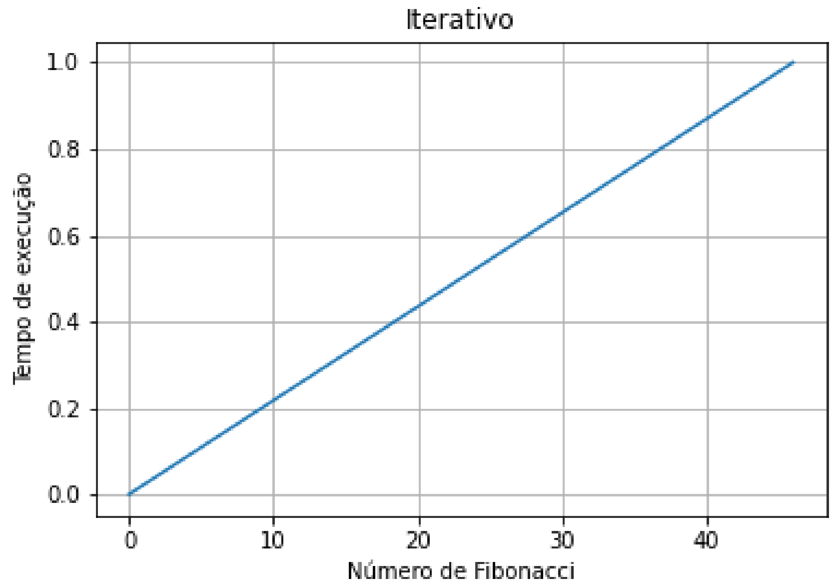
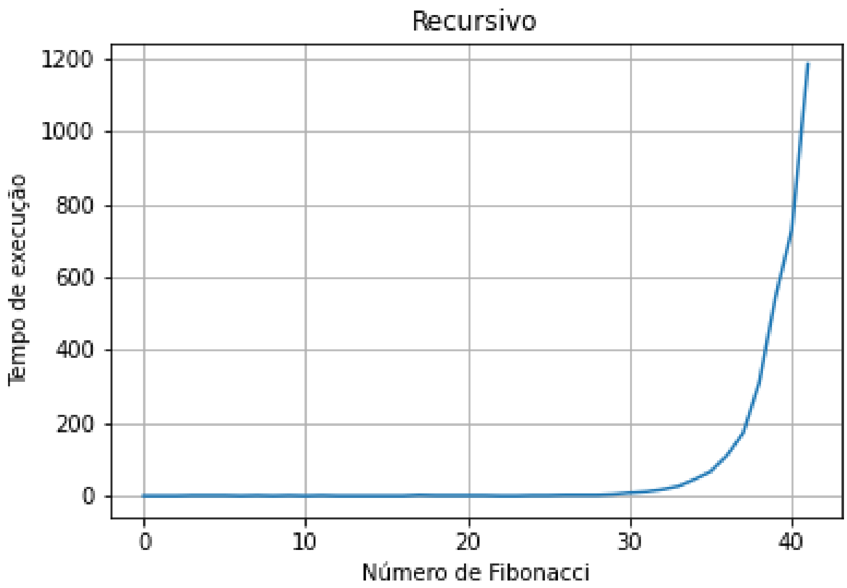
Aluno: Andrei Antonio Villa
Lista 1
Disciplina: Complexidade de Algoritmo
Data: 03/11/2021

Exercício 1:

Iterativo:



Recursivo:



Analise: O código iterativo possui complexidade linear enquanto o código recursivo possui complexidade exponencial. Ambos os algoritmos foram executados para criar 2 vetores um com o tempo de execução e o outro com o

número da de sequência de Fibonacci. Na execução do algoritmo recursivo foi dado o limite de um segundo de execução que foi alcançado no valor 41.
No algoritmo iterativo todas as execuções aconteceram praticamente instantaneamente até o limite de inteiro.
Em ambos os casos foram verificados distorções no tempo de execução de alguns valores porem isso pode ser atribuído a características da maquina ou da própria linguagem.

Exercício 2:

```
int Potencia(int b, int e){
    int r;
    if(e==0)            } O(1)
        return 1;
    r=potencia(b, e/2)    T(n/2)
    if(e%2==0)
        return r*r;     } O(1)
    else
        return r*r*b;
}
```

$T(n/2)+1$

$T(1)=1$: ultima recursão possui complexidade 1

$T(n/2)+1$

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

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

logo $T(n/2^k)+K$

analisando $\frac{n}{2^k}=1$

$n=2^k$

$K=\log_2(n)$ então $T(n)=\log_2(n)+1$

Complexidade de espaço: O(n)

Exercício 3:

3a) $T(\frac{n}{2}) + n$

$T(1) = 1$

relação entre K e n

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2(n)$

Substituindo:

$$T(n) = T(\tfrac{n}{2}) + n$$

$$T(n) = T(\tfrac{n}{2^2}) + n + \tfrac{n}{2}$$

$$T(n) = T(\tfrac{n}{2^3}) + n + \tfrac{n}{2} + \tfrac{n}{4}$$

ou seja:

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

$$\boxed{S_n + a_{n+1} = a_0 + \sum_{0}^{n} a_{i+1}}$$

↳ formula da Perturbação.

$$\sum_{i=0}^{k-1} + \tfrac{1}{2^k} = 1 + \tfrac{1}{2}\sum_{i=0}^{k-1} \tfrac{1}{2^i}$$

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

$$\sum_{i=0}^{k-1} = 2(1 - \tfrac{1}{2^k})$$

$$\sum_{i=0}^{k-1} = 2 - \tfrac{2}{2^k} \quad , n = 2^k$$

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

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

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

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

3b) $T(n) = 2T(n-1) + n$ Substituindo:

$T(1) = 1$ $\rightarrow T(n) = 2T(n-1) + n$

$T(n) = 4T(n-2) + 2(n-1) + n$

relação K e n $T(n) = 8T(n-3) + 4(n-2) + 2(n-1) + n$

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

$n - 1 - K = 1$

$n = K + 2$ ou $K = n - 2$

Separando o somatório

$n\sum_{i=0}^{K} 2^i - \sum_{i=0}^{K} i2^i$ Perturbações

$\mathrm{I} \rightarrow \sum_{i=0}^{K} 2^i + 2^{K+1} = 1 + \sum_{i=0}^{K} 2^{i+1}$

$\mathrm{II} \rightarrow \sum_{i=0}^{K} i2^i + (K+1)2^{(K+1)} = \sum_{i=0}^{K} (i+1)2^{(i+1)} + 0$

$\sum_{i=0}^{K} 2^i + 2^{K+1} = 1 + 2\sum_{i=0}^{K} 2^i$

$\sum_{i=0}^{K} i2^i + (K+1)2^{(K+1)} = 2\sum_{i=0}^{K} i2^i + 2\sum_{i=0}^{K} 2^i$

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

$\sum_{i=0}^{K} i2^i - 2\sum_{i=0}^{K} i2^i = -(K+1)2^{(K+1)} + 2(2^{K+1} - 1)$

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

$\sum_{i=0}^{K} i2^i = (K+1)2^{(K+1)} - 2^{K+2} + 2$

retomando o somatório

$\sum_{i=0}^{K} 2^i(n-i) = n[2^{K+1} - 1] - [(K+1)2^{K+1} - 2^{K+2} + 2]$

$\sum_{i=0}^{K} 2^i(n-i) = n2^{n-1} - n - (n-1)2^{n-1} + 2^n - 2$

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

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

$\sum_{i=0}^{K} 2^i(n-i) = (3) \cdot 2^{n-1} - n - 2$

Substituindo em $T(n)$:

$T(n) = 2^{n-1} + (3) \cdot 2^{n-1} - n - 2$

$T(n) = (4)2^{n-1} - n - 2$

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

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

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

$T(1) = 1$

$n/2^k = 1$

$n = 2^k$

$k = \log_2(n)$

Substituindo:

$4T(n/2) + n$

$4 \cdot 4T(n/2^2) + 2n + n$

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

logo:

$$T(n) = 4^kT(n/2^k) + \sum_{i=0}^{k-1} 2^i n$$

ou

$$T(n) = 4^kT(n/2^k) + n\boxed{\sum_{i=0}^{k-1} 2i}$$

→ resolvido na 3b

$T(n) = 4^{\log_2(n)} + n(n-1)$.

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

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

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

$T(1)$

$n/2^{k+1} = 1$

$n = 2^{k+1}$

$k+1 = \log n$

$k = \log n - 1$

substituindo:

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

$T(n) = [T(n/2^2) + \log(n/2)] + \log n$

$T(n) = [T(n/2^3) + \log(n/2^2)] + \log n/2 + \log n$

logo:

$$T(n) = T(n/2^{k+1}) + (k+1)\log_2 n - \sum_{i=0}^{k} i$$

$$\sum_{i=0}^{k} i = \frac{k\cdot(1+k)}{2}$$

$$T(n) = T(1) + \log_2 n - \frac{(\log_2 n - 1)\log_2 n}{2}$$

$$T(n) = T(1) + \tfrac{1}{2}(\log_2^2 n + \log_2(n))$$

Exercício 4:

4)

$T(n) = 4T(n/2) + n$ $\Big\}$ $T(n) = 2n^2 - n$, $n = 2^k$ e $k \geq 1$

$T(1) = 1$

Hipótese

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

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

Base $(k = 0)$

$T(n) \stackrel{?}{=} 2.(2^0) - 2^0$

$T(n) = 2 - 1$

$T(n) \stackrel{?}{=} 1$

$1 = 1$ provado

Passo indutivo

$T(2^{k+1}) = 4T(2^{k+1}/2) + 2^{k+1}$

$T(2^{k+1}) = 4T(2^k) + 2^{k+1}$

$T(2^{k+1}) = 4(2(2^k)^2 - 2^k) + 2^{k+1}$

$2(2^{k+1})^2 - 2^{k+1} = 4(2(2^k)^2 - 2^k) + 2^{k+1}$

$2^{2x+3} - 2^{k+1} = 2^2(2^{(k+1)^2} - 2^k) + 2^{k+1}$

$2^{2x+3} - 2^{k+1} = 2^2(2^{2x+1} - 2^k) + 2^{k+1}$

$2^{2x+3} - 2^{k+1} = 2^{2x+3} - 2^{x+2} + 2^{k+1}$

$2^{2x+3} - 2^{k+1} = 2^{2x+3} - 2^{x+1}$ provado

Exercício 5:

Inserir no início:

```
Lista *insereInicio(Lista *lista, int valor){
    Lista *insere = (Lista*) malloc(sizeof(Lista));
    insere->elem = valor;
    insere->ptr = lista;
    return insere;
}
```

Todas as operações realizadas pela função possuem complexidade O(1) logo o somatório dessas complexidades e por consequência a função possui complexidade de tempo O(1).

Insere no final:

```c
Lista *insereFinal(Lista *lista, int valor){
    Lista *insere = (Lista*) malloc(sizeof(Lista));
    Lista *percorre = lista;
    insere->elem = valor;
    insere->ptr = NULL;

    if(lista == NULL)
        return insere;

    while(percorre->ptr != NULL)
        percorre = percorre->ptr;

    printf("%d\n", percorre->elem);
    percorre->ptr = insere;

    return lista;
}
```

Ao contrário da função de inserir no inicio a função insere no final possui um loop while, esse loop percorre a lista inteira ou seja possui complexidade de O(n) enquanto as demais instruções possuem complexidade O(1) sendo assim a complexidade da função é de O(n)

Exercício 6:

```
6) void imprimir(Arvore n){
    if(n!=Null){                  O(1)
        imprimir(n->esq);         T(n/2)
        printf("%d", n->elem);    O(1)
        imprimir(n->dir);         T(n/2)
    }
}
```

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

$2T(n/2)+1$

$4T(n/2^2)+2$ portanto $2^kT(n/2^k)+k$

$8T(n/2^3)+3$

$\frac{n}{2^k}=1$

$n=2^k$

$k=\log_2(n)$

$2^{\log_2(n)}T(n/2^{\log_2 n})+\log_2(n)$

$nT(1)+\log_2(n)$

$n+\log n$

Complexidade de Tempo $= n+\log n$

Complexidade de espaço $= O(n)$ onde n é referente ao tamanho da arvore

```c
void imprimeIterativo(Arvore *a){
    Pilha *p = pilhaCria();
    push(p, a);
    while(!pilhaVazia(p)){
        a = pop(p);
        printf("%d", a->elem);
        if(a->dir != NULL)
            push(p,a->dir);
        if(a->esq != NULL)
            push(p, a->esq);
    }
}
```