```
////////// Soluciones de los ejercicios de la Clase Practica 8 - Complejidad

/** Para el conteo de cantidad de operaciones elementales cuento:
  * - cada acceso a valor de variable
  * - cada inicialización de variable
  * - cada asignación
  * - cada evaluación de operador matemático-lógico/size/push_back
  * *** No cuento los "accesos" a constantes
  * *** cuento incrementos "i++" como 1 operación
  *
  * ejemplos:
  * i < n                   : 2 accesos + 1 op. = 3
  * i == 0                  : 1 acceso  + 1 op. = 2
  * v += m1[i][k]*m2[k][j]  : 6 accesos + 2 op. + 1 asignacion

  *
  * whiles: En la línea que abre escribo el costo de la guarda
  *         y cuántas iteraciones va a ejecutar en peor caso.
  *         Escribo indentado el conteo de ops. del cuerpo del ciclo
  *         En la línea que cierra escribo la fórmula t(n) para todo ese ciclo
  *         t(n) = guarda + n*(guarda+t(cuerpo))
  *
  * fors :  En la línea que abre escribo
  *          - el costo de la inicialización
  *          - el costo de evaluar la guarda
  *          - el costo del incremento
  *         Escribo indentado el conteo de ops. del cuerpo del for
  *         En la línea que cierra escribo la fórmula t(n) para todo ese for
  *         La fórmula de for es =  inicializar + guarda + n*(t(cuerpo)+guarda+incremento)
  *
  * *** En la línea que cierra la función escribo la expresión del costo total, y calculo su O()
  */
```

```cpp
// Ejercicio 1

int doble(int n) {
  return 2*n;         // 2
}                     // t(n) = 2 -> O(1)



int doble2(int n) {
  int res = 0;        // 1
  int i = 0;          // 1
  while (i < n) {     // 3, n iteraciones
    res = res+2;      // 3
    i = i+1;          // 3
  }                   // t(n) = 3 + 9*n
  return res;         // 1
} //                  // t(n) = 6 + 9*n   -> O(n)
```

```cpp
// Ejercicio 2a
void f(vector<int> &v) {          // |v| = n
  int i = v.size() / 2;           // 3
  while (i >= 0) {                // 2, n/2 iteraciones
    v[v.size() / 2 - i] = i;       // 6
    v[v.size() / 2 + i] = i;       // 6
    i--;                           // 1
  }                               // t(n) = 2 + 15(n/2)
}                                 // t(n) = 5 + 15(n/2) -> O(n)

// Ejercicio 2b
// PRE: e pertenece a v1
int f(vector<int> &v1, int e) {
  int i = 0;                  // 1
  while (v1[i] != e) {        // 4, n iteraciones
    i++;                       // 1
  }                           // t(n) = 4 + 5n
  return i;                   // 1
}                             // t(n) = 6 + 5n

// Ejercicio 2c

void f(vector<int> &v1, vector<int> &v2) {  // |v1| = n, |v2| = m
  vector<int> res;                          // 1
  for (int i = 0; i < v1.size(); i++) {     // init: 1, guarda: 3, incremento: 1, n iteraciones
    res.push_back(v1[i]);                    // 3
  }                                         // t(n) = 4 + 7n
  for (int i = 0; i < v2.size(); i++) {     // init: 1, guarda: 3, incremento: 1, m iteraciones
    res.push_back(v2[i]);                    // 3
  }                                         // t(m) = 4 + 7m
  return res;                               // 1
}                                           // t(n,m) = 10 + 7n + 7m -> O(n + m)
```

```cpp
// Ejercicio 3

vector<int> restarLosPares(vector<int> S, int x) {  // |s| = n
    vector<int> res;                  // 1
    int i = 0;                        // 1
    int suma = 0;                     // 1
    while (i<S.size()) {              // 3, n iteraciones
        if (S[i] % 2 == 0)             // 4
            suma += S[i];              // 5
        res.push_back(x - suma);       // 1
    }                                 // t(n) = 3 + 13n
    return res;                       // 1
}                                     // t(n) = 7 + 13n -> O(n)
```

```cpp
// Ejercicio 4

int detTriangular(vector<vector<int> > M) { // |M| = n
    int i = 0;                              // 1
    int res = 1;                            // 1
    while (i<M.size()) {                    // 3, n iteraciones
        res = res * M[i][i];                    // 6
    }                                       // t(n) = 3 + 9n
    return res;                             // 1
}                                           // t(n) = 6 + 9n -> O(n)
                                            // En función de la cantidad de filas: n = sqrt(|M|)
                                            // t(n) -> O(n) = O(sqrt(|M|))

bool esTriangular(vector<vector<int> > M) {// |M| = n
    res = true;                             // 1
    for (int i=0; i<M.size(); i++) {        // init: 1, guarda: 3, incremento: 1, n iteraciones
        for (int j=0; j < i; j++) {             // init: 1, guarda: 3, incremento: 1, (i-1) iteraciones
            res = res && M[i][j] == 0;              // 7
        }                                       // t(i) = 4 + 11i
    }                                       // t(n) = 4 + suma(i=0,n-1)(4 + 11i)
    return res;                             // 1
}                                           // t(n) = 4 + 4n + 11*(n*(n-1)/2)   (suma de gauss 0...n-1)
                                            //      = 4 + 4n + (11n²-11n)/2     -> O(n²)
                                            // En función de la cantidad de filas: n = sqrt(|M|)
                                            // t(n) -> O(n²) = O(sqrt(|M|)²) = O(|M|)
```

```cpp
// Ejercicio 5

// Pre: m1 y m2 son matrices, y |m1[0]| = |m2|
vector<vector<int> > multiplicar(vector<vector<int> > m1, vector<vector<int> > m2) {
    vector<vector<int> > res;
    for (int i=0; i < m1.size(); i++) {
        vector<int> fila;
        for (int j=0; j<m2[0].size(); j++) {
            int v=0;
            for (int k=0; k<m2.size(); k++) {
                v += m1[i][k]*m2[k][j];
            }
            fila.push_back(v);
        }
        res.push_back(fila);
    }
    return res;
}
```

```cpp
//|m1|=n, |m1[0]|=|m2|=m, |m2[0]|=r
// 1
// init: 1, guarda: 3, incremento: 1, n iteraciones
    // 1
    // init: 1, guarda: 3, incremento: 1, r iteraciones
        // 1
        // init: 1, guarda: 3, incremento: 1, m iteraciones
            // 9
        // t(m) = 4 + 13m
        // 2
        // t(r,m) = 4 + r*(4 + 13m)
    // 2
// t(n,r,m) = 4 + n*(4 + r*(4 + 13m))
// 1
// t(n,r,m) = 6 + n*(4 + r*(4 + 13m)) = 6 + 4n + 4nr+ 13nrm
//             -> O(n*r*m)
// item 3.2) n=m=r
// t(n,r,m) = 6 + 4n + 4n² + 13n³ -> O(n³)
```