Análisis de Algoritmos 2024

Trabajo Práctico 2 - Notación Asintótica 1
Fecha de entrega: 30 de Agosto de 2024

El objetivo de este trabajo es realizar los ejercicios propuestos de forma colaborativa. Para los ejercicios compartidos entre grupos, pueden compartir el desarrollo del ejercicio o realizar dos resoluciones diferentes.
En particular para este práctico se pide en algunos casos tener código funcionando, verifica tener una computadora que ejecute tu código si te tocó algún ejercicio de este tipo. Se pedirá ejecución en clase.
Todos los ejercicios deben ser resueltos por todos los estudiantes, pero en este trabajo sólo deben ser publicados los que te tocaron según la asignación de la tabla siguiente. Dicha asignación corresponde a la presentación oral del TP. Si el día de la práctica hay varias resoluciones logradas, elegiremos 1 grupo para exponer ese ejercicio.
Por supuesto, si querés compartir la resolución de otros ejercicios en este proyecto Overleaf, también son bienvenidas pero no es obligatorio.

Nota: Colocá el nombre del/de los estudiante/s que resolvió/resolvieron el ejercicio al iniciar la resolución del ejercicio que te tocó.

Las respuestas deben quedar escritas en este mismo .tex, en Overleaf.

Grupos (Estudiantes por grupo)	**Ej. asignado**
Grupo 1 (Sthefany, Victoria, Paula, Adriano, Facundo)	1.1 || 2.1 || 2.2|| 3.1 || 3.3.h || 3.4.a
Grupo 2 (Sebastián, Bautista, Antonio, Albany, Luis)	1.2 || 2.3 || 3.2.a || 3.3.a || 3.3.g || 3.4.b
Grupo 3 (Lucas, Valentina, Nicanor, Ignacio, Franco)	1.3|| 2.4 || 3.2.b || 3.3.f || 3.4.c
Grupo 4 (Manuel, Jan, Ulises, Demian , Luciano)	2.5 || 3.2.c || 3.3.e || 3.4.d || 4.3|| 4.1.iii
Grupo 6 (Roman, Ulises, Guillermo, Jamiro, Juan)	2.4 || 3.2.d || 3.3.d || 3.4.a || 4.2
Grupo 7 (Christopher, Tomás, Joaquín, Leonard, Facundo)	2.3 || 3.2.e || 3.3.c || 3.4.b || 4.1.i
Grupo 8 (Facundo, Belén, Jeremías, Bruno, Kevin)	2.2 || 3.2.d || 3.3.b || 3.4.c || 4.1.ii

1. Preguntas teóricas

1. Dado un conjunto de algoritmos que resuelven el mismo problema, ¿qué consideraciones usaría para elegir uno?.

Resolución ejercicio 1.1 - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

Las consideraciones al momento de seleccionar un algoritmo que resuelve el mismo problema dependerán de varios factores que irán llevando a la mejor solución.

- Aspectos Teóricos:
  - **Orden de Crecimiento**: es el primer criterio que se tiene en cuenta; cuanto menor sea el orden, mejor, ya que requerirá menos tiempo que otros algoritmos de orden mayor. Si el tiempo de ejecución en el peor de los casos de un algoritmo es el menor de todos, entonces el algoritmo es el más eficiente.
  - **Eficiencia Relativa**: dentro del mismo orden se pueden comparar algoritmos de modo que se puede averiguar cuál tiene menor crecimiento cuando el volumen de datos es mayor.
  - **Factores extra**: costos de almacenamiento y legibilidad del código.
- Aspectos Empíricos:
  - **Frecuencia y Volumen de Datos**: para volúmenes grandes de datos o alta frecuencia de ejecución, se debe considerar la distribución de la carga y la optimización en sistemas distribuidos o paralelos.

item 1 y 2 iguales. Extender un poco el item de Factores extra según lo conversado en clase. Qué otras cosas considerarían?

2. ¿Qué es una prueba empírica? ¿Para qué sirve? Indique un ejemplo de un código no correcto, otro no eficiente que haya dado buenos resultados empíricos.

Resolución ejercicio 1.2 - Grupo 2 - Sebastián, Bautista, Antonio, Albany, Luis

Una prueba empírica consiste en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y en una computadora en concreto.

Sirve para representar las medidas reales del comportamiento del algoritmo. Además, si un análisis teórico sugiere que un algoritmo es eficiente, una prueba empírica puede validar o refutar esta hipótesis en la práctica.

Ejemplo de código no correcto:
Por ejemplo, el usuario solicita un programa que sume todos los números pares de un array. No seria código correcto si no cumple con el requerimiento, es decir, que el código sume todos los números del array sin filtrar cuáles sean pares, porque esto ocasiona una salida incorrecta del programa.
Ejemplo de código no eficiente que haya dado buenos resultados empiricos:
Usar un BubbleSort (que tiene un orden $O(n^2)$ que lo hace ineficiente en muchos casos) para un arreglo que este casi ordenado o sea pequeño.

revisar y proponer un código NO correcto. Definir mejor ququé es un código correcto. Ok el ejemplo de código NO eficiente

3. Si analiza un algoritmo de acuerdo a la periodicidad o frecuencia de uso, ¿qué puede decir?

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

Si analizamos un algoritmo de acuerdo a la frecuencia de uso, podemos relacionarlo con qué tan importante es la eficiencia del mismo. Si la frecuencia de uso aumenta, necesitaremos que este algoritmo sea más eficiente. En cambio, si no es tan frecuentemente utilizado, tal vez no nos importe tanto el tiempo que tarda en ejecutarse.

2. Algoritmos y su notación asintótica

1. Especifique un código que a partir de un valor de entrada n tenga un orden constante.
   a) Indique tiempos de ejecución.
   b) Ejecute dicho algoritmo con diferentes valores de entrada

Resolución ejercicio 2.1 - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

Descripción del programa:
El siguiente código plantea una operación aritmética mod 10 (en Java se denota %), la cual es de orden O(1). Esta operación nos permite obtener el último dígito de un número entero. A su vez, incluimos en el 'main' una operación llamada 'System.nanoTime()' que nos permite calcular el tiempo de ejecución empírico, en nanosegundos.

```java
public class OrdenConstante {

    public static int ultDigito(int n) {
        int ultimoDig = Math.abs(n) % 10;
        // Una op aritmetica es de orden constante.
        // Tiempo % O(1)
        // Tiempo = O(1)
        // Tiempo Math.abs O(1)

        return ultimoDig;
    }

    public static void main(String[] args) {
        int dig;
        int n;
        Random r = new Random();
        long[][] tiempos = new long[2][100000];
        long acum = 0; // Acumulador de tiempos de ejecucion

        //Generamos un arreglo para guardar los tiempos de
           ejecucion de 100000 n randoms

        for (int i = 0; i < 100000; i++) {
            n = r.nextInt();
            long inicio = System.nanoTime();
            dig = ultDigito(n);
            long fin = System.nanoTime();
            long tiempoEj = fin - inicio;
            tiempos[i] = tiempoEj;
            acum = acum + tiempoEj;
            System.out.println("Tiempo de ejecucion: " + tiempos[i
               ] + " nanosegundos_ Para n = " + n);
        }
        System.out.println("Promedio: " + acum / 100000);

    }
}
```

Tiempos de ejecución EMPÍRICOS para un lote de 100000 randoms
Aquí brindamos un enlace a un gráfico más amplio de la muestra: Prueba empírica 2.1.

n	Tiempo (nanosegundos)
1144156568	100
-1869704638	200
2105518683	100
.	.
Promedio	**76**

Resolución ejercicio 2.1 - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

Tiempo de ejecución TEÓRICO:

Línea 4: tiempo de asignación + tiempo Math.abs + tiempo op. ´ % : 1 + 1 + 1 = 3 (O(1))
Línea 10: tiempo de return = 1 (O(1))

Tiempo de ejecución para diferentes valores de n:

Valor de n	Tiempo promedio de 5 corridas (nanosegundos)
10	21820
1000	27640
100000	20340
10000000	23400

Revisar de qué forma se puede ejecutar la prueba una cantidad de veces alta para mejorar el promedio de tiempos empíricos

2. Especifique un código que realice algo específico que tenga un orden $O(n)$.

   a)Ejecútelo con un n=10000.

Resolución ejercicio 2.2 - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

El código de orden O(n) que elegimos en este caso es uno que cree y calcule la media aritmética de un arreglo de valores reales contenidos entre 1 y 10. Se especifica el mismo a continuación. Luego se muestran los tiempos empíricos de 5 corridas. Por simplicidad, nos interesa solo la medición sobre el método *mediaAritmetica*.

```
1  public class OrdenLineal {
2      public static void main(String[] args) {
3          int n = 10000;
4          int techo = 10;
5          int piso = 1;
6          double media;
7          double[] arrReales = generarArregloReales(n, techo, piso);
8          long inicio = System.nanoTime();     //inicio medicion
9          media = mediaAritmetica(arrReales, n);
10         long fin = System.nanoTime();    //fin medicion
11         System.out.println("La media aritmetica es: " + media);
12         System.out.println("Tiempo total: " + (fin-inicio) + " nS.
               ");
13     }
14     public static double[] generarArregloReales(int cant, int
           techo, int piso){
15         double[] arr = new double[cant];
16         for(int i=0;i<cant;i++){
17             arr[i] = Math.random()*(techo-piso) + piso;
18         }
19         return arr;
20     }
21     public static double mediaAritmetica(double[] arrValores, int
           n){
22         double mediaAr = 0;
23         if (n>1){
24             for(int i=0;i<n;i++){
25                 mediaAr = mediaAr + arrValores[i];
26             }
27             mediaAr = mediaAr/n;
28         }
29         else{
30             if(n==1){
31                 mediaAr = arrValores[0];
32             }
33         }
34         return mediaAr;
35     }
36 }
```

Tiempos empíricos:

Corrida 1: 14504800 nS = 14,5048 mS.
Corrida 2: 16890300 nS = 16,8903 mS.
Corrida 3: 16828700 nS = 16,8287 mS.
Corrida 4: 13914500 nS = 13,9145 mS.
Corrida 5: 17815600 nS = 17,8156 mS.
Promedio de las 5 corridas: 15990780 ns = 15,99078 mS.

revisar en qué momento se define la lectura del tiempo, hay dos algoritmos qué medir, preferentemente medir el mediaAritmetica().

Ejercicio 2.2 - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

```java
public class divisoresParesN {
    public static void main(String[] args) {
        final int NUMERO = 10000;
        int cantidadPares = 0;
        for (int i = 2; i <= NUMERO; i+2) {
            if( i % 2 == 0)
                cantidadPares++
        }
        System.out.println("Cantidad de numeros pares en: " +
            NUMERO +" :" +cantidadPares);
    }
}
```

Ejecutado en dos computadoras distintas

Computadora 1 19ms

Computadora 2 78ms

agregar código donde se hace la medición. Evaluar mas casos y dar resultado en tabla. Poner alguna conclusión de la comparación.

3. Realice lo mismo para un algoritmo de $O(n^2)$

a)Indique tiempo empírico con n = 10000.

Ejercicio 2.3 G7: Ovaillos, Medel Severini, Fernandez, Occhi, Perez Penas

```java
public class AlgOrdenCuadr {
    public static void main(String[] args) {
        long tiempoInicial = System.nanoTime();
         int n = 10000;
        int acum = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                acum = acum + 1;
            }
        }
        long tiempoFinal = System.nanoTime();
        long duracion = tiempoFinal - tiempoInicial;
        System.out.println("Tiempo en nanoseg: " + duracion);
        System.out.println("Total: " + acum);
    }}
```

El algoritmo de orden cuadratico es simple, este realiza la cuenta de n^2 por medio de un bucle anidado (utilizando FOR).

Acontinuacion mostraremos el tiempo en nanosegundos de las pruebas realizadas.

N	Tiempo promedio en 100 casos
1	123
10	1682
100	173504
1000	853683
2000	2527057
3000	5217672
5000	12691273
8000	32174529
10000	50051157

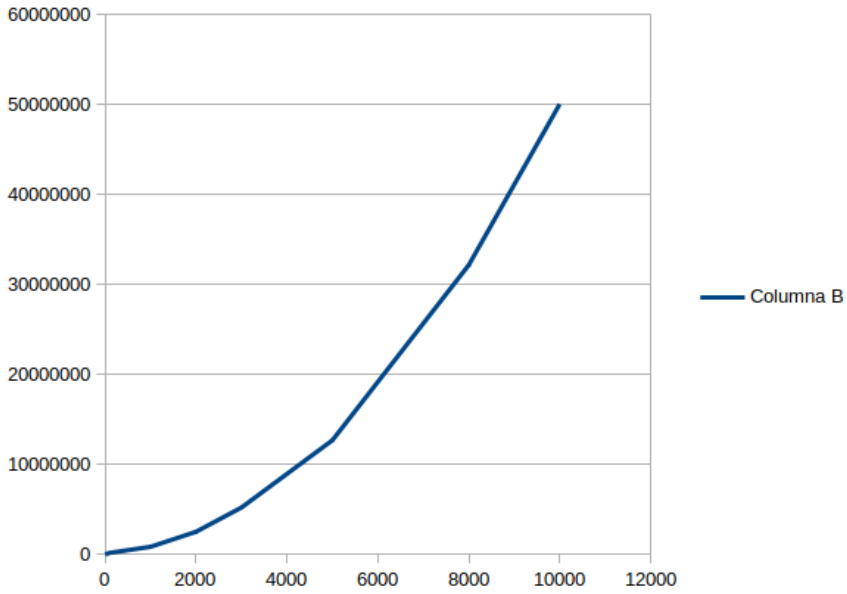
Grafico de la tabla

Tiempo de ejecucion teorico

$$
\begin{aligned}
T(n) &= T_4 + T_5 + T_6 \\
T_{\text{inicializacion}} &= 1 \\
T_{\text{incremento}} &= T_{\text{operacion}} + T_{\text{asignacion}} = 2 \\
T_{\text{condicion}} &= 1 \\
T_{\text{asignacion}} &= 1 \\
T_{\text{operacion}} &= 1 \\
T_4 &= T_{\text{asignacion}} = 1 \\
T_5 &= T_{\text{asignacion}} = 1 \\
T_6 &= T_{\text{asignacion}} + \sum_{i=1}^{n}(T_{\text{condicion}} + T_7 + T_{\text{incremento}}) + T_{\text{condicion}} \\
T_7 &= T_{\text{asignacion}} + \sum_{j=1}^{n}(T_{\text{condicion}} + T_8 + T_{\text{incremento}}) + T_{\text{condicion}} \\
T_8 &= T_{\text{asignacion}} + T_{\text{operacion}} = 2 \\
T_7 &= T_{\text{asignacion}} + \sum_{j=1}^{n}(1+2+2) + T_{\text{condicion}} = 1 + \sum_{j=1}^{n}(5) + 1 = 5n+2 \\
T_6 &= 1 + \sum_{i=1}^{n}(1 + (5n+2) + 2) + 1 = \sum_{i=1}^{n}(5n+5) + 2 = n(5n+5) + 2 \\
T_6 &= 5n^2 + 5n + 2 \\
T(n) &= T_4 + T_5 + T_6 = 1 + 1 + (5n^2 + 5n + 2) = 5n^2 + 5n + 4 \\
T(n) &= 5n^2 + 5n + 4
\end{aligned}
$$

Entonces $T(n) \in O(n^2)$

Figura 0.1: Grafico de la tabla



se puede poner un link a la tabla de resultados para el resto de los casos. Mencionan que pueden hacer una gráfica con los resultados y agregarla a la resolución.

Resolución ejercicio 2.3 - Grupo 2 - Sebastián, Bautista, Antonio, Albany, Luis

Código que realiza algo especifico que tiene $O(n^2)$:
a)Indique tiempo empírico con n = 10000.

```
1
2
3       public static void main(String[] args) {
4           int n = 10000;
5           int [][] matriz = new int[n][n];
6
7           // Medir el tiempo de ejecuci n
8           long inicioTiempo = System.nanoTime();
9           llenarMatriz(matriz);
10          long finTiempo = System.nanoTime();
11           // Calcular el tiempo en milisegundos
12          long tiempoEmpirico = (finTiempo - inicioTiempo) /
                1000000;
13          System.out.println("Tiempo emp rico para llenar una
                matriz de " + n + "x" + n + ": " + tiempoEmpirico + "
                ms");
14
15      }
16
17      public static void llenarMatriz(int[][] matriz) {
18          //Llenado de una matriz cuadrada con n = 10000
19          //O(n^2) por regla del producto.
20          int limite = matriz.length;
21          for (int i = 0; i < limite; i++) {
22              for (int j = 0; j < limite; j++) {
23                  matriz[i][j] = i+j;
24              }
25          }
26      }
```

Repetición	Tiempo (nanosegundos)
Prueba 1	71
Prueba 2	66
Prueba 3	51
Prueba 4	59
Prueba 5	55
Prueba 6	57
Prueba 7	56
Prueba 8	58
Prueba 9	58
Prueba 10	47
Promedio	57

4. Ejecute un algoritmo de $O(logn)$

a)Indique tiempo empírico con n = 10000.

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

Un algoritmo de $O(logn)$ que elegimos es el de búsqueda binaria, en este caso búsqueda de un entero en un arreglo ordenado. Este algoritmo sigue un principio de divisiones sucesivas donde cada iteración reduce el lote de búsqueda a la mitad hasta encontrar el elemento. Este algoritmo tiene dos peores casos:

- Búsqueda con éxito, que tardaría (log n).
- Búsqueda sin éxito que tardara [(log n)+1].

Para crear el lote de pruebas usamos un generador de randoms y para ordenar el arreglo usamos un merge-sort. El tiempo empirico de la busqueda, que calculamos haciendo el promedio de 10 pruebas diferentes, fue de 0,0044159 ms.

```
//algoritmo de busqueda binaria de un entero en un arreglo
public static boolean busquedaBinaria(int[] arr, int x) {
    int i = 0, fin = arr.length - 1;
    boolean encontro = false;

    while(i <= fin && !encontro) {
        int medio = (fin + i) / 2;
        if (arr[medio] == x) {
            encontro = true;
        } else {
            if(x < arr[medio]){
                fin = medio - 1;
            } else {
                i = medio + 1;
            }
        }
    }
    return encontro;
}
```

Prueba N°	Tiempo en ms
1	0.006764
2	0.003877
3	0.005598
4	0.004031
5	0.004062
6	0.00423
7	0.002394
8	0.005804
9	0.00568
10	0.001719

Resolución ejercicio 3.4.a - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani

Un algoritmo de $O(\log n)$ que encontramos es el ”pertenece”de un árbol AVL. La búsqueda sigue una rama específica en cada paso, disminuyendo el número de elementos a la mitad en cada nivel del árbol. Dado que el árbol AVL está balanceado, la altura máxima del árbol es $O(\log n)$. Como resultado, el número máximo de comparaciones realizadas en una búsqueda es directamente proporcional a la altura del árbol.
Imaginemos que tienes un árbol AVL con $n = 1,000,000$ nodos. Debido al balanceo del árbol, la altura del árbol es aproximadamente $\log_2(1,000,000) = 20$. Esto significa que para buscar un elemento, el algoritmo de búsqueda (‘pertenece‘) solo necesitaría comparar, en el peor de los casos, unos 20 nodos, independientemente de la posición del elemento en el árbol. Si el árbol no estuviera balanceado, podrías tener que recorrer hasta 1,000,000 de nodos en el peor caso.
Tiempo Empírico: En 10 pruebas de búsqueda de elementos, el promedio fue de 5.436 ms.

```
public boolean pertenece(Comparable elem) {
    return perteneceAux(raiz, elem);
}

private boolean perteneceAux(NodoAVL aux, Comparable elem) {
    boolean rta = false;
    if (aux != null) {
        if ((elem.compareTo(aux.getElem())) == 0) {
            rta = true;
        } else if ((elem.compareTo(aux.getElem())) < 0) {
            if (aux.getIzquierdo() != null) {
                rta = perteneceAux(aux.getIzquierdo(), elem);
            } else {
                rta = false;
            }
        } else {
            if (aux.getDerecho() != null) {
                rta = perteneceAux(aux.getDerecho(), elem);
            } else {
                rta = false;
            }
        }
    }
    return rta;
}
```

Falta describir el ambiente de prueba, como se genera el árbol qué datos tiene y con cuántos datos se probó y cuántas pruebas se hicieron con esa cantidad de nodos.

5. Especifique un algoritmo conocido (o no) que realice tenga parte de $O(1)$, parte de $O(n)$ y parte de $O(nlogn)$. Cuál es el orden de todo el algoritmo?

   a)Ejecútelo con un n=10000.

Resolución ejercicio 2.5 - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

En este algoritmo, se pueden identificar 3 partes en las que hay partes con distinta complejidad.

- **O(1):** El primer elemento del arreglo se pone en 0.
- **O(n):** Se genera el arreglo de n elementos de forma aleatoria
- **O($n * \log n$):** Se ordena el arreglo generado con el metodo de ordenamiento Quick Sort, el cual tiene un orden de complejidad $n * \log n$ y un orden de complejidad O(n^2) si ocurre el peor caso posible.

```java
package TP02;
import java.util.Random;

public class Ej2_5 {

    public static void main(String[] args) {
        long tiempoIni, tiempoFin, orden1, ordenN, ordenNlogN;
        int n=10000;

        tiempoIni = System.nanoTime();
        int[] arr = genArrAleatorios(n, 0, 10000);//Orden n
        tiempoFin = System.nanoTime();
        ordenN = tiempoFin-tiempoIni;

        tiempoIni = System.nanoTime();
        arr[0] = n;//Orden 1
        tiempoFin = System.nanoTime();
        orden1 = tiempoFin - tiempoIni;

        tiempoIni = System.nanoTime();
        quickSort(arr, 0, n-1);//Orden n*log(n)(siempre que no sea
            el peor caso)
        tiempoFin = System.nanoTime();
        ordenNlogN = tiempoFin-tiempoIni;


        System.out.println("Tiempo ejecucion Orden(1): " + orden1
           + "ns\n"
                + "Tiempo ejecucion Orden(n): " + ordenN + "ns\n"
                + "Tiempo ejecucion Orden(n*log(n)): " +
                    ordenNlogN + "ns");
    }

    public static int[] genArrAleatorios(int cantElementos, int
       min, int max){
        int[] salida= new int[cantElementos];
        Random gen = new Random();
        for(int i=0;i<cantElementos;i++){
            salida[i] = (int)(min + (gen.nextDouble()* (max-min)))
                ;
        }
        return salida;
    }

    public static void quickSort(int[] arr, int min, int max){
        if(min<max){
            int indPart = particion(arr, min, max);
            quickSort(arr, min, indPart-1);
```

```
            quickSort(arr, indPart+1, max);
        }
    }

    static void cambiar(int[] arr, int i, int j){//func aux para
        quick sort
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    static int particion(int[] arr, int min, int max){
        int pivote = arr[max];
        int i = min - 1;
        for(int j = min;j<max;j++){
            if(arr[j] <= pivote){
                i++;
                cambiar(arr, i, j);
            }
        }
        cambiar(arr, i+1, max);
        return i+1;
    }
}
```

Los resultados obtenidos por este algoritmo fueron:

- **Intento 1**:
  Tiempo ejecucion Orden(1): 200ns
  Tiempo ejecucion Orden(n): 1224500ns
  Tiempo ejecucion Orden(n*log(n)): 2041100ns
- **Intento 2**:
  Tiempo ejecucion Orden(1): 200ns
  Tiempo ejecucion Orden(n): 1186500ns
  Tiempo ejecucion Orden(n*log(n)): 2039600ns
- **Intento 3**:
  Tiempo ejecucion Orden(1): 200ns
  Tiempo ejecucion Orden(n): 1142400ns
  Tiempo ejecucion Orden(n*log(n)): 1963300ns
- **Intento 4**:
  Tiempo ejecucion Orden(1): 200ns
  Tiempo ejecucion Orden(n): 1136400ns
  Tiempo ejecucion Orden(n*log(n)): 2014100ns

Por último, el orden de todo el algoritmo, por propiedad del orden, es el orden del algoritmo de mayor complejidad. Por lo tanto, el orden del algoritmo es O($n * \log n$).

armar tabla, probar más casos, hacer alguna conclusion sobre los resultados de la prueba.

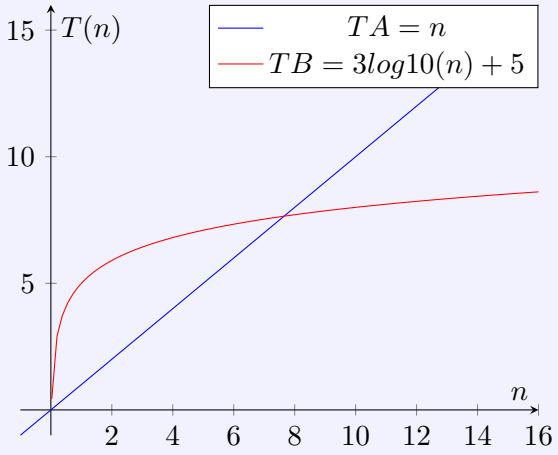
3. Notación Asintótica

1. Supongamos tener dos algoritmos para el mismo problema: el algoritmo A insume tiempo n, mientras que el algoritmo B toma tiempo $3\ log\ n + 5$. ¿Cuándo se debería elegir A y cuándo B? —

Resolución ejercicio 3.1 - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

El orden del algoritmo **A** es $O(n)$, mientras que el del algoritmo **B** es $O(\log n)$. Como $O(\log n) \subset O(n)$, podemos decir que **B** tiene un menor crecimiento asintótico que **A**, para valores grandes de n, siendo que **B** sea el mas eficiente.

Sin embargo, como se observa en el gráfico, para valores pequeños de n, en este caso cuando es aproximadamente menor a 8, el algoritmo **A** parece ser más rápido que **B**.



Analíticamente, para encontrar el valor de n, igualamos los tiempos de A y B para luego despejar n, llegando a que su valor es aproximadamente 8. Es decir:

$$n = 3\log(n) + 5$$

$$n \approx 8$$

En conclusión, para valores grandes de n, se recomienda el uso del algoritmo **B**. Para valores muy pequeños de n, podría ser conveniente utilizar **A**, aunque esto depende de detalles específicos de implementación.

mostrar como se calcularía el valor de n donde se define que una función conviene sobre la otra.

2. Demostrar las siguientes propiedades:

   a) (reflexividad) $f(n) \in O(f(n))$

Resolución ejercicio 3.2.a - Grupo 2 - Sebastián, Bautista, Antonio, Albany, Luis

Queremos demostrar que $f(n) \in O(f(n))$, es decir, que una función pertenece a la notación O de sí misma. Por definicion sabemos que:

> Decimos que una función $T(n)$ es $O(g(n))$ si existen constantes n_0 y c tales que $T(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.
>
> $$T(n) \text{ es } O(g(n)) \iff \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \mid \forall n \in \mathbb{N}, n > n_0, T(n) \leq c \cdot g(n)$$

En este caso, queremos demostrar que $f(n) \in O(f(n))$. Por lo tanto, necesitamos mostrar que existen $c > 0$ y n_0 tal que para todo $n \geq n_0$, se cumple:

$$f(n) \leq c \cdot f(n)$$

Consideremos la función $f(n)$ y tomemos $c = 1$. Entonces, la desigualdad que queremos probar se convierte en:

$$f(n) \leq 1 \cdot f(n)$$

Lo cual es equivalente a:

$$f(n) \leq f(n)$$

Luego la desigualdad es cierta para todo n, pues una función es siempre igual a sí misma. Dado que no depende de n_0, podemos elegir $n_0 = 1$, aunque cualquier valor de n_0 serviría. Por lo tanto, hemos demostrado que para $c = 1$ y cualquier n_0, se cumple que:

$$f(n) \leq c \cdot f(n)$$

Por lo tanto, esto prueba que $f(n) \in O(f(n))$ y demostramos la propiedad de reflexividad.

b) (transitividad) $f(n) \in O(g(n))\ y\ g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

Queremos demostrar la propiedad de transitividad
Para esto asumimos que $f(n) \in O(g(n))\ y\ g(n) \in O(h(n))$ es verdadero **(1)**
Y queremos llegar a que $f(n) \in O(h(n))$ también lo sea **(2)**

Reescribiremos a (1) utilizando la definición de orden: ●●

- $f(n)$ es $O(g(n)) \Leftrightarrow \exists c_0 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_0, f(n) \leq c_0 \cdot g(n)$
- $g(n)$ es $O(h(n)) \Leftrightarrow \exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_1, g(n) \leq c_1 \cdot h(n)$

Y haremos lo mismo con (2): ●●

- $f(n)$ es $O(h(n)) \Leftrightarrow \exists c_2 \in \mathbb{R}^+, \exists n_2 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_2, f(n) \leq c_2 \cdot h(n)$

Utilizando las definiciones de (1) sabemos que: ▷▷

- $f(n) \leq c_0 \cdot g(n)$ y $g(n) \leq c_1 \cdot h(n)$

Reemplazando $g(n)$ por $c_1 \cdot h(n)$ la inecuación se seguirá cumpliendo y obtenemos: ▷▷

- $f(n) \leq c_0 \cdot c_1 \cdot h(n)$

Con esto llegamos a que ▷▷

- $\exists c_2 = c_0 \cdot c_1 \in \mathbb{R}^+, \exists n_2 = max(n_0, n_1) \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_2, f(n) \leq c_2 \cdot h(n)$

Obs: sabemos que $n_2 = max(n_0, n_1)$ debido a que el orden de f y g se cumplen a partir de uno de esos valores, y si superamos ambos números entonces ambos órdenes se cumpliran

Podemos ver que lo obtenido es lo igual que la parte derecha de la definición en (2)
Y como esto es un "si y solo si", podemos concluir que la parte izquierda también se cumple

$\therefore$ Asumiendo que (1) se cumple, llegamos a que (2) también debe cumplirse
Es decir, que si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$ **(1)** $\Rightarrow f(n) \in O(h(n))$ **(2)**

c) $O(f(n)) = O(g(n)) \Leftrightarrow f(n) \in O(g(n))$ y $g(n) \in O(f(n))$

Resolución ejercicio 3.2.c - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

Debemos demostrar que: $O(f(n)) = O(g(n)) \leftrightarrow f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$
Primera parte de la demostración:
En esta parte debemos mostrar que:
$f(n) \in O(g(n)) \wedge g(n) \in O(f(n)) \rightarrow O(g(n)) = O(f(n))$
Nuestra hipótesis es: $f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$
Nuestra Tesis es: O(f (n)) = O(g(n))
Partiendo de la hipótesis:
1.1. $f(n) \in O(g(n))$
$\wedge$
1.2. $g(n) \in O(f(n))$
2.1. $\exists c \in \mathbf{R}^+, \exists n_0 \in \mathbf{N} | \forall n > n_0, f(n) \leq c \cdot g(n)$ (Por definición de 1.1)
2.2. $\exists d \in \mathbf{R}^+, \exists n_1 \in \mathbf{N} | \forall n > n_1, g(n) \leq d \cdot f(n)$ (Por definición de 1.2)
3.1. $f(n) \leq c \cdot g(n), \forall n > n_0$ (simplificamos 2.1)
3.2. $g(n) \leq d \cdot f(n), \forall n > n_0$ (simplificamos 2.2)
4. $\frac{1}{d} \cdot g(n) \leq f(n), \forall n > n_0, \frac{1}{d} = c'$ (Multiplicamos 1/d en ambos términos de 3.2)
5. $c \cdot c` \cdot g(n) \leq c \cdot f(n), \forall n > n_0, c \cdot c' = c''$ (Multiplicamos por c en ambos términos de 4)
6.1. $f(n) \leq c \cdot g(n), \forall n > n_0$ (Traemos paso 3.1)
6.2. $c'' \cdot g(n) \leq c \cdot f(n), \forall n > n_0$ (Traemos paso 5)

7. $f(n) \leq c \cdot g(n) \leq c'' \cdot g(n) \leq c \cdot f(n), \forall n > n_0, n1$ Por pasos 6.1 y 6.2
8. $O(f(n)) = O(g(n))$ Finalmente llegamos a la tesis
Por lo tanto, $O(f(n)) \in O(g(n)) \land g(n) \in O(f(n)) \rightarrow O(g(n)) = O(f(n))$
Segunda parte de la demostración:
$O(g(n)) = O(f(n)) \rightarrow O(f(n)) \in O(g(n)) \land g(n) \in O(f(n))$
Nuestra hipótesis es: O(f(n))=O(g(n))
Nuestra tesis es: $f(n) \in O(g(n)) \land g(n) \in O(f(n))$
Partiendo de la hipótesis:
1.1. O(f(n))
=
1.2. O(g(n))
2.1. $f : \mathbf{N} \rightarrow [0, \infty) | \exists c \in \mathbf{R}, c > 0, \exists n_0 \in \mathbf{N} : f(n) \leq c \cdot g(n), \forall n \geq n_0]$ Def. de omicron de 1.1
2.2. $g : \mathbf{N} \rightarrow [0, \infty) | \exists d \in \mathbf{R}, d > 0, \exists n_1 \in \mathbf{N} : g(n) \leq d \cdot f(n), \forall n \geq n_1$ Def. de omicron de 1.2
3.1. $f(n) \leq c \cdot g(n), \forall n \geq n_0$ Simplificamos 2.1
$\land$
3.2. $g(n) \leq d \cdot f(n), \forall n \geq n_1$ Simplificamos 2.2
4.1. $\exists c \in \mathbf{R}^+, \exists n_0 \in \mathbf{N} | \forall n \in \mathbf{N}, n > n_0, f(n) \leq c \cdot g(n)$ Por definición
4.2. $\exists d \in \mathbf{R}^+, \exists n_1 \in \mathbf{N} | \forall n \in \mathbf{N}, n > n_1, g(n) \leq d(n)$ Por definición
5. $f(n) \in O(g(n)) \land g(n) \in O(f(n))$ Finalmente llegamos a la tesis
Por lo tanto, $O(g(n)) = O(f(n)) \rightarrow O(f(n)) \in O(g(n)) \land g(n) \in O(f(n))$
Finalmente se demostró:
$O(f(n)) = O(g(n)) \leftrightarrow f(n) \in O(g(n)) \land g(n) \in O(f(n))$

mejorar la notación, revisar los pasos para que quede más clara la demostración, intentar con inclusión de conjuntos

d) $f(n) \in O(h(n))\ y\ g(n) \in O(h(n)) \Rightarrow (f + g)(n)) \in O(h(n))$

Resolución ejercicio 3.2.d - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani

Demostraremos la siguiente propiedad:

$$f(n) \in O(h(n))\ y\ g(n) \in O(h(n)) \Rightarrow (f + g)(n) \in O(h(n))$$

Demostración:
Sabemos que $f(n) \in O(h(n))$ y $g(n) \in O(h(n))$. Esto significa que existen dos constantes positivas $c_1 > 0$ y $c_2 > 0$, y dos valores $n_1 \geq 0$ y $n_2 \geq 0$ pertenecientes a los naturales tales que:

$$f(n) \leq c_1 \cdot h(n) \quad \text{para todo } n \geq n_1$$
$$g(n) \leq c_2 \cdot h(n) \quad \text{para todo } n \geq n_2$$

(1) Queremos demostrar que $f(n) + g(n) \in O(h(n))$. Para esto, consideramos la suma de $f(n)$ y $g(n)$:

$$f(n) + g(n) \leq c_1 \cdot h(n) + c_2 \cdot h(n)$$

(2) Podemos factorizar $h(n)$ en la expresión anterior:

$$f(n) + g(n) \leq (c_1 + c_2) \cdot h(n)$$

(3) Definimos una nueva constante $c = c_1 + c_2$ y sabemos que $c > 0$, entonces:

$$f(n) + g(n) \leq c \cdot h(n)$$

Entonces:

$$f(n) + g(n) \in O(h(n))$$

Por lo tanto queda demostrado que:

$$f(n) \in O(h(n)) \; y \; g(n) \in O(h(n)) \Rightarrow (f + g)(n) \in O(h(n))$$

Ejercicio 3.2.d - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

Demostraremos por método directo **H** $\Rightarrow$ **T** la siguiente propiedad:

$f(n) \in O(h(n))\ y\ g(n) \in O(h(n)) \Rightarrow (f+g)(n) \in O(h(n))$

H: $\underbrace{f(n) \in O(h(n))}_{\textbf{(1)}}\ y\ \underbrace{g(n) \in O(h(n))}_{\textbf{(2)}}$

T: $\underbrace{(f+g)(n) \in O(h(n))}_{\textbf{(3)}}$

Por def. orden exacto en **(1) y (2)**:

$f(n) \in O(h(n)) \Leftrightarrow \exists c_0 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}\ \forall n \in \mathbb{N}, n > n_0, f(n) \leq c_0 \cdot h(n)$
$g(n) \in O(h(n)) \Leftrightarrow \exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}\ \forall n \in \mathbb{N}, n > n_1, g(n) \leq c_1 \cdot h(n)$

Por def. orden exacto en **(3)**:

$(f+g)(n) \in O(h(n)) \Leftrightarrow \exists c_2 \in \mathbb{R}^+, \exists n_2 \in \mathbb{N}\ \forall n \in \mathbb{N}, n > n_2, (f+g)(n) \leq c_2 \cdot h(n)$

Desarrollo de la prueba, partiendo desde **H[(1) y (2)]**:

$f(n) \leq c_0 \cdot h(n)\ y\ g(n) \leq c_1 \cdot h(n) \Rightarrow$

$f(n) + g(n) \leq c_0 \cdot h(n) + c_1 \cdot h(n) \Rightarrow$

$f(n) + g(n) \leq \underbrace{(c_0 + c_1)}_{\textbf{c2}} \cdot h(n) \Rightarrow$

$(f+g)(n) \leq c2 \cdot h(n)$

Luego, $\exists c_2 = (c_0 + c_1) \in \mathbb{R}^+, \exists n_2 \in \mathbb{N}\ \forall n \in \mathbb{N}, n > n_2, (f+g)(n) \leq c_2 \cdot h(n)$

$$\therefore f(n) \in O(h(n))\ y\ g(n) \in O(h(n)) \Rightarrow (f+g)(n) \in O(h(n)).$$

e) $f(n) \in O(h(n))\ y\ g(n) \in O(l(n)) \Rightarrow (f * g)(n)) \in O(h(n) * l(n))$

Ejercicio 3.2.E G7: Ovaillos, Medel Severini, Fernandez, Occhi, Perez

Demostrar:
$f(n) \in O(h(n))\ y\ g(n) \in O(l(n)) \Rightarrow (f \cdot g)(n) \in O(h(n) \cdot l(n))$

Por Hipótesis:

(1): $f(n) \in O(h(n)) \Rightarrow$ Por definición de orden exacto se obtiene que:

$$f(n) \in O(h(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N},\ \forall n \in \mathbb{N} :\ n > n_0, f(n) \leq c \cdot h(n)$$

(2): $g(n) \in l(h(n)) \Rightarrow$ Por definición de orden exacto se obtiene que:

$$g(n) \in O(l(n)) \Leftrightarrow \exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N},\ \forall n \in \mathbb{N} :\ n > n_1, g(n) \leq c_1 \cdot l(n)$$

$Tesis$:
(3) :(f $\cdot g)(n) \in O(h(n) \cdot l(n)) \Rightarrow$ Por definición de orden exacto se obtiene que:

$$(f \cdot g)(n) \in O(h(n) * l(n)) \Leftrightarrow \exists c2 \in \mathbb{R}^+, \exists n_2 \in \mathbb{N},\ \forall n \in \mathbb{N} :\ n > n_2,$$
$$\text{(f} \cdot g)(n) \leq c_2 \cdot (h(n) \cdot l(n))$$

Demostración por (1) y (2):

$f(n) \leq c \cdot h(n)\ y\ g(n) \leq c_1 \cdot l(n) \Rightarrow$ Multiplicando miembro a miembro se obtiene:

$f(n) \cdot g(n) \leq c \cdot h(n) \cdot c_1 \cdot l(n) \Rightarrow$ Por propiedad conmutativa del producto

$f(n) \cdot g(n) \leq \underbrace{c \cdot c_1}_{(c_2)} \cdot h(n) \cdot l(n) \Rightarrow \exists c_2 \in \mathbb{R}^+, c_2 = c \cdot c_1$

$f(n) \cdot g(n) \leq c_2 \cdot h(n) \cdot l(n) \Rightarrow$ Por propiedad de factor común y propiedad de asociatividad

$$\exists n_2 = max(n_0, n_1) \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_2/$$

$(f \cdot g)(n) \leq c_2 \cdot (h(n) \cdot l(n))$ **(3)**

$\therefore$ Como desde (1) y (2) se logra llegar a (3), queda demostrado que:
$f(n) \in O(h(n))\ y\ g(n) \in O(l(n)) \Rightarrow (f * g)(n) \in O(h(n) * l(n))$.

agregar en la demostración la definición final para justificar que se demuestra 3, modificar

3. Verificar que:

(a) $1000000 \in O(1)$

Resolución ejercicio 3.3.a - Grupo 2 - Sebastián, Bautista, Antonio, Albany, Luis

Queremos veriricar que $f(n) = 10000$, pertenece al orden $O(1)$, es decir, $f(n) \in O(1)$. Por definicion de Notacion O sabemos que:

$$f(n) \leq c \cdot g(n)$$

En este caso, la funcion $f(n)$ es la funcion constrante es $f(n) = 10000$ y la funcion $g(n)$ es $O(1)$, quedando:

$$10000 \leq c \cdot 1$$

Lo cual es equivalente a:

$$10000 \leq c$$

Apartir de esto podemos concluir que c tiene que ser mayor igual que 10000, por lo que tomamos un valor c = 10000 y cualquier valor de n_0, ya que la desiguialdad es independiente de n. Finalmente podemos concluir que existe un valor c (c = 10000) tal que la funcion constante 10000 es menor o igual a c * 1 para cualquier valor de n.

(b) $10n^2 \in O(n^3)$

Ejercicio 3.3.b - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

Recordando la regla del límite:
$T(n) \in O(g(n)) \iff$ Cómo demostrar que una función está en el orden de otra, a partir de la regla del límite:

Sea $f, g : \mathbb{N} \to \mathbb{R}^+$ se cumple que:

a) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}$, entonces $f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$

b) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, entonces $f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$

c) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, entonces $f(n) \notin O(g(n)) \wedge g(n) \in O(f(n))$

Debemos probar que $\underbrace{10n^2}_{\mathbf{f}} \in O(\underbrace{n^3}_{\mathbf{g}})$

Calculamos el limite:
$\lim_{n\to\infty} \frac{10n^2}{n^3} = \lim_{n\to\infty} \frac{10}{n} = 0$ **(*)**

$\therefore$ por **(*)** y **(b)** entonces $10n^2 \in O(n^3)$ qdq.

(c) $2^n \in O(n^n)$

Ejercicio 3.3.C G7: Ovaillos, Medel Severini, Fernandez, Occhi, Perez Penas

Demuestre que $2^n \in O(n^n)$
Para demostrar que $2^n \in O(n^n)$ usaremos la regla del limite y averiguar el resultado de nuestro limite para analizar el orden.
En nuestro caso t(n) = 2^n y g(n) = n^n .
$\lim_{x\to\infty} \frac{2^n}{n^n} \overset{(1)}{=} \lim_{x\to\infty} (\frac{2}{n})^n \overset{(2)}{=} (\frac{2}{\infty})^\infty = (0)^\infty = 0$
Pasos:
(1): Aplicación de la propiedad de potencia:

$$\frac{a^n}{b^n} = \left(\frac{a}{b}\right)^n$$

(2): Reemplazo de n por ∞.
Por lo tanto : $\lim_{x\to\infty} \frac{2^n}{n^n} = 0$
Por la regla del limite si el resultado nos da 0 entonces $t(n) \in O(g(n)) \;\; \& \;\; g(n) \notin O(f(n))$
Que en nuestro caso significa que $2^n \in O(n^n)$ es verdadero

división por 0 es indefinido. reemplazando n no se puede. Hay que usar L'Hopital, no usar ampersand, usar and

(d) $n! \in O(n^n)$

Resolución ejercicio 3.3.d - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani

Queremos verificar que $n! \in O(n^n)$ es verdadera. Para eso definimos dos funciones:
- $g(n) = n^n$
- $f(n) = n!$

$$f(n) \in O(n^n) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_0, f(n) \leq c \cdot g(n)$$

Demostración:

$$f(n) \leq c \cdot g(n) \quad \text{es equivalente a} \quad n! \leq c \cdot n^n \quad \text{o} \quad n \cdot (n-1)! \leq c \cdot n^n$$

Dividimos ambos lados por n^n:

$$\frac{n!}{n^n} \leq c \quad \text{es equivalente a} \quad \frac{n \cdot (n-1)!}{n^n} \leq c$$

Reescribimos n^n como $n \cdot n^{n-1}$:

$$\frac{n \cdot (n-1)!}{n \cdot n^{n-1}} \leq c$$

Simplificamos n en ambos lados:

$$\frac{(n-1)!}{n^{n-1}} \leq c$$

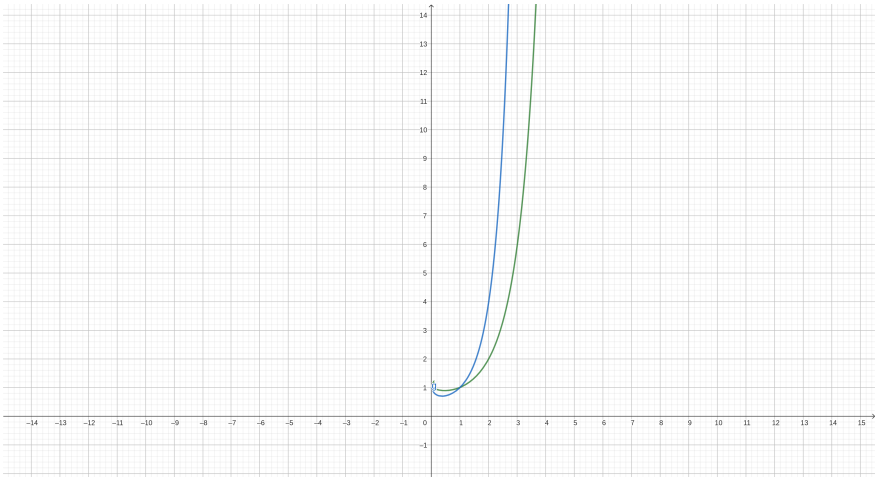
Dado que el factorial de -1 no está definido, $n \geq 1$. Entonces, sea $n_0 = 1$, para $n > n_0$, tenemos:

$$\frac{(1-1)!}{1^{1-1}} \leq c \quad \text{lo cual implica} \quad \frac{1^0}{0!} \leq c \quad \text{y} \quad \frac{1}{1} \leq c \quad \text{lo cual implica} \quad 1 \leq c$$

Conclusión: $1 \leq c$. Por lo tanto, $f(n)$ se comporta como $O(n^n)$, y es cierto que $n! \in O(n^n)$.

(e) $\sum_{k=0}^{n} k \in O(n^2)$

Figura 0.2: Grafico de las funciones



Resolución ejercicio 3.2.e - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

Debemos verificar que:

$$\sum_{k=0}^{n} k \in O(n^2)$$

Al ser una *Suma de Gauss*, sabemos que:

$$\sum_{k=0}^{n} k = \frac{n \cdot (n+1)}{2}$$

Luego, para saber si el lado derecho de la igualdad es de $O(n^2)$ calculamos el limite:

$$\lim_{n\to\infty} \frac{\frac{n\cdot(n+1)}{2}}{n^2} = \lim_{n\to\infty} \frac{n \cdot (n+1)}{2 \cdot n^2} = \lim_{n\to\infty} \frac{n+1}{2 \cdot n} = \frac{\infty + 1}{2 \cdot \infty} = \frac{\infty}{\infty}$$

Luego, como es indeterminado, aplicamos L' Hopital:

$$\lim_{n\to\infty} \frac{\frac{d}{dx} n + 1}{\frac{d}{dx} 2 \cdot n} = \lim_{n\to\infty} \frac{1}{2} = \frac{1}{2}$$

Como $\frac{1}{2} \in R$ entonces podemos afirmar que $\sum_{k=0}^{n} k \in O(n^2)$

(f) $\sum_{k=0}^{n} k^2 \in O(n^3)$

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

Sean $f(n) = \sum_{k=0}^{n} k^2$ y $g(n) = n^3$

Queremos probar que $f \in O(g)$

En primer lugar, reescribiremos a f(n):

- $f(n) = \sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$ **(1)**

Ahora queremos probar si f pertenece al orden de g, y para esto utilizaremos la definición:

- $f(n)$ es $O(g(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_0, f(n) \leq c \cdot g(n)$

Demostración:

- $f(n) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \leq n^3 + n^3 + n^3 = 3 \cdot n^3, \forall n \geq 1$ **(2)**
- En (2), tomamos $c = 3, n_0 = 1$
- Se cumple que $f(n) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \leq c \cdot g(n) = c \cdot n^3, \forall n \geq n_0$

$\therefore$ Por (1) y (2), se cumple que $\sum_{k=0}^{n} k^2 \in O(n^3)$

(g) $\sum_{k=r}^{n} k^3 \in O(n^4)$

Resolución ejercicio 3.3.g - Grupo 2 - Sebastián, Bautista, Antonio, Albany, Luis

Para demostrar que $\sum_{k=r}^{n} k^3 \in O(n^4)$, utilizamos la definición de la notación O. Queremos encontrar constantes positivas c y n_0 tales que para todo $n \geq n_0$:

$$\sum_{k=r}^{n} k^3 \leq c \cdot n^4$$

Aproximación de la suma
Primero, notemos que la suma $\sum_{k=r}^{n} k^3$ puede aproximarse por la suma completa desde $k = 1$ hasta n:

$$\sum_{k=r}^{n} k^3 \leq \sum_{k=1}^{n} k^3$$

La suma $\sum_{k=1}^{n} k^3$ es una suma conocida que se puede expresar mediante la fórmula:

$$\sum_{k=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^2(n+1)^2}{4}$$

Expando la expresión y comparo
Expandimos la expresión para obtener una forma más manejable:

$$\frac{n^2(n+1)^2}{4} = \frac{n^2(n^2+2n+1)}{4} = \frac{n^4+2n^3+n^2}{4}$$

agregar la url de la referencia de donde salió la igualdad. El resto ok

Evaluación de la cota superior
Ahora observemos que n^4 es el término dominante en esta expansión, y podemos compararlo directamente con n^4:

$$\sum_{k=r}^{n} k^3 \leq \frac{n^4+2n^3+n^2}{4} \leq \frac{n^4+2n^4+n^4}{4} = \frac{4n^4}{4} = n^4$$

Conclusión
Hemos encontrado que:

$$\sum_{k=r}^{n} k^3 \leq n^4$$

Esto cumple con la definición de $O(n^4)$, con $c = 1$ y cualquier $n_0 \geq 1$.
Por lo tanto, se puede concluir que:

$$\sum_{k=r}^{n} k^3 \in O(n^4)$$

(h) (...)= $\binom{m}{k} = 0^m a_k n^k$ ($a_k \in \mathbb{R}$), entonces $(n) \in O(n^m)$

Resolución ejercicio 3.3 (h) Forma (1) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

Sean $f(n) = (...) = \binom{m}{k} = 0^m a_k n^k \quad (a_k \in \mathbb{R})$ y $g(n) = n^m$.

En primer lugar reescribimos a f(n)

$$f(n) = \sum_{k=0}^{m} a_k n^k = a_0 n^0 + a_1 n^1 + ... + a_m n^m \textbf{ (1)}$$

Luego hay que probar que $f \in O(g)$, es decir:

$$f(n) \text{ es } O(g(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \mid \forall n \in \mathbb{N}, n > n_0, \sum_{k=0}^{m} a_k n^k \leq c \cdot n^m$$

Como n^m es el término dominante, podemos escribir la sumatoria como:

$$\sum_{k=0}^{m} a_k n^k = \sum_{k=0}^{m-1} a_k n^k + a_m n^m$$

Demostración:

- $\sum_{k=0}^{m-1} a_k n^k + a_m n^m \leq \sum_{k=0}^{m-1} a_k n^k + (a_m + 1) \cdot n^m, \forall n \geq 1$ **(2)**
- En (2), tomamos $c = (a_m + 1)$ y $n_0 = 1$
- Se cumple que $\sum_{k=0}^{m} a_k n^k \leq c \cdot n^m$

revisar esta parte de la demostración, estan diciendo que una sumatoria es menorque una constante por n a la m, revisar...

$\therefore$ Por (1) y (2), se cumple que $(n) \in O(n^m)$

Resolución ejercicio 3.3 (h) Forma (2) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

Sean $f(n) = (...) = \binom{m}{k} = 0^m a_k n^k \quad (a_k \in \mathbb{R})$ y $g(n) = n^m$.

En primer lugar reescribimos a f(n)

$$f(n) = \sum_{k=0}^{m} a_k n^k = a_0 n^0 + a_1 n^1 + ... + a_m n^m$$

Para probar que $f \in O(g)$, se utliza la propiedad:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R} \text{ entonces } f(n) \in O(g(n)) \wedge g(n) \in O(f(n)) \textbf{ (1)}$$

Demostración:

$$\begin{aligned}
\lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{a_0 n^0 + a_1 n^1 + ... + a_m n^m}{n^m} \\
&= \underbrace{\lim_{n\to\infty} \frac{a_0 n^0}{n^m} + ... + \lim_{n\to\infty} \frac{a_{m-1} n^{m-1}}{n^m}}_{\text{como el grado de } f(n) < \text{ el grado de } g(n)} + \lim_{n\to\infty} \frac{a_m n^m}{n^m} \\
&= 0 + \underbrace{\lim_{n\to\infty} \frac{a_m n^m}{n^m}}_{\text{como el grado de } f(n) = \text{ el grado de } g(n)} \\
&= 0 + a_m \\
&= a_m \quad (a_m \in \mathbb{R}) \textbf{ (2)}
\end{aligned}$$

$\therefore$ Por (1) y (2), se cumple que $(n) \in O(n^m)$

4. Analizar si cada una de las siguientes afirmaciones es verdadera o falsa, argumentando apropiadamente las respuestas dadas:

   (a) $2^{n+1} \in O(2^n)$

Resolución ejercicio 3.4 (a) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

Sea $f(n) = 2^{n+1}$

En primer lugar reescribimos a f(n)

$$f(n) = 2^{n+1} = 2 \cdot 2^n$$

Por definición de orden:

$$f(n) \in O(2^n) \Leftrightarrow \exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_1, f(n) = 2^{n+1} \leq c \cdot 2^n \quad (1)$$

$$\exists c_1 = 2 \in \mathbb{R}^+, \exists n_1 = 2 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_1, f(n) = 2 \cdot 2^n \leq 2 \cdot 2^n \quad (2)$$

$\therefore$ Por (1) y (2), se cumple que $2^{n+1} \in O(2^n)$

Resolución ejercicio 3.4.a - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani

Analizar si: $2^{n+1} \in O(2^n)$

Utilizando la regla del limite tenemos que: f(n) = 2^{n+1} y g(n) = 2^n

$$\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} \frac{2 \cdot 2^n}{2^n} = \lim_{n\to\infty} 2 = 2$$

Como $\lim_{n\to\infty} \frac{2^{n+1}}{2^n} \in \mathbb{R}$ entonces por la regla del limite puedo afirmar que:

$$2^{n+1} \in O(2^n) \wedge 2^n \in O(2^{n+1})$$
$$\therefore 2^{n+1} \in O(2^n)$$

(b) $2^{2n} \in O(2^n)$

Resolución ejercicio 3.4.b - Grupo 2 - Sebastián, Bautista, Antonio, Albany, Luis

Queremos analizar si la siguiente afirmación es verdadera o falsa: $2^{2n} \in O(2^n)$

Utilizamos la regla del limite
Sea $f, g : \mathbb{N} \to \mathbb{R}^+$ se cumple que:

a) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}$, entonces $f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$

b) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, entonces $f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$

c) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, entonces $f(n) \notin O(g(n)) \wedge g(n) \in O(f(n))$

$$\text{Sea} f(n) = 2^{2n} \quad \text{y} \quad g(n) = O(2^n), \text{con } f, g : \mathbf{N} \to \mathbf{R}^+$$

Reescribimos a $f(n) = 2^n \cdot 2^n$ y resolvemos

$$\lim_{n\to\infty} \frac{2^n \cdot 2^n}{2^n} = \lim_{n\to\infty} 2^n \tag{3}$$
$$= \infty \tag{4}$$

Referencias

- (3) Reducción de términos
- (4) Propiedad de los límites: $\lim_{x\to\infty} a^x = \infty \quad \text{con} \quad a > 1$

Luego, como se cumple que

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \quad \text{entonces} \quad f(n) \notin O(g(n)) \wedge g(n) \in O(f(n))$$

Qdq $2^{2n} \in O(2^n)$ es falsa

Ejercicio 3.4.b Grupo 7: Ovaillos, Medel Severini, Fernandez, Occhi, Perez Penas

Analizar si es verdadero o falso: $2^{2n} \in O(2^n)$:

Resuelvo por regla del limite

$$\lim_{x\to\infty} \frac{2^{2n}}{2^n} \left(\frac{\infty}{\infty}\right) \; indeterminado$$

$$\lim_{x\to\infty} \frac{2^{2n}}{2^n} \overset{(1)}{=} \lim_{x\to\infty} 2^{2n-n} = \lim_{x\to\infty} 2^n = \infty$$

Propiedades:

$$(1)\ \frac{a^n}{a^m} = a^{n-m}$$

Por Regla del limite si $\lim_{x\to\infty} \frac{f(n)}{g(n)} = \infty \; entonces \; f(n) \notin O(g(n)) \wedge g(n) \in O(f(n))$

$\therefore 2^{2n} \notin O(2^n)$

$Luego \; 2^{2n} \in O(2^n) \; es \; Falso$

(c) $O(1) \subseteq O(2^n)$

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

Sea $f(n)$ una función cualquiera de orden constante. Es decir, $f(n) \in O(1)$
Queremos probar que para cualquiera que sea $f(n)$, también será de $O(2^n)$

Por definición de orden:
●●

- $f(n) \in O(1) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_0, f(n) \leq c \cdot 1$ **(1)**

Ahora queremos ver si esta función también pertenece a $O(2^n)$, es decir que: ●●

- $f(n) \in O(2^n) \Leftrightarrow \exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}, \forall n \in \mathbb{N}, n > n_1, f(n) \leq c_1 \cdot 2^n$ **(2)**

Por (1) podemos ver que $f(n) \leq c \cdot 1$. Si en (2) reemplazamos a c_1 por c obtenemos una inecuación que es verdadera (debido a que $1 \leq 2^n, \forall n \in \mathbb{N}$)
Elegimos a $c_1 = c$ y $n_1 = n_0$ (pues en (1) $f(n)$ cumple el orden constante a partir de ese valor) y observamos que se cumple (2)
$f(n) \leq c \cdot 1 \leq c \cdot 2^n$ **(3)**

$\therefore$ en (3) demostramos que para cualquier función $f(n) \in O(1) \Rightarrow f(n) \in O(2^n)$
O lo que es equivalente, $O(1) \subseteq O(2^n)$

Ejercicio 3.4.c - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

Debemos analizar si la siguiente afirmación es **V** o **F**
$O(1) \subseteq O(n^2)$

Por definición de Orden:
$O(1) = \{t : \mathbb{N} \to [0, \infty) | \exists c_0 \in \mathbb{R}, c_0 > 0, \exists n_0 \in \mathbb{N} : t(n) \leq c_0 \cdot 1, \forall n \geq n_0\}$
Y además:
$O(n^2) = \{s : \mathbb{N} \to [0, \infty) | \exists c_1 \in \mathbb{R}, c_1 > 0, \exists n_1 \in \mathbb{N} : s(n) \leq c_1 \cdot n^2, \forall n \geq n_1\}$

Sabemos que:
$t(n) \leq c_0 \cdot 1$, considerando cualquier $c_0 \leq c_1$ se cumple que
$t(n) \leq c_0 \cdot 1 \leq c_1 \cdot n^2$, luego
$t(n) \leq c_1 \cdot n^2, \forall n > n_1$

$$\therefore O(1) \subseteq O(n^2)$$

revisar la demostración para no ir de lo particular a lo general. Se sugiere probar por definición y no por regla del límite.

(d) Si $f(n) \in O(g(n))$, entonces $log\ f(n) \in O(log\ g(n))$

Resolución ejercicio 3.4.d - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

Dada la **propiedad de limite**:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R} \Rightarrow f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$$

El **enunciado**:

$$f(n) \in O(g(n)) \Rightarrow log(f(n)) \in O(log(g(n)))$$

Para que se cumpla dicha implicación:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R \text{ y el } \lim_{n\to\infty} \frac{log(f(n))}{log(g(n))} \in R$$

Demostración por contraejemplo:

Siendo $f(n) = 5$ y $g(n) = 1$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5}{1} = 5 \tag{5}$$

$$\lim_{n\to\infty} \frac{log(f(n))}{log(g(n))} = \lim_{n\to\infty} \frac{log(5)}{log(1)} = \lim_{n\to\infty} \frac{0{,}69}{0} = Indefinido \tag{6}$$

Luego, por (1), $5 \in R$ pero por (2) $Indefinido \notin R$

$\therefore$ por contraejemplo, queda demostrado que no se cumple.

4. Algoritmia en Algoritmos iterativos sencillos

1. Para cada una de las siguientes funciones:

   (a) averiguar el tiempo teórico de cada una de ellas. Expresar la respuesta como función de n

   (b) ¿qué es lo que calculan? Expresar la respuesta en función de n.

   (i)

```
1  int f (int n){
2      int sum=0;
3      for(int i=1;i<=n;++i)
4          sum+=i;
5      return sum;
6  }
```

Ejercicio 4.1.i - Grupo 7: Ovaillos, Medel Severini, Fernandez, Occhi, Perez Penas

(a) Tiempo teórico
$T = T_2 + T_{for} + T_5$
$T_2 = 1_{asig} = 1$
$T_{for} = T_{ini} + c(T_{cond} + T_{bucleInterior} + T_{incr}) + T_{cond}$

Desarrollando el tiempo del bucle for:
$T_{ini} = 1_{asig} = 1$
$c = n$
$T_{cond} = 1_{opLog} = 1$
$T_{bucleInterior} = 1_{opMat} + 1_{asig} = 1 + 1 = 2$
$T_{incr} = 1_{opMat} + 1_{asig} = 1 + 1 = 2$

De esta forma, el tiempo del for es:
$T_{for} = 1 + n(1 + 2 + 2) + 1 = 1 + 5n + 1 = 5n + 2$
$T_5 = 1_{retorno} = 1$

El tiempo total es de:
$T = T_2 + T_{for} + T_5 = 1 + (5n + 2) + 1 = 5n + 4$

(b) El algoritmo calcula la suma de los primeros n números enteros positivos, lo cual puede expresarse como:

$$f(n) = \sum_{i=1}^{n} i = \frac{(n)(n+1)}{2}$$

(ii)

```
1 int g(int n){
2     int sum=0;
3     for(int i=1;i<n;++i)
4         sum+=i+f(n);
5     return sum;
6 }
```

Ejercicio 4.1.ii - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

(a) Tiempo teórico:
$T_T = T_2 + T_{for} + T_5$

Desarrollando:
$T_2 = 1$

$T_{for} = T_{ini} + c \cdot (T_{cond} + T_{bucle} + T_{inc}) + T_{cond}$
$T_{ini} = 1$
$T_{cond} = 1$
$T_{bucle} = 1 + 1 + 1 + T_{f(n)} = 3 + T_{f(n)}$
$T_{inc} = 1 + 1 = 2$
$c = n - 1$
$T_{for} = 1 + (n-1) \cdot (1 + T_{f(n)} + 3 + 2) + 1 = (n-1) \cdot (T_{f(n)} + 6) + 2$

$T_5 = 1$

$T_T = 1 + (n-1) \cdot (T_{f(n)} + 6) + 2 + 1$

Reemplazamos f(n)
$T_T = 1 + (n-1) \cdot (5 \cdot n + 4 + 6) + 2 + 1$
$T_T = (n-1) \cdot (5 \cdot n + 10) + 4$
$T_T = 5 \cdot n^2 + 10 \cdot n - 5 \cdot n - 10 + 4$
$T_T = 5 \cdot n^2 + 5 \cdot n - 6$

Luego, tiempo total teorico:
$T_T = 5n^2 + 5n - 6$

(b) El código calcula la sumatoria:

$$g(n) = \sum_{i=1}^{n-1} (i + \frac{(n)(n+1)}{2})$$

(iii)

```
int h(int n){
    return f(n)+g(n);
}
```

Resolución ejercicio 4.1.iii - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

a. Sean $T_{f(n)}$ y $T_{g(n)}$ los tiempos de $f(n)$ y $g(n)$ respectivamente. Y por los ejercicios 4.1.i y

4.1.ii sabemos que:

$$\begin{aligned} T_{f(n)} &= 5n + 4 \\ T_{g(n)} &= 4 + (n-1)(T_{f(n)} + 3) \\ &= 4 + (n-1)(5n + 4 + 3) \\ &= 4 + 5n^2 + 7n - 5n - 7 \\ &= 5n^2 + 2n - 3 \end{aligned}$$

Luego, el tiempo teórico $T_{h(n)}$ de $h(n)$ es:

$$\begin{aligned} T_{h(n)} &= T_{f(n)} + T_{g(n)} + T_{return} \\ &= 5n + 4 + 5n^2 + 2n - 3 + 1 \\ &= 5n^2 + 7n + 2 \end{aligned}$$

b. La función calcula la suma de las dos funciones $f(n)$ y $g(n)$.

$$h(n) = f(n) + g(n)$$

2. Demostrar formalmente si existe relación de pertenencia entre $f(n)$ y $O(g(n))$ y también entre $g(n)$ y $O(f(n))$ considerando $f(n) = T(n)$ y $g(n) = n^3$, donde $T(n)$ es la función resultante del ejercicio *2.1.a.iii* anterior.

Resolución ejercicio 4.2 - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani

Sean:

- $f(n) = 5n^2 + 10n - 2$
- $g(n) = n^3$

Para probar que $f \in O(g(n))$, y $g \in O(f(n))$ utilizamos la regla del limite:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R} \text{ entonces } f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$$

Desarrollo:
Evaluando el limite:

$$\lim_{n\to\infty} \frac{5n^2 + 10n - 2}{n^3} = \frac{\infty}{\infty}\ indeterminado$$

Transformamos la expresión realizando factor común n^3

$$\lim_{n\to\infty} \frac{5n^2 + 10n - 2}{n^3} = \lim_{n\to\infty} \frac{n^3 * (\frac{5}{n} + \frac{10}{n^2} - \frac{2}{n^3})}{n^3} =$$

$$\lim_{n\to\infty} \frac{5}{n} + \frac{10}{n^2} - \frac{2}{n^3} = \lim_{n\to\infty} \frac{5}{n} + \lim_{n\to\infty} \frac{10}{n^2} - \lim_{n\to\infty} \frac{2}{n^3} = 0 + 0 - 0 = 0$$

$$\therefore \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \text{ entonces } f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$$

3. Demostrar formalmente si existe relación de pertenencia entre $f(n)$ y $O(g(n))$ y también entre $g(n)$ y $O(f(n))$ considerando $f(n) = T(n)$ y $g(n) = log\ n$, donde $T(n)$ es la función resultante del ejercicio *2.1.a.i* anterior.

Resolución ejercicio 4.3 - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

Siendo $f(n)$ = 5n + 4 y $g(n)$ = log n debemos demostrar que:

$$f(n) \in O(g(n)) \text{ y } g(n) \in O(f(n))$$

Verificamos si $g(n) \in O(f(n))$**:**

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{log \cdot n}{5n+4} = \frac{\infty}{\infty}$$

Es indeterminado, aplicamos L´Hopital derivando ambas funciones:

$$\lim_{n\to\infty} \frac{(log \cdot n)'}{(5n+4)'} = \lim_{n\to\infty} \frac{0}{5} = 0$$

$\therefore$ Por regla del limite $g(n) \in O(f(n))$ y $f(n) \notin O(g(n))$

Regla del limite:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$$