1. Suponha que cada expressão abaixo represente o tempo *T(n)* consumido por um algoritmo para resolver um problema de tamanho *n*. Escreva os termo(s) dominante(s) para valores muito grandes de *n* e especifique o tempo de execução em notação Θ

Expressão	**Termo(s) Dominante(s)**	**Θ (...)**
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$\Theta(n^3)$
$500n + 100n^{1.5} + 50n\log_{10}(n)$	$100n^{1.5}$	$\Theta(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5n^{1.75}$	$2.5n^{1.75}$	$\Theta\ (n^{1.75})$
$n^2\log_2(n) + n(\log_2(n))^2$	$n^2\log_2(n)$	$\Theta(n^2\log_2(n)$
$n\log_3(n) + n\log_2(n)$	nlog3(n) , nlog2(n)	Θ(nlogn)
$3\log_8(n) + \log_2(\log_2(\log_2(n)))$	$3\log_8(n)$	Θ(logn)
$100n + 0.01n^2$	$0.01n^2$	$\Theta(n^2)$
$0.01n + 100n^2$	$100n^2$	$\Theta(n^2)$
$2n + n^{0.5} + 0.5n^{1,25}$	$0.5n^{1,25}$	$\Theta(5n^{1,25})$
$0.01n\log_2(n) + n(\log_2(n))^2$	$n(\log2(n))^2$	$\Theta(n\log n)^2)$
$100n\log3(n) + n^3 + 100n$	n^3	$\Theta(n^3)$
$0.003\log4(n) + \log_2(\log_2(n))$	0.003log4(n)	Θ(logn)

2. Analise o algoritmo abaixo, escrito em C, que recebe dois vetores, *a* e *b*, de tamanhos iguais *n*. Determine:

```
float f(float* a, float* b, int n) {
 int i, j;
 float s = 0.0;
 for (i=1; i<n; i++) {
      if (a[i]>600) {
           for (j=n-1; j>=0; j--) {
                s += a[i]*b[j];
           }
      } else if (a[i]<300) {
           for (j=n; j<n*n; j+=5) {
                s += a[i]*b[j];
           }
      } else {
           for (j=1; j<n; j=3*j) {
           s += a[i]*b[j];
           }
      }
 }
return s;
}
```

a) qual o tempo de execução do melhor caso em notação Θ.

b) qual o tempo de execução do pior caso em notação Θ

c) quais as condições que o vetor a deve satisfazer para caracterizar o melhor caso.

Nesse programa temos um for externo e três for internos, que rodam dependendo do valor da posição corrente do vetor a seja maior que 600, ou seja a[i] >600, menor que 300, ou seja a[i] <300, ou se 300< a[i] <600.

O loop externo é Θ(n) porque temos exatamente n interações ao longo de sua execução. Veremos a seguir os loops internos

Quando a[i] < 600, pelos valores que a variável j assume é claramente Θ(n).

Quando a[i] <300, temos iterações proporcionais a $n*n = n^2$. Não importa que j seja incrementado de 5 em 5. No máximo, isso fará com quem existam $n^2/5$ iterações, que ainda é $\Theta(n^2)$.

Quando 300=<a[i] <=600, neste loop, a variável j é incrementada em uma progressão geométrica de razão 3 (i.e.: 1, 3, 9, 27, 81, 243...). Pegando um exemplo desta progressão, para $n=243$, temos $log_3 243 = 5$ iterações. Portanto, concluímos que este laço possui limite assintótico Θ(logn).

a) O melhor caso ocorre quando todas as iterações caem na última condição. Este loop é Θ(logn). O loop exterior é Θ(n). Logo, pela regra do produto, o melhor caso é Θ(nlogn).

b) O pior caso ocorre quando todas as iterações caem na segunda condição. Usando um raciocínio análogo ao utilizado no item anterior, concluímos, pela regra do produto, o pior caso é $\Theta(n^3)$

c) Para que o melhor caso ocorra a condição é que 300=< a[i] <=600.

3. Qual o tempo de execução para o pior caso em notação Θ, para o algoritmo abaixo escrito em linguagem C.

```
int f(int n) {
    int i, j, k, sum = 0;
    for ( i=1; i < n; i *= 2 ) {
        for ( j = n; j > 0; j /= 2 ) {
            for ( k = j; k < n; k += 2 ) {
                sum += (i + j * k );
            }
        }
    }
}
```

Temos três loops for aninhado.
O loop externo, a variável i cresce numa PG de razão 2, 1, 2, 4,8,16 ... logo Θ(logn).
O primeiro loop interno a variável j é uma PG de razão ½, n, n/2, n/4, n/8, n/16... logo Θ(logn)
O loop mais interno a variável k cresce numa PA de razão 2, j, j+2, j+4, j+6, até n, logo Θ(n).
Multiplicando os três loops, o pior caso é $\Theta(n(\log n)^2)$

4. Suponha que o vetor *a* contenha *n* valores. Suponha também que a função *randomValue* necessite de um número constante de processamentos para retornar cada valor, e que a função *goodSort* leve um número de etapas computacionais proporcional a *nlogn* para ordenar o vetor. Qual o tempo de execução para o pior caso em notação Θ, para o seguinte fragmento de código, escrito em linguagem C.

```
for ( i = 0; i < n; i++ ) {
    for ( j = 0; j < n; j++ ) {
        a[ j ] = randomValue( i );
    }
    goodSort( a );
}
```

O loop interior demanda um número de processamentos proporcional a *n*, mas a função chamada logo a seguir possui complexidade maior, proporcional a *nlogn*. Pela regra da soma, o tempo de execução da função é dominante sobre o loop. Dado que o loop exterior, que engloba ambos os itens analisados, tem complexidade n, chegamos à conclusão que o pior caso é $\Theta(n^2 logn)$.

5. Utilize uma das técnicas conhecidas de análise de algoritmos recursivos e forneça um limite assintótico θ() para cada algoritmo abaixo, escrito em C:

   a)
   ```
   int SomaInteiros(int A[], int n){
       if (n<0) return 0;
       else   return A[n] + SomaInteiros(A, n-1);
   }
   ```
   Para n<0 T(n) = 1
   Para n> 0 T(n) = T(n-1), logo
   T(n) = T(n-1) + 1 portanto θ*(n)*

   b)
   ```
   b) int easyQuestion(int* A, int n) {
    int  i;
    if (n < 2)return (A[0]);
     for (i=n/2; i<(n/2)+8; i++)
             return A[i] + easyQuestion(A, 3*n/4);
     }
   ```
   É possível perceber que, para cada recursão, o algoritmo executa um laço de complexidade constante (o loop *for* executa aproximadamente 8 iterações em qualquer circunstância) e realiza sua chamada recursiva de tamanho *3n/4*. Assim, chegamos na seguinte relação de recursividade:
   $T(n) = T(3n/4) + c$ portanto θ(lgn)

   c)
   ```
   int youWontGuessThisOne(int* A, int n){
    if (n < 50)  return (A[n]);
    int x, j;
    x = youWontGuessThisOne(A, n/4);
    for (j=0; j<n/3; j++)   A[j] = A[n-j] - A[j];
    x += youWontGuessThisOne(A, n/4);
    return x;
   }
   ```

O algoritmo executa, para qualquer caso diferente do caso base, duas chamadas recursivas (uma antes do loop e outra depois) e um *loop for* de complexidade $\theta(n)$. Isto nos dá a seguinte equação de recorrência:

$T(n) = 2T(n/4) + cn$ portanto $\theta(n)$