LISTA 09 - FRANCISCO BRAZ

1-)
a-)
{ - Considerando que nossos arrays começam a partir do index 0.
Entrada: Quantidade de itens do conjunto, representado pelo *n*, vetor contendo os valores de cada item.
Saída: Valor do maior montante de valor. -}

```
função maximo(x, y): inteiro
início
    se x > y então retorna x
    senão retorna y
fim

função calcMontante(n, C): inteiro
início
    M[n+1]
    para i = 0 até n faça
        se i = 0 então M[i] := 0
        senão se i = 1 então M[i] := C[i-1]
        senão
            M[i] := maximo(C[i-1] + M[i-2], M[i-1])
    retorna M[n]
fim
```

b-) As instruções dentro do nosso loop irão executar de i = 0 até n, ou seja:

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

Ignorando a constante, temos que nossa complexidade é O(n).

c-) Implementação no arquivo montanteProgDinamica

2-)

a-)
{ - Considerando que nossos arrays começam a partir do index 0.
Entrada: Tamanho do pão, representado por n, vetor contendo os preços por tamanho.
Saída: Valor do maior montante de valor. -}

```
função maximo(x, y): inteiro
início
    se x > y então retorna x
    senão retorna y
fim

função breadCut(P, n): inteiro
início
    A[n+1]
    para i = 1 até n faça
    início
        q := -1
         para j = 1 até i faça
         início
              q := maximo(q, P[j-1] + A[i-j])
         fim
         A[i] := q
     retorna A[n]
fim
```

b-)

$$\sum_{i=1}^{n}\sum_{j=1}^{i} c = \sum_{i=1}^{n} c * (i - 1 + 1) = \sum_{i=1}^{n} c * i$$

$= (c * 1) + (c * 2) + (c * 3) + ... + (c * n)$ = c * [n(n-1)/2]
= c * [(n^2 -n)/2]

Ignorando as constantes e considerando o termo de maior grau, nossa complexidade seria: O(n^2).

c-) Implementação no arquivo breadCut.c

3-)
OBS: Eu considerei que cada item tinha um peso associado.

a-)
{ - Considerando que nossos arrays começam a partir do index 0.
Entrada: Vetor *V* com os valores associados a cada item, vetor *P* com os pesos associados a cada item, variável *maxCap* para informar a capacidade máxima e variável *n* informando o tamanho de *V* e *P*.
Saída: Valor com a capacidade máxima. -}

```
função maximo(x, y): inteiro
início
    se x > y então retorna x
    senão retorna y
fim

função mochila(V, P, maxCap, n)
início
    M[n+1][maxCap+1]
    para i = 0 até n faça
    início
        para j = 1 até maxCap faça
        início
            se i = 0 ou j = 0 então M[i][j] = 0
            senão se P[i-1] ≤ j então
                M[ i ][ j ] = maximo(V[i-1] + M[ i ][ j - P[i-1] ], M[i-1][ j ]);
            senão M[ i ][ j ] = M[i-1][ j ];
        fim
     fim
     retorna M[n][maxCap]
fim
```

b-) Chamaremos a variável *maxCap* de *m*. Assim:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c = \sum_{i=1}^{n} c * (m - 1 + 1) = \sum_{i=1}^{n} c * m$$
$$c * m * (n - 1 + 1) = c * m * n$$

Ignorando a constante, temos que nossa complexidade é: O(mn)

c-) Implementação no arquivo mochila.c