Laboratorio Nro. 2

Notación O grande

David Felipe Garcia Contreras

Universidad Eafit

Medellín, Colombia

dfgarciac1@eafit.edu.co

Esteban Gonzalez Tamayo

Universidad Eafit

Medellín, Colombia

egonzalezt@eafit.edu.co

2)

2.1.1

El ejercicio countEvents tiene como objetivo analizar cada número contenido en el arreglo y determinar si los valores ingresados son múltiplos de 2, por lo cual se emplea el modulador para determinar el valor de este, el método emplea un ciclo for que recorrerá todo el arreglo y este tendrá una condición que analizará cada valor del arreglo y si este es válido hace que un contador aumente cada vez que la condición se cumpla, si no se cumple este no habrá ningún aumento.

2.1.2

El ejercicio bigDiff busca en el arreglo el valor más grande y más pequeño que contenga el arreglo, luego el número más grande será restado con el más pequeño luego este método retorna el resultado de la diferencia, este método se empleó un ciclo y dos condiciones, que determinarán el valor más grande y el más pequeño, por lo cual habrán dos números enteros donde el valor mínimo será comparado con el valor más grande de los enteros y el más grande

será comparado con 0,este posee un ciclo for que recorrerá todos los valores del arreglo y luego comienza a comparar número por número.

2.1.3

El ejercicio lucky13 es un método que verifica cada número contenido en un arreglo sea diferente a 1 y 3, para realizar las verificaciones el método utiliza un ciclo que recorrerá todo el arreglo y una condición para detectar que el arreglo no contiene el 1 ni el 3. Si el método detecta al menos uno de los números, este automáticamente retorna falso.

2.1.4

El ejercicio sum 28 recorre todo el arreglo buscando el número dos para sumarlos, si el total de la suma es 8 el método retorna válido de lo contrario retorna falso. El método posee un ciclo y dentro de este hay una condición que valida si el número evaluado es igual a dos, una vez se termina de recorrer todo el arreglo, está comprueba si los números sumados es igual a 8.

2.1.5

El ejercicio only14 evalúa todos los elementos contenidos en el arreglo, luego este verifica por medio de una condición si los elementos son iguales a 1 o 4 ya que, si no lo son, este retorna falso ya que el ejercicio solicita que el arreglo es válido si este contiene solamente 1 o 4.

2.1.6

El ejercicio either24 evalúa todos los elementos del arreglo empleando un ciclo, donde este posee dos condiciones que comprueban si en la posición i y i+1 hay un número 2 o si en la posición i y i+1 hay un 4. el ejercicio solicita que haya solo parejas de 2 o de 4 pero no los dos a la vez por lo cual hay dos booleanos donde uno almacena si es válido que haya una pareja de 2 y el otro lo mismo, pero con 4, solamente uno de ellos debe ser verdadero, si esto se cumple el arreglo es válido.

2.1.7

El ejercicio has22 emplea un ciclo que recorre todos los elementos del arreglo y este rectifica si hay un elemento que sea igual a dos para luego comprobar si en la posición i+1 también hay un dos, si esto es válido el método retorna true, ya que el ejercicio necesita saber si cuando hay un dos, el elemento que le sigue también es un dos.

2.1.8

El ejercicio no14 es un ejercicio que necesita evaluar todos los elementos del arreglo validando que ninguno de sus elementos posee un número 1 y tampoco un 4, por lo cual en este método se emplean 3 condiciones y dos booleanos, donde la primera válida si hay un 1 convierte la

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

condición 1 en falsa y sí también hay un 4 realiza lo mismo, si ambas condiciones son falsas el ejercicio tomará el arreglo como inválido.

2.2.1

El ejercicio canBalance es un ejercicio que necesita buscar dos subgrupos de un arreglo y sumar cada subgrupo y si ambos son iguales es porque el subgrupo es válido, este realiza un recorrido por todo el arreglo empleando un ciclo anidado donde el segundo ciclo va disminuyendo a medida de que el ciclo original aumenta. Cada iteración hace que se cree un nuevo subgrupo y esto lo realizará hasta que ambos grupos sean iguales, si ninguneo es válido, este lo retorna falso.

2.2.2

El ejercicio fix 34 analiza un arreglo con n elementos, en busqueda del 3 y del 4, el objetivo del problema es tratar de juntar el 3 y el 4 ya que estos valores están separados o en una posición diferente en el arreglo, por lo cual este tendrá que encontrar el 3 y luego cuando encuentre un 4 este intercambiará su posición con el numero que esta despues del 3.

Derek Bikoff (2012) fix34.java[Source code]

2.2.3

El ejercicio countClumps es un ejercicio que calcula cuántas parejas de un mismo número hay en un arreglo, este método necesita de 2 condiciones, un contador, un boolean y finalmente un ciclo for que recorre todos los elementos del arreglo. este metodo rectifica que si en la posicion i y i+1 los elementos son iguales pero si la condición no se cumple, este rectifica si en la posición i y i+1 los elementos son iguales. cuando son iguales el contador aumentará hasta que el ciclo finalice.

2.2.4

El ejercicio linearIn se realiza con comparaciones de los dos arreglos dados en el problema, el arreglo mayor y el menor en el tamaño, para esto se debe realizar dos ciclos que comparen el valor de cada arreglo con respecto al otro y si ambos son iguales un contador aumentará, y el valor hallado será reemplazado en el arreglo pequeño. finalmente si el tamaño del arreglo es igual al contador, significa que el arreglo pequeño si está contenido dentro del arreglo mayor.

2.2.5

El ejercicio maxSpan compara el valor que está a la izquierda del arreglo y el que está a la derecha del arreglo y si son iguales la distancia que los separa es el tamaño del intervalo creado en el caso de [1, 2, 1, 1, 3] el intervalo es [1, 2, 1, 1] y su tamaño es 4 pero también hay otro intervalo y es [1, 2, 1] y su tamaño es de 3, pero este necesita el más grande que es el de

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

4, para esto se implementó un ciclo anidado que compara todos los elementos del arreglo de izquierda a derecha y viceversa hasta encontrar el intervalo.

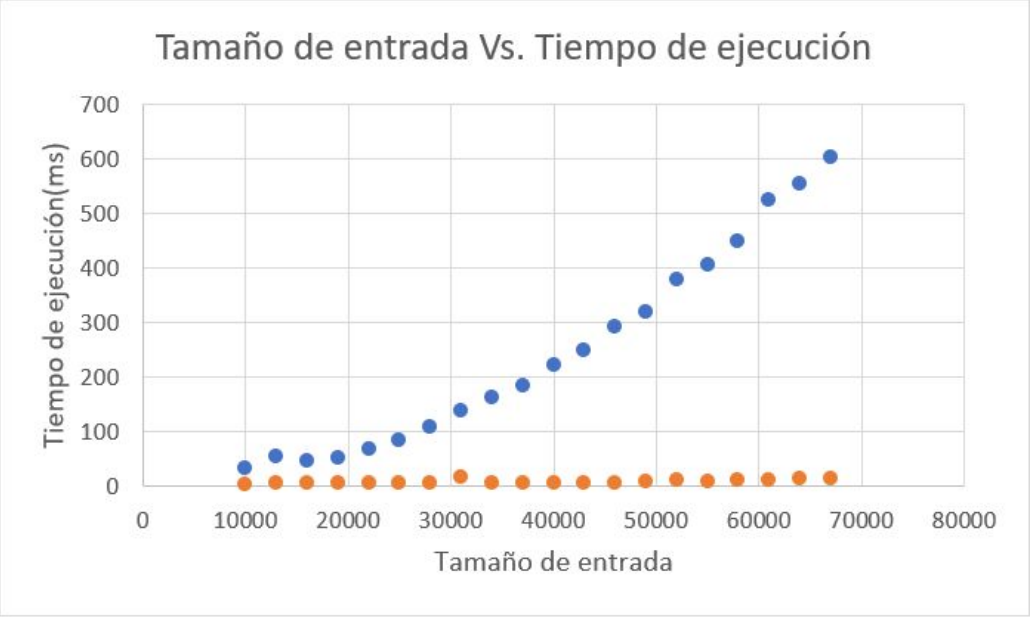
3) Simulacro de preguntas de sustentación de Proyectos

3.1

Insertion sort		Merge sort	
Dato	Tiempo(ms	Dato	Tiempo(ms)
10000	33	10000	3
13000	54	13000	5
13001	47	13001	6
13002	52	13002	6
13003	68	13003	6
13004	84	13004	5
13005	109	13005	5
13006	138	13006	16
13007	163	13007	6
13008	185	13008	7
13009	222	13009	6
13010	249	13010	6
13011	294	13011	7
13012	319	13012	9
13013	378	13013	11
13014	407	13014	10
13015	450	13015	11
13016	524	13016	11
13017	554	13017	13
13018	603	13018	13

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.2



3.3

Basándonos en la gráfica del ejercicio 3.2 se nota por mucho que el Merge sort es mucho más veloz ya que este es de complejidad logarítmica lo que significa una simplificación en el problema haciéndolo más óptimo

3.4

Basándonos en un videojuego si es recomendable el Merge sort ya que este tiene una complejidad muy buena y esta misma se respalda en la gráfica del ejercicio 3.2 por lo tanto entre la valores tome este no va a incrementar en términos de tiempo en una manera totalmente opuesta al resultado deseado

3.5

Cuando n tenga una línea de tendencia con datos muy grandes se toma como más eficiente Mergesort ya cuando n tiende a números menores entonces insertion sort es mejor por lo que su complejidad lo ayuda en tiempos menores

3.6

El ejercicio maxSpan, es un ejercicio que nos pide calcular el intervalo más grande que hay en un arreglo. el arreglo[1, 2, 1, 1, 3] tiene un intervalo [1, 2, 1, 1] ya que el 1 es el valor que coincide al extremo izquierdo y al extremo derecho y su tamaño es de 4, pero también está

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

el intervalo [1, 2, 1] con tamaño 3 por lo cual el intervalo más grande es 4. para realizar el código se implementó un dos ciclos para que dos condiciones comparen si son iguales y si el intervalo es más grande que otro ya almacenado previamente.

3.7

Array 2:

3.7.1

```
public int countEvens(int[] nums) {

int cont=0; // C1 ==O(1) si

 for(int i=0;i<nums.length;i++){ //C2+ C3n ==O(n)

 if(nums[i]%2==0)//C4n ==O(n)

 {

   cont++;//C5n== O(n)

 }

 return cont;//C6== O(1)

 }

 C1+C2+C3n+C3n+C4n+C5n+C6==O(n)
```

3.7.2

```
 public int bigDiff(int[] nums) {

   int min = Integer.MAX_VALUE; //C1== O(1)

   int max = 0; //C2== O(1)

   for(int i=0;i<nums.length;i++) //C3+C4n==O(n)

   {

     if(nums[i]>max) //C5n==O(n)

     {
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
      max=nums[i]; //C6n==O(n)
     }
    if(nums[i]<min) //C7n==O(n)
     {
      min=nums[i]; //C8n==O(n)
     }
  }
  return max-min; C9 ==O(1)
}
```

C1+C2+C3+C4n+C5n+C6nC7n+C8n+C9=O(n)

3.7.3

```
public boolean lucky13(int[] nums) {
  for(int i=0;i<nums.length;i++)C1+C2n==O(n)
  {
    if(nums[i]==1||nums[i]==3) C3n==O(n)
    {
      return false;C4n==O(n)
    }
  }
  return true;C5==O(1)
}
```

C1+C2+C3n+C4n+C5==O(n)

3.7.4

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
public boolean sum28(int[] nums) {
  int sum=0; //C1==O(1)
  for(int i=0;i<nums.length;i++) //C2 +C3n==O(n)
  {
    if(nums[i]==2) //C4n==O(n)
     {
      sum+=nums[i]; //C5n==O(n)
     }
  }
  if(sum==8) //C6 ==O(1)
  {
     return true; //C7==O(1)
  }else return false; //C8==O(1)
}
```

C1+C2+C3n+C4n+C5n+C6+C7+C8= O(n)

3.7.5

```
public boolean only14(int[] nums) {
  boolean cond = true; //C1 ==O(1)
  for(int i=0;i<nums.length;i++) //C1+C2n==O(n)
  {
    if(nums[i]!=1&&nums[i]!=4)//C3n==O(n)
     {
     cond =false; //C4n==O(n)
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
    }
  }
  return cond; C5 ==O(1)
 }
```

C1+C2n+C3n+C4n+C5=O(n)

3.7.6

```
public boolean either24(int[] nums) {
        boolean cond=false; //C1 = O(1)
        boolean cond2=false; //C2 = O(1)
        for(int i=0;i<nums.length-1;i++) //C3+C4n = O(n)
        {
           if(nums[i]==2&&nums[i+1]==2) //C5n = O(n)
           {
              cond= true; //C6n = O(n)
           }
           if(nums[i]==4&&nums[i+1]==4) //C7n = O(n)
           {
              cond2= true; //C8n = O(n)
           }
        }
        if(cond!=cond2) //C9 = O(1)
        {
           return true; //C10 = O(1)
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
        }else return false; //C11 = O(1)
    }
     C1+C2+C3+C4n+C5n+C6n+C7n+C8n+C9+C10+C11 = O(n)
```

3.7.7

```
 public boolean has22(int[] nums) {
        for(int i=0;i<nums.length-1;i++) //C1+C2n = O(n)
        {
            if(nums[i]==2&&nums[i+1]==2) //C3n = O(n)
            {
                return true; //C4n = O(n)
            }
        }
        return false; //C5 = O(1)
    }
 C1+C2n+C3n+C4n+C5 = O(n)
```

3.7.8

```
public boolean no14(int[] nums) {
        boolean cond=true; //C1 = O(1)
        boolean cond1=true; //C2 = O(1)
        for(int i=0;i<nums.length;i++) //C3+C4n = O(n)
        {
            if(nums[i]==1) //C5n = O(n)
            {
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
            cond= false; // C6n = O(n)
        }
        if(nums[i]==4) // C7n = O(n)
        {
            cond1= false; // C8n = O(n)
        }
    } //end for
    if(cond==false&&cond1==false) // C9 = O(1)
    {
        return false; // C10 = O(1)
    }
    else return true; // C11 = O(1)
}
```

C1+C2+C3+C4n+C5n+C6n+C7n+C8n+C9+C10+C11 = O(n).

3.8

3.8.1 coutEvents= N va a ser las veces que se tendrá que tomar el código para a través del arreglo nums pueda encontrar si es múltiplo de 2 por lo tanto se toma anteriormente dicho como constante ya que cada vez que se aumente el arreglo así n irá incrementado

3.8.2 bigDiff=N va a ser las veces que se tendrá que tomar el código para a través del arreglo nums pueda encontrar el mayor número del arreglo y el menor haciendo la diferencia por lo tanto se toma anteriormente dicho como constante ya que cada vez que se aumente el arreglo así n irá incrementado para el peor de los casos que no se encuentre a la primera vez

3.8.3 lucky13=n va ser las veces que se repita el ciclo para que cada número contenido en un arreglo nums sea diferente a 1 y 3 por lo tanto se toma como complejidad constante

3.8.4 sum28=n va a ser las veces que recorre todo el arreglo buscando dos números para sumarlos y por lo consecuente cumpliendose las condiciones para que la suma sea 8

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.8.5 only14= n se va tomar cuantas veces el arreglo se recorre , luego este verifica por medio de una condición si los elementos son iguales a 1 o 4

3.8.6 either24=n va a ser las veces que recorre el arreglo y a la vez el cumplimiento de la condiciones

3.8.7 has22=n va a ser las veces que recorre un arreglo y en el peor de los casos donde tenga que hacer esto varias veces

3.8.8 no14= n va a ser las veces que se recorre el arreglo y a su vez el cumplimiento de las condiciones

4) Simulacro de Parcial

4.1 *c) O(n+m)*

4.2 *d) O(m*n)*

4.3 b) O(ancho)

4.4 *b) O(n^3)*

4.5 c) O(i)

4.6 *a) T(n)=T(n-1)+c*

4.8 *i) Esta ejecuta T(n)=c + T(n-1)*

4.9 *IV)Más de nm pasos*

4.10 *III) Menos de n log n pasos*

4.11 *III) T(n)=T(n-1)+T(n-2) + c*

4.12 *II)O(m*n*log(n) + n*m^2+ n^2*log(n)+m^3)*

4.13 *c) T(n) =2T(n/2)+n*

4.14*c)O(m* raíz m+ n^3)*

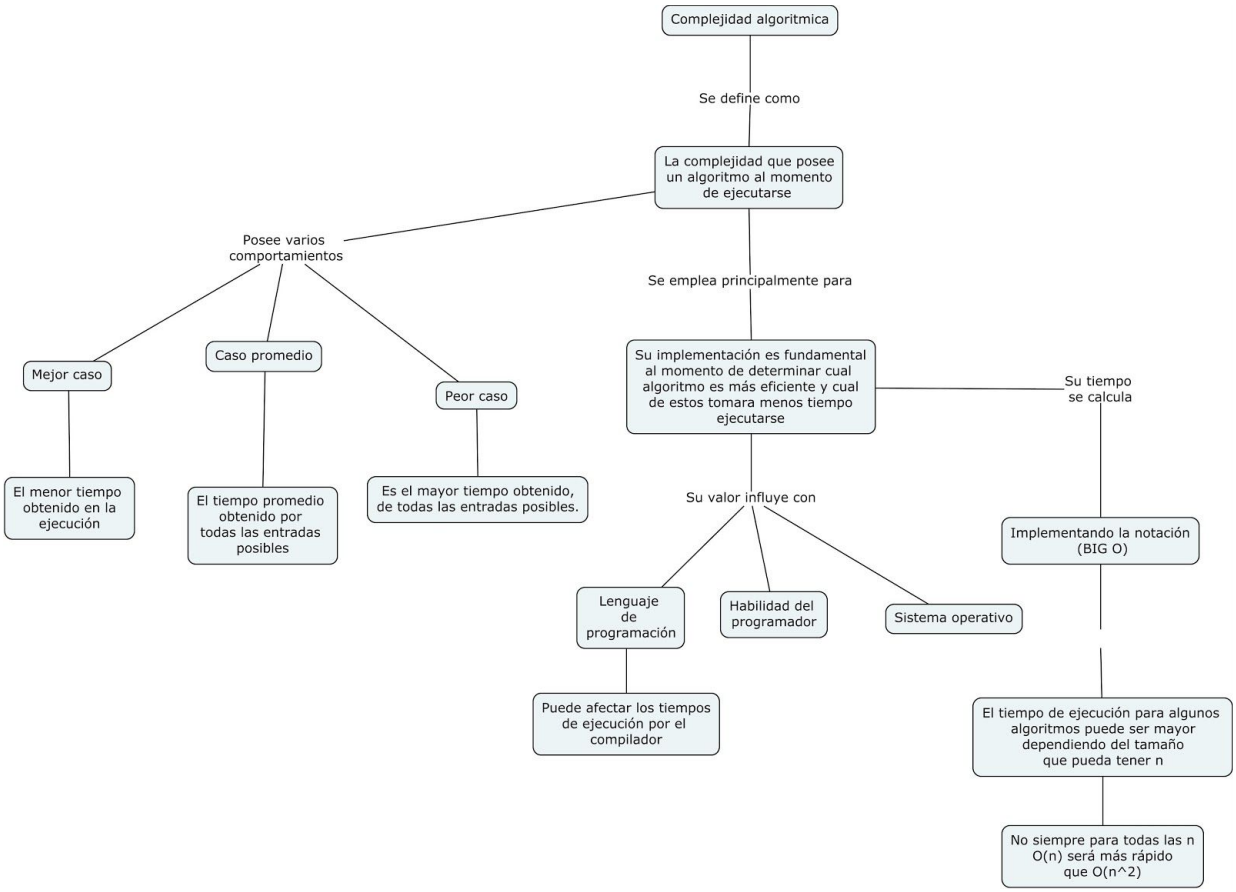
PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

5) Lectura recomendada (opcional)

Resumen:

La complejidad de un algoritmo a la hora de hacer la correspondiente notación de complejidad se puede por medio de la complejidad temporal. Está misma se trata de tomar en cuenta varias cosas, muchas personas tomarían en cuenta la velocidad del algoritmo pero esto es incorrecto ya que de esta omitiendo los distintos factores que pueden influir. Por lo tanto a la hora de tomar en cuenta la complejidad de un algoritmo se toma por medio de pasos matemáticos los cuales se pueden justificar dónde se debe cumplir la base y para n veces más.

Mapa conceptual:



PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473