```c
int foo6(int a, int b) {
    ...
    if (b < a) {
        return foo4(a+b);
    } else {
        return 24;
    }
}

int foo5(int a, int b) {
    if (a > b) {
        return foo6(a+a, b*b);
    } else {
        return foo6(a/2, b/2);
    }
}

int foo4(int a) {
    ...
    return foo5(a, a*a);
}

int foo3(int a, int b) {
    ...
    if (a != 0) {
        return foo3(a-1, b+1);
    } else {
        return b+1;
    }
}

int foo2(int b) {
    return foo3(foo4(b), b+1);
}

int foo1(int a, int b) {
    ...
    if (a > b) {
        return foo1(b, a);
    } else {
        return foo2(b);
    }
}

int foo() {
    ...
    int v1 = foo1(23, 10);
    int v2 = foo2(10);
    int v3_v4 = foo3(34, 12) + foo4(34-12);
    return v1+v2+v3_v4;
}
```

Listagem 1: Trecho de código para as funções `foo` na Linguagem C

```c
int foobar1(int n, int b) {
    if (n < 2) {
        return n + b;
    } else {
        return foobar1(n-1, b) + foobar1(n-2, b);
    }
}

float foobar2t(int n, int b, float resp) {
    if (n < 1) {
        return resp;
    } else {
        return foobar2t(n-1, b, resp + (b*n));
    }
}

float foobar2(int n, int b) {
    return foobar2t(n, b, 0);
}

float foobar3t(int n, int b, float resp) {
    if (n < 1) {
        return resp;
    } else {
        if (n % 2 != 0) {
            return foobar3t(n-1, b, resp + n/foobar1(n, b));
        } else {
            return foobar3t(n-1, b, resp + foobar2(n, b));
        }
    }
}

float foobar3(int n, int b) {
    return foobar3t(n, b, 0);
}
```

Listagem 2: Trecho de código para as funções `foobar` na Linguagem C

```c
int zoot(int n, int resp) {
    if (n == 1) {
        return resp+1;
    } else {
        if (n %2 == 0)
            return zoot(n/2, resp+1);
        else
            return zoot((n*3)+1 , resp+1);
    }
}

int zoo(int n) {
    return zoot(n, 0);
}
```

Listagem 3: Função `zoo` na Linguagem C

```c
#include <stdio.h>

void foo(int v[], int i, int j);
int bar(int v[], int value, int i, int j);

int bar(int v[], int value, int  i, int j) {
    if (i >= j) {
        return i;
    } else {
        if (v[i] > value && v[j] <= value) {
            int temp = v[i];
            v[i] = v[j];
            v[j] = temp;
        }
        if (v[i] <= value) i++;
        if (v[j] > value) j--;
        return bar(v, value, i, j);
    }
}

void foo(int v[], int i, int j) {
    if (i <= j) {
        int value = v[(i+j)/2];
        int k = bar(v, value, i, j);
        foo(v, i, k-1);
        foo(v, k+1, j);
    }
}

int main (void) {
    int v1[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int v2[] = {4, 65, 2, -31, 0, 99, 2, 83, 2, 1};

    foo(v1, 0, 5);
    foo(v2, 5, 9);

    return 0;
}
```

Listagem 4: Código de programa na Linguagem C

```c
int max_div_comum(int n1, int n2) {
    if (n1 == 0)
        return n2;
    else
        return max_div_comum(n1%n2, n2/n1);
}
```

Listagem 5: Função que calcula o máximo divisor comum de `n1` e `n2`

```c
int count_aux(int v[], int e, int i, int j) {
    if (i > j)
        return 0;
    int k = (i+j)/2;
    if (v[k] == e) {
        i = k;
        j = k;
        while (v[i] == e) i--;
        while (v[j] == e) j++;
        return j-(i+1);
    } else {
        if (v[i] != v[j])
            if (v[k] > e)
                return count_aux(v, e, i, k-1);
            else
                return count_aux(v, e, k+1, j);
        else
            if (v[k] > e)
                return count_aux(v, e, i, k-1);
            else
                return count_aux(v, e, k+1, j);
        }
    }
}

int count(int v[], int e, int n) {
    return count_aux(v, e, 0, n-1);
}
```

Listagem 6: Função `count` que calcula o número de vezes que o elemento `e` aparece no vetor `v[n]` (`v` pode estar ordenado de maneira ascendente ou de maneira descendente - o algoritmo funciona para ambos casos)