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

1.
```
int thisOneIsTricky(int* A, int n) {
 if (n < 12) return (A[0]);
 int y, i, j, k;
 for (i=0; i<n/2; i++) {
   for (j=0; j<n/3; j++) {
          for (k=0; k<n; k++) {
                 A[k] = A[k] - A[j] + A[i];
          }
   }
}
 y = thisOneIsTricky(A, n-5);
 return y;
 }
```

   Ao analisarmos o algoritmo, percebemos que, para cada recursão, há 3 loops *for* de complexidade θ(n) cada (totalizando $\theta(n^3)$) e uma chamada recursiva, fora dos loops, de tamanho *n-5*. Assim, chegamos à seguinte relação de recursividade:

   $T(n) = T(n-5) + cn^3$

   A resolução dessa equação de recorrência resulta em

   $\theta(n^4)$

2.
```
int okLastOneIPromise(int* A, int n){
 if (n < 15)  return (A[n]);
 int x=0, i, j, k;
for (i = 0; i<4; i++){
    for (j=0; j<n-i; j++){
          for (k=0; k<n/2; k++){
                A[j] = A[k] - A[n-j];
          }
    }
   x += okLastOneIPromise(A, n/2);
}
 return x;
}
```

A recursividade encontra-se dentro de um *loop for* de 4 iterações e, portanto, é chamada quatro vezes por execução, cada uma dividindo o problema pela metade. O trabalho desenvolvido fora da recursão encontra-se principalmente dentro dos dois *loops for* mais internos, de complexidade $\theta(n)$ cada. Como esses dois loops estão entrelaçados, temos uma complexidade total de $\theta(n^2)$. Isto nos leva à seguinte relação de recursividade:

$T(n) = 4\,T(n/2) + cn^2$

A resolução dessa equação de recorrência resulta em **$\theta(n^2 \log n)$**

3.
```
int pow2(int a, int n) {
    if (n == 0)
        return 1;
    if (n % 2 == 0)
        return pow2(a,n/2) * pow2(a,n/2);
    else
        return pow2(a,(n-1)/2) * pow2(a,(n-
            1)/2) * a;
        }
```

$T(n) = 2T(n/2) + 1$ portanto $\theta(n)$.

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 ++ ) {
                        for ( k = j; k < n; k /= 2 ) {
                                sum += (i + j * k );
                        }
                }
         }
}
```

$\theta(n^2 logn)$

. Calcule a complexidade, no pior caso e no melhor caso em notação θ(), do fragmento de código abaixo:

```
int i , j ;
A[ i ] [ j ] = 0;
for ( i =0; i < N; i=2*i){
       if(m <500) {
        for ( j =0; j < N*N; j ++){
        A[ i ] [ j ] += A[ i ] [ j ] *B[ i] [ j ] ;
         }
         else if(m>1000){
        for ( j=1; j < N; j*=2)
                A[ i ] [ j ] -= A[ i ] [ j] *B[ i ] [ j ] ;
        }
        else{
        for(j=1; j<N; j++)
                A[ i ] [ j ] += A[ i ] [ j ] +B[ i] [ j ] ;
```

```
        }
}
```

Pior caso: $\boldsymbol{\theta(n^2 logn)}$

Melhor caso: $\boldsymbol{\theta(logn^2)}$