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;
 }
```

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;
 }
```

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;
          }
```

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 );
                }
            }
        }
}
```

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

```
1       int i , j ;
2       A[ i ] [ j ] = 0;
3       for ( i =0; i < N; i=2*i){
4               if(m <500) {
5                for ( j =0; j < N*N; j ++){
6                A[ i ] [ j ] += A[ i ] [ j ] *B[ i] [ j ] ;
 7               }
 8               else if(m>1000){
 9              for ( j=1; j < N; j*=2)
10                      A[ i ] [ j ] -= A[ i ] [ j] *B[ i ] [ j ] ;
11              }
13              else{
14              for(j=1; j<N; j++)
15                      A[ i ] [ j ] += A[ i ] [ j ] +B[ i] [ j ] ;
16              }
17      }
```