Ejercicio 1

```
int suma(vector<int> &v) {
    int suma = 0;
    for (int i = 0: i < v.size(); i++) {
        suma = suma + v[i];
    }

    return suma;
}
```

- **Definir el tamaño de entrada n.**
  El tamaño de entrada n es el tamaño del vector v.

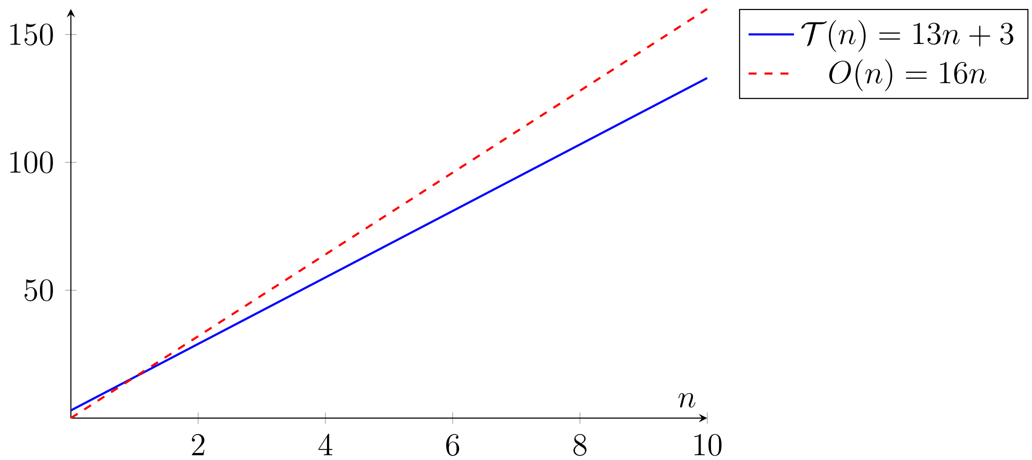
- **Identificar el peor caso de ejecución del programa.**
  Todos los casos son los peores, entre más grande sea $|v|$, más tardará en ejecutar el programa.

- **Escribir la función de costo temporal $\mathcal{T}(n)$ asociada al peor caso.**
  $\mathcal{T}(n) = (1) + (1) + n(4 + 3 + 6) + (1)$
  $\implies \mathcal{T}(n) = 3 + n(13)$
  $\therefore \mathcal{T}(n) = 13n + 3$
  Las declaraciones tienen un costo de 1, que es el primer paréntesis.
  En el segundo paréntesis también, es el `int i = 0` que se hace una sola vez en el ciclo `for`.
  Luego, se hace la comparación, acceso a `i`, acceso a `v` y la operación `.size()`. Esto tiene un costo de 4 que se ejecuta n veces. También se ejecuta n veces cuando se actualiza la variable `suma`: el acceso a la variable `suma` y `v`, sumado al operador $+$ e $=$, y el acceso a la posición `i` hace que tenga un costo esta línea de 6. Luego se incrementa `i`, se accede, se suma uno y se vuelve a guardar con un costo de `3`.
  Finalmente, se devuelve la variable `suma`, que en los `return` tiene un costo de 1.

- **Proponer un orden de complejidad $\mathcal{O}$ para $\mathcal{T}(n)$.**
  $\mathcal{T}(n) = 13n + 3 \in O(n)$

- **Mostrar que $\mathcal{T}(n)$ pertenece al órden de complejidad propuesto.**
  Quiero verificar que:
  $0 \leq \mathcal{T}(n) \leq c \cdot O(n), (\forall n \geq n_0)$
  $\implies 0 \leq 13n + 3 \leq c \cdot n$
  Sabemos que $13n + 3$ es estrictamente creciente, por lo tanto, será siempre mayor o

igual a cero con un $n_0 \geq 0$.
$\implies 13n + 3 \leq c \cdot n$
$\implies \frac{13n+3}{n} \leq c$
Ya cuando despejo, puedo usar el n_0.
$\implies 13 + \frac{3}{n_0} \leq c$
Si tomamos que $n_0 = 1 \land c = 16$, vemos que se cumple para todo $n \geq n_0$.



$\therefore \mathcal{T}(n) = 13n + 3 \in O(n)$

Ejercicio 2

```
int sumaMientrasMenor(vector<int> &v, int max) {
    int suma = 0;
    int i = 0;
    while (i < v.size() && v[i] <= max) {
        suma = suma + v[i];
        i = i + 1;
    }

    return suma;
}
```

- **Definir el tamaño de entrada n.**
  El tamaño de entrada n será el tamaño del vector `v`, llamado $|v|$.

- **Identificar el peor caso de ejecución del programa.**
  El peor caso es si `max` es mayor a todos los elementos del vector `v`. En este caso, el ciclo iterará más veces porque todos los elementos serán menores a `max`.

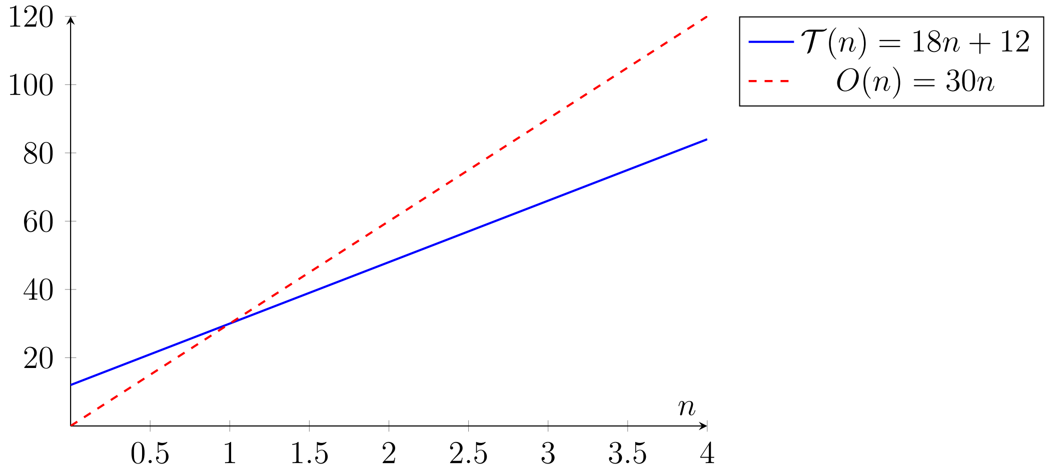
- **Escribir la función de costo temporal $\mathcal{T}(n)$ asociada al peor caso.**
  La definición de las variables en la primera y segunda línea tienen un costo de 1 cada una.
  En la comparación `i < v.size()` se debe acceder a las dos variables, acceder al valor de una y después compararlas, por lo que tiene un costo de 4. En la comparación `v[i] <= max` se accede a tres variables, se accede a un valor guardado en una, y se comparan, por lo que tiene un valor de 5.
  Dentro del `while`, se accede en la primera línea a cuatro variables, se suman y se guarda en una variable. Esto tiene un costo de 6.
  Se incrementa `i`, se accede a ella, se le suma uno y se guarda. Tiene un costo esto de 3. Esto se ejecuta n veces.
  Finalmente, se devuelve `suma` con costo 1.

  $\mathcal{T}(n) = (1 + 1) + (4 + 5 + 6 + 3) \cdot n + (4 + 5) + (1)$
  Incluyo la negación de la guarda (es el $4 + 5$ después de la multiplicación)
  $\implies \mathcal{T}(n) = (2) + (18) \cdot n + (9) + (1)$

$\implies \mathcal{T}(n) = (2) + (18) \cdot n + 10$
$\therefore \mathcal{T}(n) = 18n + 12$

- **Proponer un orden de complejidad $\mathcal{O}$ para $\mathcal{T}(n)$.**
  $\mathcal{T}(n) \in O(n)$

- **Mostrar que $\mathcal{T}(n)$ pertenece al orden de complejidad propuesto.**
  Quiero probar que $0 \leq \mathcal{T}(n) \leq c \cdot O(n), (\forall n \geq n_0)$.
  $\implies 0 \leq 18n + 12 \leq c \cdot n, (\forall n \geq n_0)$
  $\implies 0 \leq 18n + 12$ Vale porque $18n + 12$ es estrictamente creciente.
  $\implies 18n + 12 \leq c \cdot n, (\forall n \geq n_0)$
  $\implies \frac{18n+12}{n} \leq c, (\forall n \geq n_0)$
  $\implies 18 + \frac{12}{n_0} \leq c, (\forall n \geq n_0)$
  Si tomo $n_0 = 1 \land c = 30$, se cumple la desigualdad para todo $n \geq n_0$.



$\therefore \mathcal{T}(n) = 18n + 12 \in O(n)$

Ejercicio 3

```
int sumaDoblePares(vector<int> &v) {
    int suma = 0;
    int i = 0;
    while (i < v.size()) {
        if(v[i] % 2 == 0) {
            int j = 0;
            while (j < v.size()) {
                suma = suma + v[j];
                j = j + 1;
            }
        }
        i = i + 1;
    }

    return suma;
```

- **Definir el tamaño de entrada n.**
  El tamaño de la entrada es `|v|`.

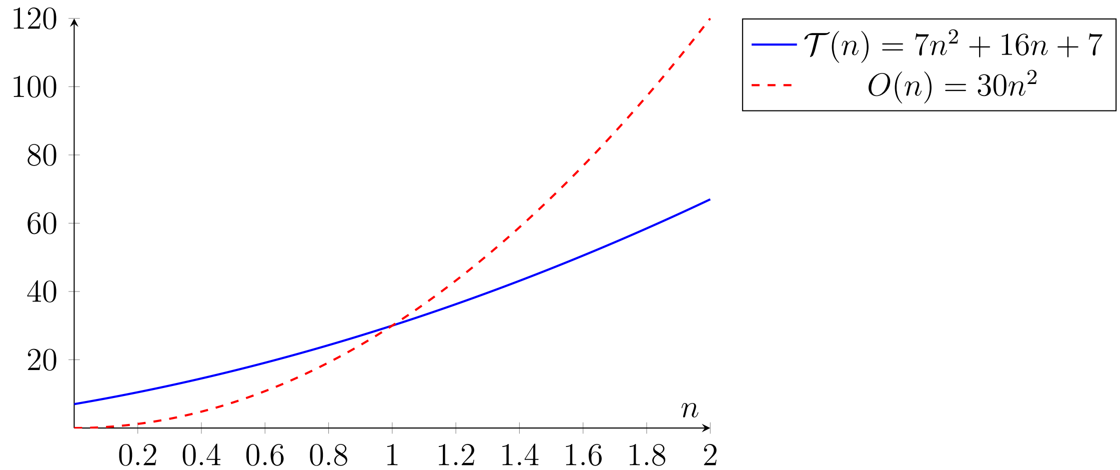
- **Identificar el peor caso de ejecución del programa.**
  El peor caso sucede si tiene todos elementos pares el vector. En ese caso, si se encuentra un par, suma todos los elementos a la variable `suma` hasta el final del vector. Entre más pares haya, más operaciones hará.

- **Escribir la función de costo temporal $\mathcal{T}(n)$ asociada al peor caso.**
  Definición de las variables `suma` e `i` tienen un costo de 1.
  En la guarda del primer while, se accede a dos variables, se hace una operación en una y se hace una comparación. Esto tiene un costo de 4.
  En el `if` se accede a dos variables, se hace una operación y se hace una comparación. Esto tiene un costo de 4.

Se define una variable con costo 1.
En la guarda del segundo while, se accede a dos variables, se hace una operación en una y se hace una comparación. Esto tiene un costo de 4.
Se acceden a tres variables, se hace una operación, y se guarda el valor. Esto tiene un costo de 5.
Se accede a `j` y se hace una operación. Tiene un costo de 2.
Se incrementa la variable `i` en uno. Costo de 2.
Finalmente, se devuelve la variable `suma` con costo 1.

$\mathcal{T}(n) = (1 + 1) + (4 + 4 + 1 + 4 + 2 + 1 + (5 + 2) \cdot n) \cdot n + (4) + (1)$
$\implies \mathcal{T}(n) = (16 + 7n) \cdot n + 7$
$\therefore \mathcal{T}(n) = 16n + 7n^2 + 7$

- **Proponer un orden de complejidad $\mathcal{O}$ para $\mathcal{T}(n)$.**
  $\mathcal{T}(n) \in O(n^2)$
- **Mostrar que $\mathcal{T}(n)$ pertenece al orden de complejidad propuesto.**
  Quiero probar que $0 \leq \mathcal{T}(n) \leq c \cdot O(n^2), (\forall n \geq n_0)$
  $\implies 0 \leq 16n + 7n^2 + 7 \leq c \cdot n^2, (\forall n \geq n_0)$
  $\implies 0 \leq 16n + 7n^2 + 7 (\forall n \geq n_0)$ Vale, porque $16n + 7n^2 + 7$ es una función creciente.
  $\implies 7n^2 + 16n + 7 \leq c \cdot n^2, (\forall n \geq n_0)$
  $\implies \frac{7n^2+16n+7}{n^2} \leq c, (\forall n \geq n_0)$
  $\implies 7 + \frac{16}{n_0} + \frac{7}{n_0^2} \leq c, (\forall n \geq n_0)$
  Si tomamos que $n_0 = 1 \land c = 30$, se cumple la desigualdad para todo $n \geq n_0$.



$\therefore \mathcal{T}(n) = 7n^2 + 16n + 7 \in O(n^2)$

Ejercicio 3b

```
int sumaDoblePares(vector<int> &v) {
    int suma = 0;
    int i = 0;
    while (i < v.size()) {
        if(v[i] % 2 == 0) {
            int j = 0;
            while (j < i) { // antes: j < v.size()
                suma = suma + v[j];
                j = j + 1;
            }
        }
        i = i + 1;
    }

    return suma;
```

- **Definir el tamaño de entrada n.**
  El tamaño de entrada es el tamaño del vector `|v|`. $\implies n = |v|$

- **Identificar el peor caso de ejecución del programa.**
  El peor caso es cuando todos los elementos del vector son pares. Esto implica que se van a sumar todos los elementos previos a un índice par.

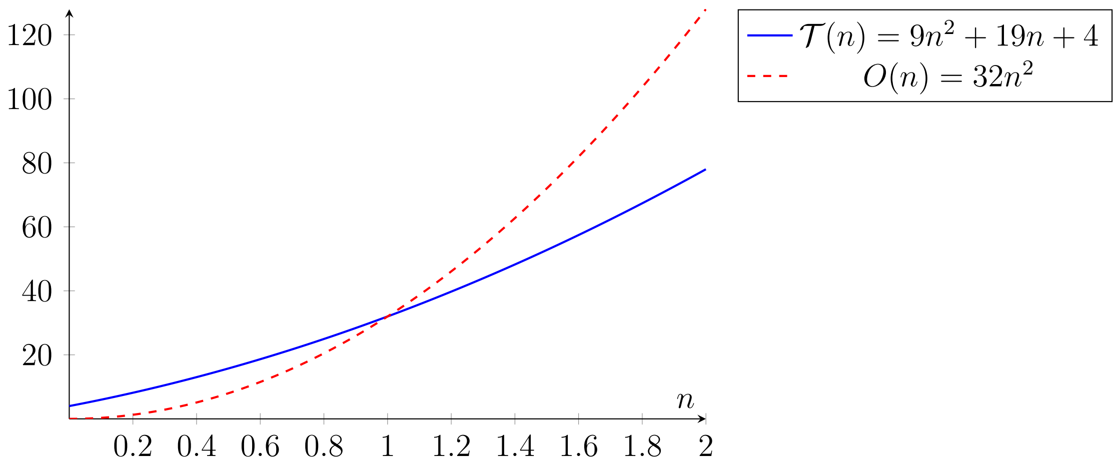
- **Escribir la función de costo temporal $\mathcal{T}(n)$ asociada al peor caso.**
  En las primeras dos líneas se hacen dos operaciones, cuatro en total.
  En la guarda del `while` se hacen cuatro operaciones: se accede a `i`, `v`, se hace la operación `.size()` y se hace una comparación. Todo lo que está en este scope se ejecuta n veces.
  En la condición del `if` se accede a dos variables, se le hace una operación y una comparación. Son, en total, cuatro operaciones.
  Se define la variable `j`, cuenta como una operación.
  En esta guarda del `while` se hacen tres operaciones porque se acceden a dos variables y se hace una comparación entre ellas. Todo lo que está en este scope se ejecuta n veces.
  Se hacen cuatro operaciones, porque se accede a `suma`, `v` y `j` y se hace una operación

entre ellas. No cuenta cuando se guarda de nuevo en `suma` porque es una variable que ya se accedió.
Para incrementar `j`, se tuvo que acceder a la variable y sumarle uno. Son dos operaciones.
Se incrementa `i` en un total de dos operaciones.
Se devuelve `suma` en una sola operación.

$\mathcal{T}(n) = (2+2) + n(4+4+1+n(3+4+2)+3+4+2)+1$
$\implies \mathcal{T}(n) = 4 + n(9+n(9)+9)+1$
$\implies \mathcal{T}(n) = 4 + n(19+9n)$
$\implies \mathcal{T}(n) = 4 + 19n + 9n^2$
$\therefore \mathcal{T}(n) = 9n^2 + 19n + 4$

- **Proponer un orden de complejidad $\mathcal{O}$ para $\mathcal{T}(n)$.**
  $\mathcal{T}(n) \in O(n^2)$

- **Mostrar que $\mathcal{T}(n)$ pertenece al orden de complejidad propuesto.**
  Quiero probar que $0 \leq \mathcal{T}(n) \leq c \cdot O(n^2), (\forall n \geq n_0)$.
  $\implies 0 \leq 9n^2 + 19n + 4 \leq c \cdot n^2, (\forall n \geq n_0)$
  $\implies 0 \leq 9n^2 + 19n + 4, (\forall n \geq n_0)$ Vale, porque $9n^2 + 19n + 4$ es una función creciente.
  $\implies 9n^2 + 19n + 4 \leq c \cdot n^2, (\forall n \geq n_0)$
  $\implies \frac{9n^2+19n+4}{n^2} \leq c, (\forall n \geq n_0)$
  $\implies 9 + \frac{19}{n_0} + \frac{4}{n_0^2} \leq c, (\forall n \geq n_0)$
  Si tomo que $n_0 = 1 \wedge c = 32$:
  $\implies 9 + \frac{19}{1} + \frac{4}{1^2} \leq 32, (\forall n \geq n_0)$
  $\implies 32 \leq 32, (\forall n \geq n_0)$ Se cumple la desigualdad para todo $n \geq n_0$.

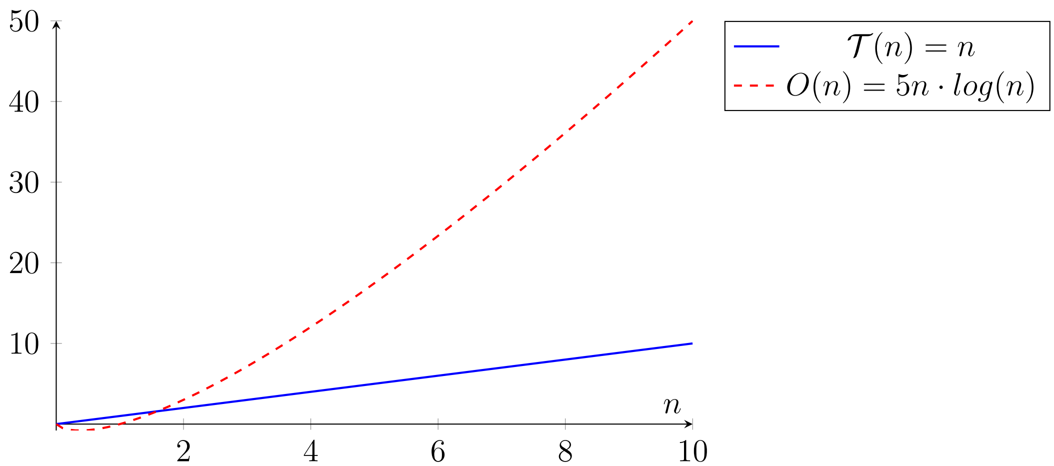


$$\therefore \mathcal{T}(n) = 9n^2 + 19n + 4 \in O(n^2)$$

Análisis de funciones

1. $n \in O(n \cdot log(n))$

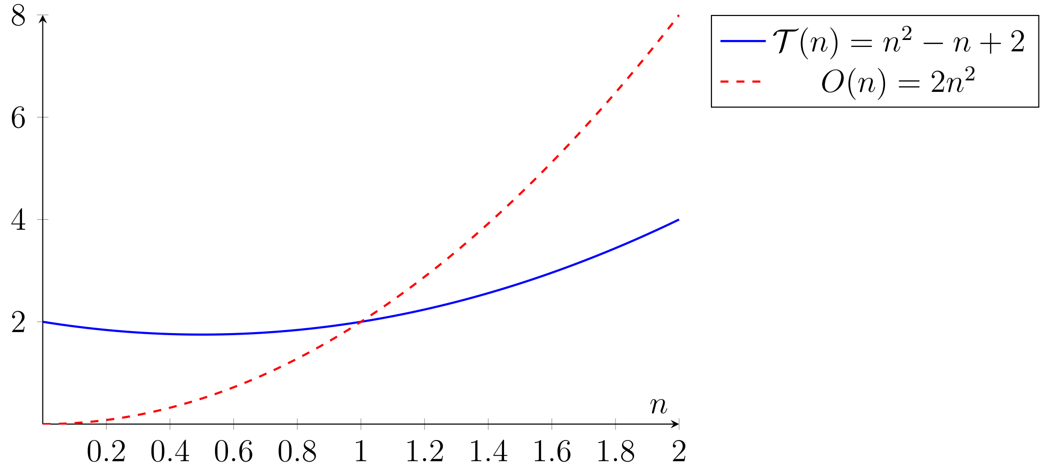
Quiero probar que $0 \leq n \leq c \cdot O(n \cdot log(n)), (\forall n \geq n_0)$.
$\implies 0 \leq n, (\forall n \geq n_0)$ vale, porque n es una función creciente.
$\implies n \leq c \cdot n \cdot log(n), (\forall n \geq n_0)$
$\implies \frac{n_0}{n_0 \cdot log(n_0)} \leq c, (\forall n \geq n_0)$
$\implies \frac{1}{log(n_0)} \leq c, (\forall n \geq n_0)$
Si tomo que $n_0 = 2 \wedge c = 5$ se cumple la desigualdad $\forall n \geq n_0$



$\therefore$ Esta afirmación es verdadera.

2. $n^2 - n + 2 \in O(n^2)$

Quiero probar que $0 \leq n^2 - n + 2 \leq c \cdot O(n^2), (\forall n \geq n_0)$.
$\implies 0 \leq n^2 - n + 2, (\forall n \geq n_0)$ vale porque $n^2 - n + 2$ es una función creciente y positiva para todo $n \geq 0$.
$n^2 - n + 2 \leq c \cdot n^2, (\forall n \geq n_0)$
$\frac{n_0^2 - n_0 + 2}{n_0^2} \leq c, (\forall n \geq n_0)$
$1 - \frac{n_0}{n_0^2} + \frac{2}{n_0^2} \leq c, (\forall n \geq n_0)$
$1 - \frac{1}{n_0} + \frac{2}{n_0^2} \leq c, (\forall n \geq n_0)$
Si tomo que $n_0 = 1 \wedge c = 2$, se cumple la desigualdad para todo $n \geq n_0$.



$\therefore$ Esta afirmación es verdadera.

Complejidad de algoritmos recursivos

```
minimoAux(v, 0, |v|)

int minimoAux(vector<int> &v, int desde, int hasta) {
    if (desde - hasta <= 1) {
        return v[desde];
    }
    int mitad = (desde+hasta)/2;
    int sol1 = minimoAux(v, desde, mitad);
    int sol2 = minimoAux(v, mitad, hasta);
    return min(sol1, sol2);
}
```

- **Definir el tamaño de entrada n.**
  El tamaño de entrada es el tamaño del vector `v`. $\implies n = |v|$

- **Identificar el peor caso de ejecución del programa.**
  El peor caso sucede cuando la variable `hasta` es igual al tamaño de `v` y cuando `desde` es igual a cero. Esto implica que se quiere buscar el mínimo de todo el vector. Peor será el caso que tenga una ejecución con un vector `v` más grande.

- **Escribir la ecuación de recurrencia $\mathcal{T}(n)$ asociada al peor caso.**
  $\mathcal{T}(n) = \begin{cases} k & \text{si } n = 0, n = 1 \\ 2\mathcal{T}(n/2) + k & \text{si } n > 1 \end{cases}$

- **usar el árbol de recursión para dar una fórmula cerrada para $\mathcal{T}(n)$.**
  $\mathcal{T}(n) = 2^i\mathcal{T}(n/2^i) + k(2^i - 1)$

- **usando la fórmula cerrada, proponer un orden de complejidad $\mathcal{O}$ para $\mathcal{T}(n)$ y demostrar que pertenece.**
  $\mathcal{T}(n) \in O(n)$
  $\implies 0 \leq 2kn - k \leq c \cdot n$
  $\implies 2kn - k \leq c \cdot n$
  $\implies \frac{2kn_0 - k}{n_0} \leq c$
  $\implies 2k - \frac{k}{n_0} \leq c$
  Si tomo que $n_0 = 1 \land c = k$,
  $\implies 2k - \frac{k}{1} \leq k$
  $\implies k \leq k$ Se cumple $\forall n > n_0$

$\therefore \mathcal{T}(n) \in O(n)$