FUNDAÇÃO GETÚLIO VARGAS

ESCOLA DE MATEMÁTICA APLICADA

MESTRADO 2015.1

ESTRUTURA DE DADOS E SEUS ALGORITMOS

Prof Alexandre Rademaker

Resolução dos Exercícios Selecionandos dos Capítulos 1 e 2

GRUPO:

KIZZY TERRA

GUSTAVO AVILA

CAROLINE FIALHO

RIO DE JANEIRO

MARÇO DE 2015

1 Implementação dos algoritmos discutidos em sala

Os algoritmos a seguir foram implementados em Python:

Implementação 1 Multiplicação a la Francais

```
def multiply1(x, y): # y>= 0
    if y == 0:
        return 0
    else:
        z = multiply1(x, y//2)
        if y%2 == 0:
            return 2*z
        else:
            return x + 2*z
```

Implementação 2 Multiplicação Tradicional

```
def shiftright(x, s):
    while s > 0:
        x *=10
        s = s-1
    return x


def multiply(x, y):
    if x == 0 or y == 1:
        return x
    if y == 0 or x == 1:
        return y
    else:
        x_digits = str(x)
        y_digits = str(y)
        n_x = 0
        n_y = 0
        res = 0
        for n_y in range(0, len(y_digits)):
            res_partial = 0
            for n_x in range(0, len(x_digits)):
                current = shiftright(int(y_digits[len(y_digits) - 1 - n_y])*int(x_digits[len(x_digits) - 1 - n_x]), n_x + n_y)
                res_partial += current
                n_x +=1
            res += res_partial
            n_y += 1
        return res
```

Implementação 3 Karatsuba

```
def splitNumber(x):
    x_digits = str(x)
    x_numberOfDigits = len(x_digits)
    p = int( x_digits[:x_numberOfDigits/2])
    q = int( x_digits[x_numberOfDigits/2:])
    return (x_numberOfDigits, p, q)

def karatsuba(x, y):
    if x<10 and y<10:
        return x*y
    else:
        (x_numberOfDigits, p ,q ) = splitNumber(x)
        (y_numberOfDigits, r, s) = splitNumber(y)
        if x_numberOfDigits%2 == 0:
            n = x_numberOfDigits
        else:
            n = x_numberOfDigits + 1
        i = karatsuba(p, r)
        j = karatsuba(q, s)
        k = karatsuba( (p+q), (r+s))
        res = shiftright(i, n) + j + shiftright(k - i - j, n/2)
        return res
```

2 Implementações MergeSort

Os algoritmos a seguir foram implementados em Python:

Implementação 4 Merge (função auxilar para o MergeSort)

```
def merge(x, y):
    if len(x) == 0:
        return y
    if len(y) == 0:
        return x
    if x[0] <= y[0]:
        return [x[0]] + merge(x[1:], y)
    else:
        return [y[0]] + merge(x, y[1:])
```

Implementação 5 MergeSort Iterativo

```
import Queue
    def mergesort_iterativo(x):
    q = Queue.Queue()
    for i in range(0, len(x)):
        q.put([x[i]])
        i = i + 1
    while q.qsize() > 1:
        q.put(merge(q.get(), q.get()))
    return q.get()
```

Implementação 6 MergeSort Recursivo

```
def mergesort(x):
  if len(x) > 1:
    return merge(mergesort(x[0:len(x)/2]), mergesort(x[len(x)/2:len(x)]))
  else:
    return x
```

3 Pilhas e filas com listas encadeadas ou arrays

Para a resolução das questões a seguir considera-se os seguintes conceitos:

Lista encadeada: É uma estrutura dinâmica composta por células que apontam para o próximo elemento da lista. A lista encadeada é acessada através de seu primerio elemento e seu último elemento aponta para um elemento nulo. Os custos das operações básicas nesta estrutura são:

1) Inserção: O(1)
2) Remoção: O(n)
3) Acesso aleatório: O(n)

Array: É uma estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, um índice ou uma chave. Os custos das operações básicas nesta estrutura são

1) Inserção: O(n)
2) Remoção: O(n)
3) Acesso aleatório: O(1)

3.1 Pilhas utilizando Lista Encadeada

Uma vez que em uma estrutura de pilha temos a propriedade "Last In, First Out" implementá-la utilizando lista encadeada tem a vantagem de realizar as operações POP e PUSH em O(1); visto que em uma lista encadeada as operações de inserção e a remoção na "cabeça" da lista tem cutso constante.

3.2 Pilhas utilizando Arrays

Se utilizarmos um array para implementar uma pilha de forma a inserir e remover os elementos da pilha no início do array, as operações POP e PUSH terão custo O(n) opção dsvantajosa em relação a implementação utilizando lista encadeada. Entretanto, é possível implementar a pilha inserindo e removendo os elementos no final do array, para isso basta possuir uma variável a qual armzena a informação da posição do último elemento e assim, as operações de inserção (PUSH) e remoção(POP) na pilha passam a ter um custo constante O(1).

3.3 Filas utilizando Lista Encadeada

As filas possuem a propriedade "First In, First Out", por essa razão o custo de remoção (POP) em uma fila implementada utilizando-se lista encadeada é O(n), dado que é preciso percorrer toda a lista para remover o último elemento. A operação de inserção (PUSH), por sua vez, tem custo O(1).

3.4 Filas utilizando Arrays

Considerando uma implementação de fila utilizando array de forma a inserir os elementos da fila no início do array e remover os elementos do final, a operação de inserção (PUSH) terá custo O(n), ao passo que a operação de remoção (POP) terá custo constante O(1).

4 Exercício 2.1

$$x.y = 2^n x_L.y_L + 2^{n/2}(x_L.y_R + x_R.y_L) + x_R.y_R$$

$$x = 10011011 \ , \ y = 10111010 \ e n = 8 :$$

$$x.y = 2^8(1001).(1011) + 2^{n/2}((1001).(1010) + (1011).(1011)) + (1011).(1010)$$

$$x.y = 2^8(1001).(1011) + 2^4[(1001 + 1011).(1011 + 1010) - (1001).(1011) - (1011).(1010)] + (1011).(1010)$$

$$x.y = 2^8(\mathbf{1001}).(\mathbf{1011}) + 2^4[(\mathbf{10100}).(\mathbf{10101}) - (1001).(1011) - (1011).(1010)] + (\mathbf{1011}).(\mathbf{1010})$$

1) Calculando $(\mathbf{1001}).(\mathbf{1011})$:

$(1001).(1011) = 2^4(10).(10) + 2^2[(10 + 01).(10 + 11) - (10).(10) - (01).(11)] + (01).(11) = 2^4(\mathbf{10}).(\mathbf{10}) + 2^2[(\mathbf{11}).(\mathbf{101}) - (10).(10) - (01).(11)] + (\mathbf{01}).(\mathbf{11})$

1.1) Calculando $(\mathbf{10}).(\mathbf{10})$:

$(10).(10) = 2^2(1).(1)+2[(1+0).(1+0)-(1).(1)-(0).(0)]+(0).(0) = 2^2.1+2.(1-1-0)+0 = 100 + 2.0 + 0 = 100$

1.2) Calculando $(\mathbf{01}).(\mathbf{11})$:

$(01).(11) = 2^2(0).(1)+2[(0+1).(1+1)-(0).(1)-(1).(1)]+(1).(1) = 2^2.0+2.(10-0-1)+1 = 2.(1) + 1 = 10 + 1 = 11$

1.3) Calculando $(\mathbf{11}).(\mathbf{101})$:

$(0011).(0101) = 2^4(00).(01) + 2^2[(00 + 11).(01 + 01) - (00).(01) - (11).(01)] + (11).(01) = 2^4(\mathbf{00}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{10}) - (00).(01) - (11).(01)] + (\mathbf{11}).(\mathbf{01})$

1.3.1) Calculando $(\mathbf{00}).(\mathbf{01})$:

$(00).(01) = 2^2(0).(0) + 2[(0 + 0).(0 + 1) - (0).(0) - (0).(1)] + (0).(1) = 2^2.0 + 2[(0).(1) - 0 - 0] + 0 = 0$

1.3.2) Calculando $(\mathbf{11}).(\mathbf{01})$:

$(11).(01) = 2^2(1).(0)+2[(1+1).(0+1))-(1).(0)-(1).(1)]+(1).(1) = 2^2.0+2[(\mathbf{10}).(\mathbf{01})-0-1]+1 = 2.(1) + 1 = 11$

1.3.3) Calculando $(\mathbf{11}).(\mathbf{10})$:

$(11).(10) = 2^2(1).(1) + 2[(\mathbf{10}).(\mathbf{01}) - (1).(1) - (1).(0)] + (1).(0)$

1.3.3.1) Calculando $(\mathbf{10}).(\mathbf{01})$:

$(10).(01) = 2^2(1).(0)+2[(1+0).(0+1)-(1).(0)-(0).(1)]+(0).(1) = 2^2.0+2[(1).(1)-0-0]+0 = 10$

$(11).(10) = 2^2(1).(1)+2[(1+1).(1+0)-(1).(1)-(1).(0)]+(1).(0) = 2^2.1+2[10-1-0]+0 = 100 + 10 = 110$

$(0011).(0101) = 2^4(\mathbf{00}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{10}) - (00).(01) - (11).(01)] + (\mathbf{11}).(\mathbf{01}) = 2^4.0 + 2^2[110 - 0 - 11] + 11 = 1100 + 11 = 1111$

$(1001).(1011) = 2^4(\mathbf{10}).(\mathbf{10}) + 2^2[(\mathbf{11}).(\mathbf{101}) - (10).(10) - (01).(11)] + (\mathbf{01}).(\mathbf{11}) = 2^4.(100) + 2^2[1111 - 100 - 11] + 11 = 1000000 + 2^2(1000) + 11 = 1000000 + 100000 + 11 = 1100011$

2) Calculando $(\mathbf{1011}).(\mathbf{1010})$:

$(1011).(1010) = 2^4(10).(10) + 2^2[(10 + 11).(10 + 10) - (10).(10) - (11).(10)] + (11).(10) = 2^4(\mathbf{10}).(\mathbf{10}) + 2^2[(\mathbf{101}).(\mathbf{100}) - (10).(10) - (11).(10)] + (\mathbf{11}).(\mathbf{10})$

2.1) Calculando $(\mathbf{10}).(\mathbf{10})$:

$(10).(10) = 2^2(1).(1)+2[(1+0).(1+0)-(1).(1)-(0).(0)]+(0).(0) = 2^2.1+2.(1-1-0)+0 = 100 + 2.0 + 0 = 100$

2.2) Calculando $(\mathbf{11}).(\mathbf{10})$:

$(11).(10) = 2^2(1).(1) + 2[(\mathbf{10}).(\mathbf{01}) - (1).(1) - (1).(0)] + (1).(0)$

2.2.1) Calculando $(\mathbf{10}).(\mathbf{01})$:

$(10).(01) = 2^2(1).(0)+2[(1+0).(0+1)-(1).(0)-(0).(1)]+(0).(1) = 2^2.0+2[(1).(1)-0-0]+0 = 10$

$(11).(10) = 2^2(1).(1)+2[(1+1).(1+0)-(1).(1)-(1).(0)]+(1).(0) = 2^2.1+2[10-1-0]+0 = 100 + 10 = 110$

2.3) Calculando $(\mathbf{0101}).(\mathbf{0100})$:

$(0101).(0100) = 2^4(01).(01) + 2^2[(01 + 01).(01 + 00) - (01).(01) - (01).(00)] + (01).(00) = 2^4(\mathbf{01}).(\mathbf{01}) + 2^2[(\mathbf{10}).(\mathbf{01}) - (01).(01) - (01).(00)] + (\mathbf{01}).(\mathbf{00})$

2.3.1) Calculando $(\mathbf{01}).(\mathbf{01})$:

$(01).(01) = 2^2(0).(0) + 2[(0 + 1).(0 + 1) - (0).(0) - (1).(1)] + (1).(1) = 2^2.0 + 2[(1).(1) - 0 - 1] + 1 = 1$

2.3.2) Calculando $(\mathbf{01}).(\mathbf{00})$:

$(01).(00) = 2^2(0).(0) + 2[(0+1).(0+0) - (0).(0) - (1).(0)] + (1).(0) = 2^2.0 + 2[(1).(0) - 0 - 0] + 0 = 0$

2.3.3) Calculando $(\mathbf{10}).(\mathbf{01})$:

$(10).(01) = 2^2(1).(0) + 2[(1+0).(0+1) - (1).(0) - (0).(1)] + (0).(1) = 2^2.0 + 2[(1).(1) - 0 - 0] + 0 = 10$

$(0101).(0100) = 2^4(\mathbf{01}).(\mathbf{01}) + 2^2[(\mathbf{10}).(\mathbf{01}) - (01).(01) - (01).(00)] + (\mathbf{01}).(\mathbf{00}) = 2^4(1) + 2^2[(10) - (1) - (0)] + (00) = 10000 + 100 = 10100$

$(1011).(1010) = 2^4(\mathbf{10}).(\mathbf{10}) + 2^2[(\mathbf{101}).(\mathbf{100}) - (10).(10) - (11).(10)] + (\mathbf{11}).(\mathbf{10}) = 2^4.(100) + 2^2[10100 - 100 - 110] + 110 = 1000000 + 101000 + 110 = 1101110$

3) Calculando $(\mathbf{010100}).(\mathbf{010101})$:

$(010100).(010101) = 2^6(010).(010) + 2^3[(010+100).(010+101) - (010).(010) - (100).(101)] + (100).(101) = 2^4(\mathbf{10}).(\mathbf{10}) + 2^2[(\mathbf{110}).(\mathbf{111}) - (10).(10) - (100).1(01)] + (\mathbf{100}).(\mathbf{101})$

3.1) Calculando $(\mathbf{10}).(\mathbf{10})$:

$(10).(10) = 2^2(1).(1) + 2[(1+0).(1+0) - (1).(1) - (0).(0)] + (0).(0) = 2^2.1 + 2.(1-1-0) + 0 = 100 + 2.0 + 0 = 100$

3.2) Calculando $(\mathbf{0110}).(\mathbf{0111})$:

$(0110).(0111) = 2^4(01).(01) + 2^2[(01+10).(01+11) - (01).(01) - (10).(11)] + (10).(11) = 2^4(\mathbf{01}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{100}) - (01).(01) - (10).(11)] + (\mathbf{10}).(\mathbf{11})$

3.2.1) Calculando $(\mathbf{01}).(\mathbf{01})$:

$(01).(01) = 2^2(0).(0) + 2[(0+1).(0+1) - (0).(0) - (1).(1)] + (1).(1) = 2^2.0 + 2[(1).(1) - 0 - 1] + 1 = 1$

3.2.2) Calculando $(\mathbf{10}).(\mathbf{11})$:

$(10).(11) = 2^2(1).(1) + 2[(1+0).(1+1)) - (1).(1) - (0).(1)] + (0).(1) = 2^2.1 + 2[(\mathbf{01}).(\mathbf{10}) - 1 - 0] + 0$

3.2.2.1) Calculando $(\mathbf{01}).(\mathbf{10})$:

$(01).(10) = 2^2(0).(1) + 2[(0+1).(1+0) - (0).(1) - (1).(0)] + (1).(0) = 2^2.0 + 2[(1).(1) - 0 - 0] + 0 = 10$

$(10).(11) = 2^2(1).(1) + 2[(1+0).(1+1)) - (1).(1) - (0).(1)] + (0).(1) = 2^2.1 + 2[(\mathbf{01}).(\mathbf{10}) - 1 - 0] + 0 = 100 + 10 = 110$

3.2.3) Calculando $(\mathbf{11}).(\mathbf{100})$:

$(0011).(0100) = 2^4(00).(01) + 2^2[(00+11).(01+00) - (00).(01) - (11).(00)] + (11).(00) = 2^4(\mathbf{00}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{01}) - (00).(01) - (11).(00)] + (\mathbf{11}).(\mathbf{00})$

3.2.3.1) Calculando $(\mathbf{00}).(\mathbf{01})$:

$(00).(01) = 2^2(0).(0) + 2[(0+0).(0+1) - (0).(0) - (0).(1)] + (0).(1) = 2^2.0 + 2[(0).(1) - 0 - 0] + 0 = 0$

3.2.3.2) Calculando $(\mathbf{11}).(\mathbf{00})$:

$(11).(00) = 2^2(1).(0) + 2[(1+1).(0+0)) - (1).(0) - (1).(0)] + (1).(0) = 2^2.0 + 2[(\mathbf{10}).(\mathbf{00}) - 0 - 0] + 0$

3.2.3.2.1) Calculando $(\mathbf{10}).(\mathbf{00})$:

$(10).(00) = 2^2(1).(0) + 2[(1+0).(0+0) - (1).(0) - (0).(0)] + (0).(0) = 2^2.0 + 2[(1).(0) - 0 - 0] + 0 = 0$

$(11).(00) = 2^2.0 + 2[(\mathbf{10}).(\mathbf{00}) - 0 - 0] + 0 = 2^2.0 + 2[0 - 0 - 0] + 0 = 0$

3.2.3.3) Calculando $(\mathbf{11}).(\mathbf{01})$:

$(11).(01) = 2^2(1).(0) + 2[(1+1).(0+1)) - (1).(0) - (1).(1)] + (1).(1) = 2^2.0 + 2[(10).(1) - 0 - 1] + 1 = 2.(1) + 1 = 11$

$(0011).(0100) = 2^4(\mathbf{00}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{01}) - (00).(01) - (11).(00)] + (\mathbf{11}).(\mathbf{00}) = 2^4.0 + 2^2[11 - 0 - 0] + 0 = 1100$

$(0110).(0111) = 2^4(\mathbf{01}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{100}) - (01).(01) - (10).(11)] + (\mathbf{10}).(\mathbf{11}) = 2^4(1) + 2^2[(1100) - (1) - (110)] + (110) = 10000 + 10100 + 110 = 101010$

3.3) Calculando $(\mathbf{0100}).(\mathbf{0101})$:

$(0100).(0101) = 2^4(01).(01) + 2^2[(01+00).(01+01) - (01).(01) - (00).(01)] + (00).(01) = 2^4(1) + 2^2[(\mathbf{01}).(\mathbf{10}) - (01).(01) - (00).(01)] + (\mathbf{00}).(\mathbf{01})$

3.3.1) Calculando $(\mathbf{01}).(\mathbf{01})$:

$(01).(01) = 2^2(0).(0) + 2[(0+1).(0+1) - (0).(0) - (1).(1)] + (1).(1) = 2^2.0 + 2[(1).(1) - 0 - 1] + 1 = 1$

3.3.2) Calculando $(\mathbf{00}).(\mathbf{01})$:

$(00).(01) = 2^2(0).(0) + 2[(0+0).(0+1) - (0).(0) - (0).(1)] + (0).(1) = 2^2.0 + 2[(0).(1) - 0 - 0] + 0 = 0$

3.3.3) Calculando $(\mathbf{01}).(\mathbf{10})$:

$(01).(10) = 2^2(0).(1) + 2[(0+1).(1+0) - (0).(1) - (1).(0)] + (1).(0) = 2^2.0 + 2[(1).(1) - 0 - 0] + 0 = 10$

$(0011).(0100) = 2^4(\mathbf{00}).(\mathbf{01}) + 2^2[(\mathbf{11}).(\mathbf{01}) - (00).(01) - (11).(00)] + (\mathbf{11}).(\mathbf{00}) = 2^4.0 + 2^2[11 - 0 - 0] + 0 = 1100$

$(0100).(0101) = 2^4(\mathbf{01}).(\mathbf{01}) + 2^2[(\mathbf{01}).(\mathbf{10}) - (01).(01) - (00).(01)] + (\mathbf{00}).(\mathbf{01}) = 2^4(1) + 2^2[10 - 1 - 0] + 0 = 10000 + 100 = 10100$

$(010100).(010101) = 2^6(\mathbf{10}).(\mathbf{10}) + 2^3[(\mathbf{110}).(\mathbf{111}) - (10).(10) - (100).(101)] + (\mathbf{100}).(\mathbf{101}) = 2^6(100) + 2^3[101010 - 100 - 10100] + 10100 = 100000000 + 10010000 + 10100 = 110100100$

$$x.y = 2^8(1001).(1011) + 2^4[(10100).(10101) - (1001).(1011) - (1011).(1010)] + (1011).(1010) =$$
$$2^8(1100011) + 2^4[110100100 - 1100011 - 1101110] + 1101110 =$$
$$110001100000000 + 110100110000 + 1101110$$

$$x.y = 111000010011110\square$$

5 Exercício 2.3

a) Resolvendo a recorrência como se pede:

$T(n) = 3T(n/2) + O(n)$

$T(n) \leq 3T(n/2) + cn$

$T(n) \leq 3[3T(n/4) + cn/2] + cn = 9T(n/4) + 5cn/2$

$T(n) \leq 9[3T(n/8) + cn/4] + 5cn/2 = 27T(n/8) + 19cn/4$

$T(n) \leq 27[3T(n/16) + cn/8] + 19cn/4 = 81T(n/16) + 65cn/8$

$\vdots$

$T(n) < 3^kT(n/2^k) + 3^kcn/2^{k-1}$

b) É possível resolver mesmo para este caso em que o teorema mestre não funciona:
$T(n) = T(n-1) + O(1)$

$T(n) \leq T(n-1) + c$

$T(n) \leq T(n-2) + 2c$

$T(n) \leq T(n-3) + 3c$

$\vdots$

$T(n) \leq T(n-k) + kc$, para $k = n-1$

$T(n) \leq T(n-(n-1)) + (n-1)c = T(1) + cn - c$

$T(n) = O(n)$

6 Exercício 2.4

Algortimo A: Soluciona o problema dividindo em 5 subproblemas com a metade do tamanho, resolvendo cada subproblema recursivamente e combinando suas soluções em tempo linear.

Algoritmo B: Soluciona o problemade tamanho n resolvendo recursivamente dois subproblemas de tamanho n-1e combinando suas soluções em tempo constante.

Algoritmo C: soluciona problemas de tamanho n dividindo-os em nove subproblemas de tamanho n/3, resolvendo recursivamente cada subproblema e combinando suas soluções em $O(n^2)$.

Analisando cada algortimo temos:

A) $T(n) = 5T(n/2) + O(n)$

Pelo Teorema Mestre $T(n) = O(n^{log_2 5}) = O(n^{2,32})$

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

$T(n) \leq 2T(n-1) + c$

$T(n) \leq 2[2T(n-2) + c] + c = 4T(n-2) + 3c$

$T(n) \leq 4[2T(n-3) + c] + 3c = 8T(n-3) + 7c$

$\vdots$

$T(n) \leq 2^k T(n-k) + 2k - 1$, para k = n-1

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

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

C) $T(n) = 9T(n/3) + O(n^2)$

Pelo Teorema Mestre $T(n) = O(n^2 logn)$

O algortimo B possui complexidade bem maior do que os algoritmos A e C. Entretanto, a diferença entre os algoritmos A e C é mais sútil, por essa razão utilizaremos o WolframAlpha (disponível em http://www.wolframalpha.com) para plotar o gráfico das funções e escolher o melhor algoritmo:

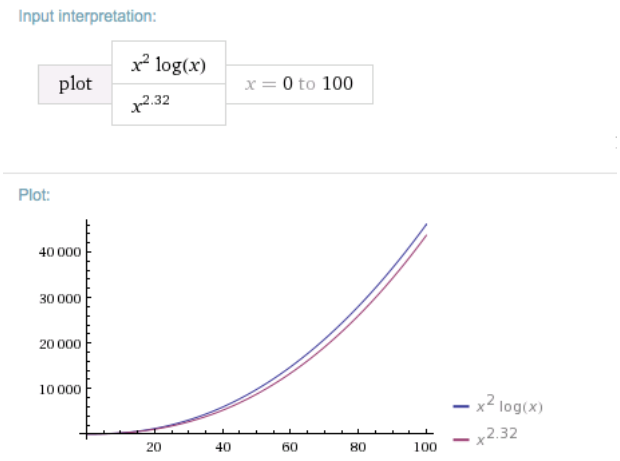
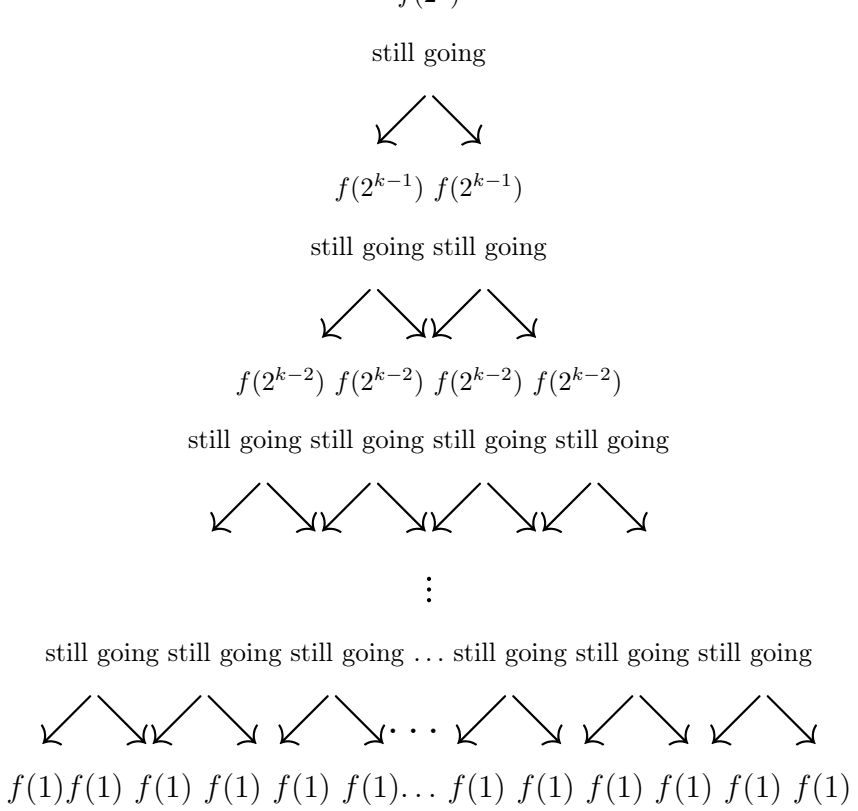


Figura 1: Análise de complexidade dos algoritmos A e C)

Com o auxílio da figura acima podemos observar que o algortimo de menor complexidade o qual deve ser escolhido é o algoritmo A.

7 Exercício 2.12

Analisando a recorrência para n potência de 2:



A partir da observação da árvore de recorrência:
T(1) = 0
T(2) = 1
T(4) = 3 = 2*T(2) + 1
T(8) = 7 = 2*T(4) + 1

⋮

$T(2^k) = 2*T(2^{k-1}) + 1 \Longrightarrow T(n) = 2*T(n/2) + 1 = 2*T(n/2) + O(1) \Longrightarrow$Pelo Teorema Mestre: $d = 0 < log_b a = log_2 2 = 1 \Longrightarrow T(n) = O(n)$

Logo o programa imprime $O(n)$ linhas.

8 Exercício 2.14

A idéia do algoritmo implementado coniste em ordenar os elementos em O(nlogn) e então remover os duplicados em O(n) de forma que o algortimo final terá complexidade O(nlogn). O algortimo pedido foi implementado em Python como segue:

Implementação 7 exercício 2.14

```
def exercicio14(L):
    L_sorted = mergesort(L)
    L_without_duplicates = []
    j=0
    for i in range(0,len(L_sorted)):
        if i == 0:
            L_without_duplicates.insert(j, L_sorted[i])
            i+=1
            j+=1
        else:
            if L_sorted[i] > L_without_duplicates[j-1]:
                L_without_duplicates.insert(j,L_sorted[i])
                i+=1
                j+=1
            else:
                i+=1
    return L_without_duplicates
```

9 Exercício 2.15

Implementação 8 exercício 2.15

```
def splitInPlace(S,v):
    current = 0
    for i in range(0, len(S)):
        if S[i] < v:
            temp = S[i]
            S[i] = S[current]
            S[current] = temp
            current +=1
    for i in range(current, len(S)):
        if S[i] == v:
            temp = S[i]
            S[i] = S[current]
            S[current] = temp
            current +=1
    return S
```

10 Exercício 2.17

Uma vez que os elementos do array são distintos podemos O algoritmo implementado é O(log n) uma vez que T(n) = T(n/2) + O(1)

Implementação 9 exercício 2.17

```
def exercicio17(L):
    n = len(L)
    if n==1:
        if L[0] == 0:
            print 'true'
    else:
        if L[n/2] == n/2:
            print 'true'
        else:
            if L[n/2] > n/2:
                exercicio17(L[:n/2])
            else:
                exercicio17(L[n/2:]-n/2)
```

11 Exercício 2.18

Se um algoritmo de busca em um array ordenado utilizar apenas comparações, então considerando o pior caso o algoritmo realizará a busca em tempo O(n). Uma vez que o número procurado poderá estar na extremidade oposta a extremidade inicial do array e nesse caso será necessário comparar este número com todos os elementos que pertencem ao array.

12 Exercício 2.19

(a) O algoritmo merge implementado no capítulo dois realiza a operação em tempo O(n) uma vez que percorre uma vez cada array, mais precisamente tempo O(m+n) sendo m e n o tamanho dos respectivos arrays para os quais será realizado o merge. Para o caso dados tendo n listas, serão realizados n -1 "merge", logo a complexidade deste algortimo é (n-1)*O(n) = O(n^2).

(b) Uma solução mais eficiente para este problema utilizando a técnica dividir-para-conquistar seria dividir os arrays em dois conjuntos (cada um com k/2 arrays), realizar recursivamente o merge recursivamente nesses conjuntos e finalmente fazer o merge dos dois conjuntos iniciais. O algoritmo foi implementado como segue:

Implementação 10 exercício 2.19

```
def merge(x, y):
    if len(x) == 0:
        return y
    if len(y) == 0:
        return x
    if x[0] <= y[0]:
        return [x[0]] + merge(x[1:], y)
    else:
        return [y[0]] + merge(x, y[1:])

def mergeLists(L):
    n = len(L)
    if len(L) == 0:
        return L
    if len(L) == 1:
        return L[0]
    else:
        return merge(mergeLists(L[:n/2]),mergeLists(L[n/2:]))
```