Laboratorio Nro. 2
Complejidad de algoritmos

Miguel Ángel Sarmiento Aguiar
Universidad Eafit
Medellín, Colombia
msarmie4@eafit.edu.co

Marlon Perez Rios
Universidad Eafit
Medellín, Colombia
mperez@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

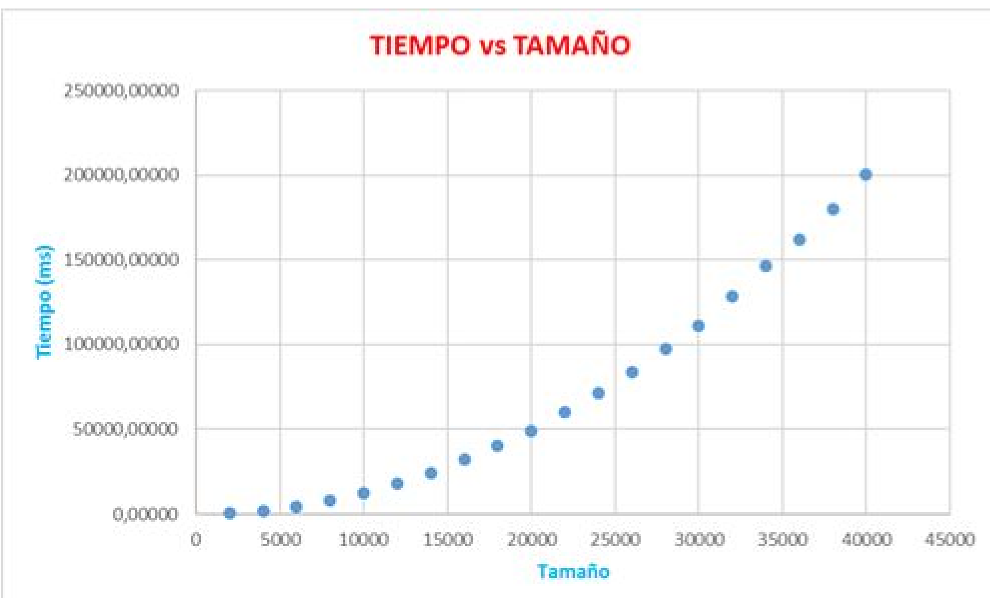
3.1 **3.1.1** insertionSort

DATO	TAMAÑO	TIEMPO (s)	TIEMPO (ms)
1	2000	0,48074	480,73562
2	4000	1,96197	1961,97482
3	6000	4,35699	4356,98716
4	8000	8,01319	8013,18572
5	10000	12,30778	12307,77938
6	12000	17,73821	17738,20540
7	14000	24,26454	24264,53970
8	16000	32,19590	32195,90360
9	18000	40,31825	40318,24759
10	20000	49,22416	49224,16489
11	22000	59,92139	59921,39125
12	24000	71,03539	71035,38815
13	26000	83,56955	83569,54892
14	28000	97,59683	97596,82977
15	30000	110,91042	110910,42346
16	32000	128,46143	128461,43376
17	34000	146,22562	146225,62327
18	36000	161,66713	161667,12556
19	38000	179,94835	179948,35151
20	40000	200,45813	200458,13183

3.1.2 mergeSort

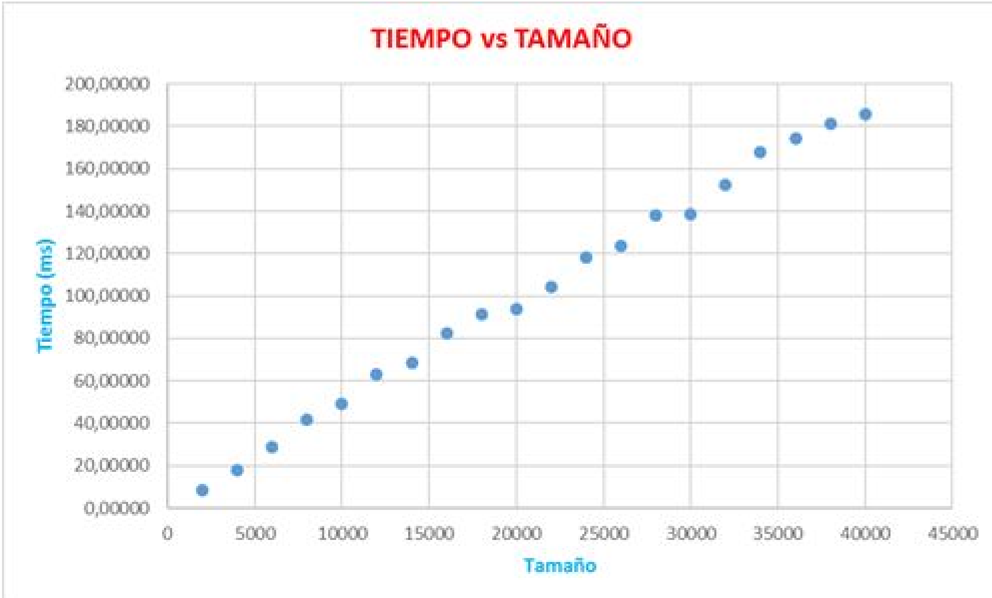
DATO	TAMAÑO	TIEMPO (s)	TIEMPO (ms)
1	2000	0,00825	8,25017
2	4000	0,01790	17,90302
3	6000	0,02882	28,82364
4	8000	0,04153	41,52949
5	10000	0,04898	48,97537
6	12000	0,06285	62,84996
7	14000	0,06825	68,24694
8	16000	0,08215	82,15107
9	18000	0,09109	91,08946
10	20000	0,09392	93,91597
11	22000	0,10427	104,26971
12	24000	0,11812	118,12490
13	26000	0,12380	123,80074
14	28000	0,13793	137,92954
15	30000	0,13832	138,31998
16	32000	0,15253	152,52609
17	34000	0,16765	167,65034
18	36000	0,17423	174,22996
19	38000	0,18112	181,11643
20	40000	0,18572	185,72159

3.2 **3.2.1** insertionSort



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

3.2.2 mergeSort



3.3 Para arreglos grandes, se puede evidenciar que es mucho más eficiente mergeSort que insertionSort, ya que con ninguno de los 20 tamaños de arreglos mergeSort tuvo una duración de más de 0.2 segundos. Por el contrario, con todos los tamaños de arreglos insertionSort duró más de dicho tiempo.

3.4 No es apropiado usar insertionSort para un videojuego con millones de elementos en una escena y demandas de tiempo real en la renderización, debido a que no sería posible actualizar dichos aspectos a una velocidad real, sino que habría que esperar mucho tiempo entre actualización y actualización, lo cual haría que el videojuego no sea realista en este aspecto.

3.5 Para arreglos grandes, insertionSort solo sería más rápido que mergeSort si el arreglo que se ingresa a insertionSort posee más datos ya ordenados que el que se le ingresa a mergeSort. Por lo tanto, para que insertionSort sea más rápido que mergeSort los datos deben de estar ya ordenados, lo cual no es realista.

3.7 **3.7.1**

- **countEvens:**

```
def countEvens(nums):
  n = 0
  for  i in range(len(nums)):    # C1*n
    if (nums[i]%2 == 0):
      n = n+1                    # C2*n
  return n                       # C3
```

T(n) = C1*n + C2*n + C3

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

T(n) = O(C1*n + C2*n + C3)
T(n) = O(n + n)
T(n) = O(n)

- **lucky13:**

```
def lucky13(nums):
  for i in range(len(nums)):                     # C1*n
    if ((nums[i] == 1) or (nums[i] == 3)):
      return False                               # C2*n
  return True                                    # C3
```

T(n) = C1*n + C2*n + C3
T(n) = O(C1*n + C2*n + C3)
T(n) = O(n + n)
T(n) = O(n)

- **isEverywhere:**

```
def isEverywhere(nums, val):
  for i in range(len(nums)-1):                         # C1*(n-1)
    if ((nums[i] != val) and (nums[i+1] != val)):
      return False                                     # C2*(n-1)
  return True                                          # C3
```

T(n) = C1*(n-1) + C2*(n-1) + C3
T(n) = O(C1*(n-1) + C2*(n-1) + C3)
T(n) = O(n-1 + n-1)
T(n) = O(n)

- **modThree:**

```
def modThree(nums):
  for i in range(len(nums)-2):                                                   # C1*(n-2)
    if ((nums[i]%2 == 0) and (nums[i+1]%2 == 0) and (nums[i+2]%2 == 0)):
      return True                                                                # C2*(n-2)
    elif ((nums[i]%2 == 1) and (nums[i+1]%2 == 1) and (nums[i+2]%2 == 1)):
      return True                                                                # C3*(n-2)
  return False                                                                   # C4
```

T(n) = C1*(n-2) + C2*(n-2) + C3*(n-2) + C4
T(n) = O(C1*(n-2) + C2*(n-2) + C3*(n-2) + C4)
T(n) = O(n-2 + n-2 + n-2)
T(n) = O(n)

- **tripleUp:**

```
def tripleUp(nums):
  for i in range(len(nums)-2):                                        # C1*(n-2)
    if ((nums[i+1] == nums[i]+1) and (nums[i+2] == nums[i]+2)):
      return True                                                     # C2(n-2)
  return False                                                        # C3
```

T(n) = C1*(n-2) + C2*(n-2) + C3
T(n) = O(C1*(n-2) + C2*(n-2) + C3)
T(n) = O(n-2 + n-2)
T(n) = 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

3.7.2

- **linearIn:**

```
def linearIn(outer, inner):
  n = 0
  for i in range(len(inner)):       # C1*n
    for j in range(len(outer)):     # C2*n*m
      if (inner[i] == outer[j]):
        n = n+1
        break                       # C3*n*m
  return n == len(inner)            # C4
```

T(n) = C1*n + C2*n*m + C3*n*m + C4
T(n) = O(C1*n + C2*n*m + C3*n*m + C4)
T(n) = O(n + n*m + n*m)
T(n) = O(n*m)

- **seriesUp:**

```
def seriesUp(n):
  array = [None]*(n*(n+1)//2)
  num = 0
  for i in range(n):            # C1*n
    for j in range(i+1):        # C2*n*(n+1)
      array[num] = j+1
      num = num+1
  return array                  # C3
```

T(n) = C1*n + C2*n*(n+1) + C3
T(n) = O(C1*n + C2*n*(n+1) + C3)
T(n) = O(n + n*(n+1))
T(n) = O(n^2)

- **fix34:**

```
def fix34(nums):
  n = 0
  for i in range(len(nums)):          # C1*n
    if (nums[i] == 3):                # C2*n
      for j in range(n, len(nums)):   # C3*n*n
        if (nums[j] == 4):
          n = j
          aux = nums[j]
          nums[n] = nums[i+1]
          nums[i+1] = aux
          break                       # C4*n*n
  return nums                         # C5
```

T(n) = C1*n + C2*n + C3*n*n + C4*n*n + C5
T(n) = O(C1*n + C2*n + C3*n*n + C4*n*n + C5)
T(n) = O(n + n + n*n + n*n)
T(n) = O(n^2)

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

- **maxSpan:**

```
def maxSpan(nums):
  cont = 0
  cont2 = 0
  for i in range(len(nums)):                  # C1*n
    for j in range(len(nums)-1, 0, -1):       # C2*n(n-1)
      if (nums[i] == nums[j]):                # C3*n(n-1)
        cont = (j-i)+1
        if (cont > cont2):                    # C4*n(n-1)
          cont2 = cont
          cont = 0
  return cont2                                # C5
```

T(n) = C1*n + C2*n*(n-1) + C3*n*(n-1) + C4*n*(n-1) + C5
T(n) = O(C1*n + C2*n*(n-1) + C3*n*(n-1) + C4*n*(n-1) + C5)
T(n) = O(n + n*(n-1) + n*(n-1) + n*(n-1))
T(n) = O(n^2)

- **squareUp:**

```
def squareUp(n):
  array = [0]*(n*n)
  p = 0
  for i in range(n):          # C1*n
    p = n*(i+1)-1
    for j in range(i+1):      # C2*n*(n+1)
      array[p] = j+1
      p = p-1
  return array                # C3
```

T(n) = C1*n + C2*n*(n+1) + C3
T(n) = O(C1*n + C2*n*(n+1) + C3)
T(n) = O(n + n*(n+1))
T(n) = O(n^2)

3.8 **3.8.1**

- **countEvens:** n es el tamaño del arreglo.
- **lucky13:** n es el tamaño del arreglo.
- **isEverywhere:** n es el tamaño del arreglo.
- **modThree:** n es el tamaño del arreglo.
- **tripleUp:** n es el tamaño del arreglo.

3.8.2

- **linearIn:** n es el tamaño del arreglo outer. m es el tamaño del arreglo inner.
- **seriesUp:** n es el tamaño del arreglo.
- **fix34:** n es el tamaño del arreglo.
- **maxSpan:** n es el tamaño del arreglo.
- **squareUp:** n es el tamaño del arreglo.

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) 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 a. O(n)

4.6 T(10000) = 10000 segundos

4.7 1, 3 y 4

4.8 d. O(2^n)

4.9 a. O(n^3)

4.10 c. Ejecuta menos de n*log(n) pasos

4.11 c. T(n) = T(n-1) + T(n-2) + C

4.12 b. 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 a. O(n^3 + n*(log(log(m)) + m*n^(1/2)

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