 2 + 3 + \ldots + L - 1) + 1$
$T(n) = \log_2 n * \log_2 n - (1 + 2 + 3 + \ldots + L - 1) + 1$
$T(n) = \log^2_2 n - \sum_{i=0}^{L-1} i + 1$
$T(n) = \log^2_2 n - ((L - 1)*(L - 1 + 1))/2 + 1$
$T(n) = \log^2_2 n - ((L - 1) * L)/2 + 1$
$T(n) = \log^2_2 n - (L^2 - L)/2 + 1$
$T(n) = \log^2_2 n - (\log^2_2 n - \log_2 n)/2 + 1$

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 $k \geq 1$.

Resposta:

$T(n) = 4T(n/2) + n$
$T(1) = 1$
é
$T(n) = 2n^2 – n$, para n = 2 k e $k \geq 1$.

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:
$T(2^{k+1})^{k+1} = 4T(2^k) + 2^{k+1}$
$T(2^{k+1}) = 4(2(2^k)^2 - 2^k) + 2^{k+1}$
$2(2^{k+1})^2 - 2^{k+1} = 4(2(2^k)^2 - 2^k) + 2^{k+1}$
$2(2.2^k)^2 - 2^{k+1} = 4(2.2^{2k} - 2^k) + 2^{k+1}$
$\cancel{2.2^{2k}.2^2} - 2^{k+1} = \cancel{4.2.2^{2k}} - 4.2^k + 2^{k+1}$
$- 2^{k+1} = - 4.2^k + 2^{k+1}$
$- 2.2^k = - 4.2^k + 2.2^k$
$2.2^k = 2.2^k$ **C.Q.D.**

5) Considerando a estrutura de dados ***Lista*** que representa uma lista encadeada, implemente uma função que insira no início da lista um elemento inteiro passado como parâmetro uma função que insira o elemento ao final da lista. Qual a complexidade de tempo de cada versão? Explique sua resposta.

```
struct Lista
{
  int elem;
  struct Lista*ptr;
};
```

Resposta:
Implementação no arquivo **.c**

Inserção no início: Complexidade **O(1)**, pois sempre é constante, não há presença de loops e a operação é geral independente do tamanho da lista, ou seja, ela independe de **n**.

Inserção na cauda: Complexidade **O(n)**, pois sempre terá que percorrer toda lista em um loop até **n** devido a natureza da lista simplesmente encadeada.

6) Considerando uma árvore binária de pesquisa balanceada representada pela estrutura de dados declarada abaixo, mostre a relação de recorrência que descreve a complexidade de tempo para a função ***imprimir***. Resolva essa relação de recorrência. Mostre a complexidade de tempo e espaço para essa função.

```
struct Arvore
{
  int elem;
  struct Arvore *esq, *dir;
};
void imprimir(struct Arvore *r)
{
  if (r != NULL)
  {
    imprimir (r->esq);
    printf("%d ", r->elem);
    imprimir (r->dir);
  }
}
```

Reescreva a função ***imprimir*** sem usar recursividade.

Resposta:
Para calcular a **complexidade de tempo** devemos fazer:
$\mathbf{T(n) = 2T(n/2) + 1}$
$\mathbf{T(1) = 1}$

$2T(n/2) = 2^2T(n/2^2) + 1$
$2^2T(n/2^2) = 2^2T(n/2^3) + 1$
$\vdots$
$2^{L-1}T(n/2^{L-1}) = 2^{L-1}T(n/2^L) + 1$
$n/2^L = 1 \rightarrow$ Base
$n = 2^L$
$L = \log_2 n$
$T(n) = 1 + L$
$T(n) = 1 + \log_2 n = \log_2 n + 1 \quad \therefore$ Complexidade de tempo = **O(log n)**

Para a complexidade de espaço, basta perceber que a altura da árvore será $\log_2 n$, portanto, também será **O(log n)**.

Implementação no arquivo **.c**