Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Inteligencia Artificial, Análisis y Diseño de Algoritmos
Sem: 2024-1, 3BV1, Práctica 3, 18 de octubre de 2023

Práctica 3: Funciones Recursivas vs Iterativas.

Catonga Tecla Daniel Isaí 1, Olguin Castillo Victor Manuel 2.

*daniel*9513*importantes*@*gmail.com*$_1$, *manuelevansipn*@*gmail.com*$_2$

Resumen:

Palabras Clave: Algoritmo, Big O, C++, Recursividad

1. Introducción

Un algoritmo es un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema [2]. Los algoritmos son de suma importancia, estos ayudan a la resolución de problemas muy grandes o complejos como problemas matemáticos y científicos, por lo tanto, los algoritmos son fundamentales en varias áreas de las ciencias o ingeniería.

En programación, las operaciones pueden ser manejadas por recursividad o iteraciones. Recursividad es una función que se llama a sí misma, directa o indirectamente. Una de las estructuras fundamentales de programación son los bucles, las cuales forman parte de las estructuras de la mayoría de los lenguajes, los bucles sirven para los algoritmos iterativos. [1]

El análisis de algoritmos es otro parte fundamental por lo que los algoritmos tienen que ser los más eficientes posibles en complejidad temporal y complejidad espacial. En cuestión de complejidad temporal algunos algoritmos pueden requerir años en resolver un solo problema, por lo que, no es suficiente para la práctica. , y se opta por otros algoritmos que resuelvan el problema que presenten de una forma más eficaz. Por otra parte, una maquina no cuenta con recursos infinitos de memoria o espacio de almacenamiento, por lo que, existen algoritmos que ocupan mucho espacio de memoria y esto puede ocasionar problemas para equipos que no cuentan con el espacio suficiente, entonces se tiene que analizar también lo que es la complejidad espacial para tratar de ocupar la menor cantidad de memoria posible. Ambas complejidades son importantes, pero es más importante la complejidad temporal ya que las empresas pueden comprar más memorias de almacenamiento, pero no pueden comprar tiempo, por lo que algunas empresas optan sacrificar complejidad espacial por una complejidad temporal más eficiente.

Algunos algoritmos iterativos se pueden hacer de forma recursiva y viceversa, lo importante es conocer cuál es más eficaz con respecto a la otra implementación, por lo que, es importante conocer la complejidad de cada algoritmo en la forma recursiva o iterativa y compararlas para saber que algoritmos implementar en un caso práctico.

2. Conceptos Basicos

- **Algoritmo**. Un algoritmo es una secuencia de pasos lógicos que son precisos, ordenados y finitos que se ocupan para resolver un problema deseado [3].

- **Análisis de algoritmos** Es un proceso de evaluación donde conoceremos el rendimiento y la eficiencia de un algoritmo. Se evaluará el consumo de tiempo y de recursos computacionales que requiere el algoritmo para ser ejecutado con diversos datos de entrada, y esto determinará su complejidad [3].

- **Cota superior asintótica**. Es una función que delimita por la parte superior a otra función a medida que la entrada de la función delimitada crece.

- **Cota inferior asintótica**. Es una función que delimita por la parte inferior a otra función a medida que la entrada de la función delimitada crece.

- **Notacion O**. Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota superior asintótica en el peor caso de ejecución de un algoritmo [3].

$$O(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists\ c \text{ constante positiva y } n_0 \notin \mathbb{N} :$$
$$f(n) \leq cg(n),\ \forall\ n \geq n_0\}$$

- **Notacion Tetha "Θ"** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota superior y una cota inferior, esto nos da una idea más precisa del comportamiento y complejidad del mismo [3].

$$\Theta(g(n) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists\, c_1, c_2 \text{ constantes positivas}, n_0\ :$$
$$0 < c_1 g(n) \leq f(n) \leq c_2 g(n), \forall\, n \geq n_0\}$$

- **Notacion Omega "Ω"** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota inferior asintótica en el peor caso de ejecución de un algoritmo [3].

$$\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists\, c \text{ constante positiva y } n_0\ :$$
$$0 < cg(n) \leq f(n), \forall\, n \geq n_0\}$$

- **Analisis a posteriori**. Es una evaluación que se realiza de forma empírica, donde los resultados se obtienen con la ejecución del algoritmo y la medición del tiempo y recursos computacionales que requiere [3].
- **Analisis a priori**. Es una evaluacion que se realiza de forma teorica, donde los resultados se pueden obtener con el conteo de operaciones y/o analisis matematico que en base a sus formulas se obtiene su complegidad algoritmica [3].
- **Funciones recursivas**. En la programacion las funciones recursivas son aquellas que durante su proceso se invocan asi mismas.

```
Suma_Recursiva(n):
If n == 1
    return 1
Else
    return n + sumaRecursiva(n - 1)
```

- **Funciones iterativas**. En la programacion las funciones iterativas son aquellas se ejecuta en un ciclo n numero de veces y existe una condicion de validacion en cada iteracion que controla si se itera una ves mas o finaliza en ciclo

```
Suma_iterativa(n):
resultado = 0
for i = 0 to n do:
    resultado += i
return resultado
```

- **Pseudocódigo de Búsqueda iterativo**. El algoritmo de búsqueda, retorna el índice del valor que se busca dividiendo el arreglo en 3 partes por cada operación que se realiza hasta encontrar el valor o en caso contrario retorna -1 si no existe el valor en el arreglo.

```
BusquedaIterativa(Arr[1,2,...,n],valor):
low = 0
high = longitud del vector - 1
i entero
j entro
while low <= high do:
  i = low + (high-low)/3
  j = low + 2*(high-low)/3
```

```
    if Arr[j]<valor do:
      low = j+1
    else if Arr[i]>valor do:
      high = i-1
    else if Arr[i]<valor and Arr[j]>valor do:
      low = i+1
      high = j-1
    else if Arr[i] == valor do:
      return i
    else if Arr[j] == valor do:
      return j
  return -1
```

- **Pseudocódigo de Búsqueda recursiva**. El algoritmo de búsqueda recursivo se llama así msimo varias veces, retorna el índice del valor que se busca dividiendo el arreglo en 3 partes por cada operación que se realiza hasta encontrar el valor o en caso contrario retorna -1 si no existe el valor en el arreglo.

```
BusquedaRecursiva(Arr[1,2,...,n],low,high,valor):
if low <= high do:
  i = low + (high-low)/3
  j = low + 2*(high-low)/3

  if Arr[j]<valor do:
    return BusquedaRecursiva(Arr,j+1,high,valor)
  else if Arr[i]>valor do:
    return BusquedaRecursiva(Arr,low,i-1,valor)
  else if Arr[i]<valor and Arr[j]>valor do:
    return BusquedaRecursiva(Arr,j+1,i-1,valor)
  else if Arr[i] == valor do:
    return i
  else if Arr[j] == valor do:
    retur
else:
  return -1
```

3. Experimentación y Resultados

2. Algoritmo de Búsqueda

Análisis a priori de algoritmo de Búsqueda Iterativo. El análisis a priori del algoritmo de búsqueda iterativa se basa con bloques con el algoritmo que se muestra abajo "BusquedaIterativa".

```
BusquedaIterativa(Arr[1,2,...,n],valor):
01.  low = 0
02.  high = longitud del vector - 1
03.  i entero
04.  j entro
05.  while low <= high do:
06.   i = low + (high-low)/3
07.   j = low + 2*(high-low)/3
08.   if Arr[j]<valor do:
09.     low = j+1
10.   else if Arr[i]>valor do:
11.    high = i-1
12.   else if Arr[i]<valor and Arr[j]>valor do:
13.    low = i+1
14.    high = j-1
15.   else if Arr[i] == valor do:
16.    return i
17.   else if Arr[j] == valor do:
18.    return j
19.  return -1
```

Análisis a posteriori del algoritmo de búsqueda iterativa. El algoritmo en ejecución se muestra en la imagen 5 y el número de pasos se muestra en la tabla 6. Los datos muestran un aumento logaritmico base 3 en la complejidad a medida que n crece, para la tabla de datos considera que para el peor caso el valor que se busca no se encuentra en el vector o arreglo.

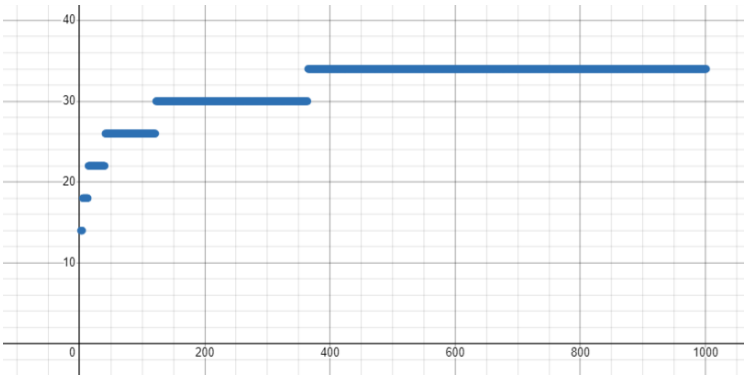
```
[2,4]
El valor 4 se encuentra en el index [1]
[2,4,6]
El valor 6 se encuentra en el index [2]
[2,4,6,8]
El valor 8 se encuentra en el index [3]
[2,4,6,8,10]
El valor 10 se encuentra en el index [4]
[2,4,6,8,10,12]
El valor 12 se encuentra en el index [5]
[2,4,6,8,10,12,14]
El valor 14 se encuentra en el index [6]
```

Imagen 5. Ejecución algoritmo de búsqueda iterativa.

Valor de n	**# de pasos**
2	14
3	14
4	14
5	18
6	18
7	18
8	18
9	18
10	18
11	18
12	18
13	18
14	22
15	22
⋮	⋮

Tabla 6. Datos de algoritmo de búsqueda iterativa.

Los datos de ejecución del algoritmo de búsqueda iterativa se muestran en la gráfica 6. El algoritmo muestra un crecimiento logaritmico base 3, demostrando que para peor caso o Big O es $O(log_3(n))$.



Gráfica 6. Grafica del comportamiento del algoritmo de búsqueda iterativa.

En la gráfica 7 se muestra la función tal que $g(n) = 7{,}5log_3(n)$ acota por arriba al algoritmo. El ajuste asíntotico es para cuando $n_0 \geq 120$. En la gráfica 8 se muestra que $f(n)$ esta delimitado por $g(n)$ cuando $n \geq n_0$.

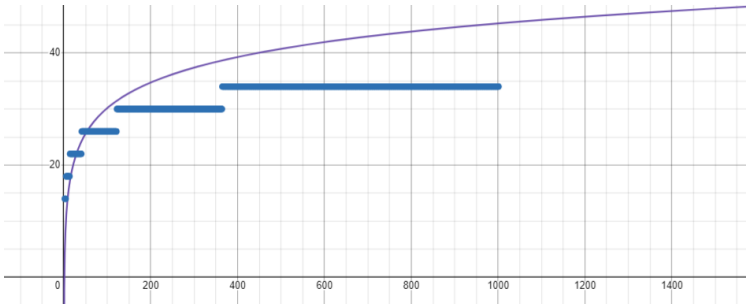


Figura 7. Gráfica con acotación para $f(n)$.

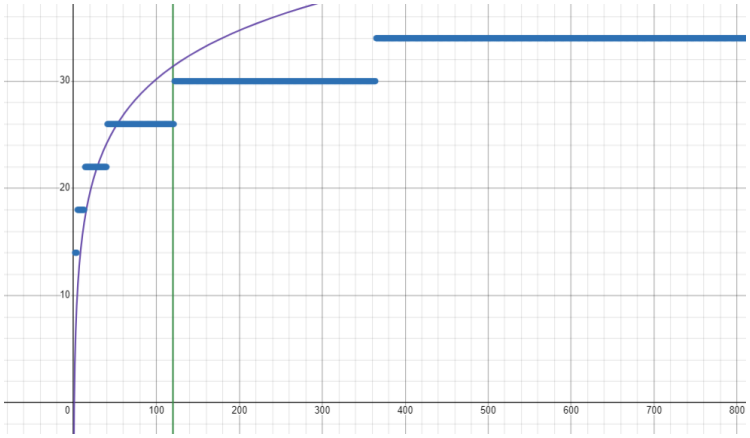


Figura 8. Gráfica para valor n_0.

$$\therefore T(n) \in O(log_3 n)$$

Análisis a priori de algoritmo de Búsqueda Recursivo. El análisis a priori del algoritmo de búsqueda recursivo se muestra abajo con respecto el algoritmo de "BusquedaRecursiva".

```
BusquedaRecursiva(Arr[1,2,...,n],low,high,valor):
01.  if low <= high do:
02.  i = low + (high-low)/3
03.  j = low + 2*(high-low)/3
04.  if Arr[j]<valor do:
05.    return BusquedaRecursiva(Arr,j+1,high,valor)
06.  else if Arr[i]>valor do:
07.    return BusquedaRecursiva(Arr,low,i-1,valor)
08.  else if Arr[i]<valor and Arr[j]>valor do:
09.    return BusquedaRecursiva(Arr,j+1,i-1,valor)
10.  else if Arr[i] == valor do:
11.    return i
12.  else if Arr[j] == valor do:
13.    return j
14.  return -1
```

$$\therefore T(n) = T(\tfrac{n}{3}) + C$$
$$\text{Sea } n = 3^k \ (k = log_3 n)$$
$$\Rightarrow T(3^k) = T(3^{k-1}) + C$$
Resolviendo mediante sustitución hacía atrás se tiene:
$$= [T(3^{k-2}) + C] + C$$
$$= T(3^{k-2}) + 2C$$
$$= [T(3^{k-3}) + C] + 2C$$
$$= T(3^{k-3}) + 3C$$
$$\vdots$$
$$= T(3^{k-i}) + iC$$
$$k - i = 0 \ \Rightarrow \ k = i$$
$$\Rightarrow$$
$$= T(3) + kC$$
$$= C + kC$$
$$= C + log_3(n)C$$
$$\therefore T(n) \in O(log_3 n)$$

Análisis a posteriori del algoritmo de búsqueda Recursivo. El algoritmo en ejecución se muestra en la imagen 5 y el número de pasos se muestra en la tabla 6. Los datos muestran un aumento logaritmico base 3 en la complejidad a medida que n crece, para la tabla de datos considera que para el peor caso el valor que se busca no se encuentra en el vector o arreglo.

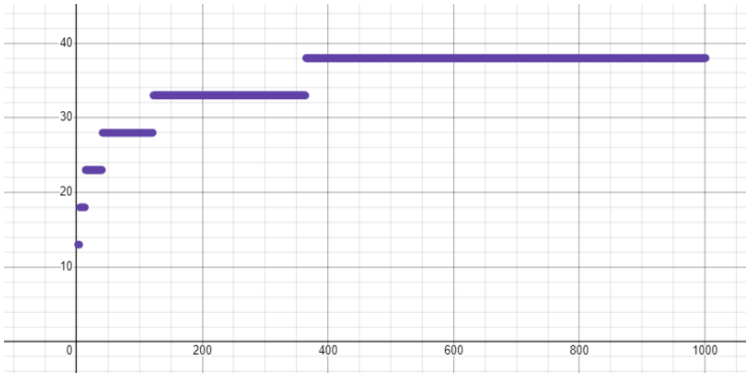
```
[2,4]
El valor 4 se encuentra en el index [1]
[2,4,6]
El valor 6 se encuentra en el index [2]
[2,4,6,8]
El valor 8 se encuentra en el index [3]
[2,4,6,8,10]
El valor 10 se encuentra en el index [4]
[2,4,6,8,10,12]
El valor 12 se encuentra en el index [5]
[2,4,6,8,10,12,14]
El valor 14 se encuentra en el index [6]
```

Imagen 5. Ejecución algoritmo de búsqueda recursiva.

Valor de n	**# de pasos**
2	13
3	13
4	13
5	18
6	18
7	18
8	18
9	18
10	18
11	18
12	18
13	18
14	23
15	23
⋮	⋮

Tabla 6. Datos de algoritmo de búsqueda recursiva.

Los datos de ejecución del algoritmo de búsqueda rescursiva se muestran en la gráfica 6. El algoritmo muestra un crecimiento logaritmico base 3, demostrando que para peor caso o Big O es $O(log_3(n))$.



Gráfica 9. Gráfica del comportamiento de 'MostrarPerfectos'.

En la gráfica 7 se muestra la función tal que $g(n) = 8log_3(n)$ acota por arriba al algoritmo. El ajuste asíntotico es para cuando $n_0 \geq 120$. En la gráfica 8 se muestra que $f(n)$ esta delimitado por $g(n)$ cuando $n \geq n_0$.

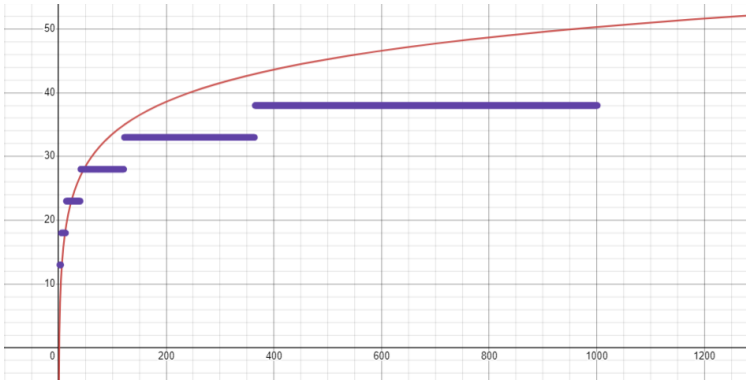


Figura 7. Gráfica con acotación para $f(n)$.

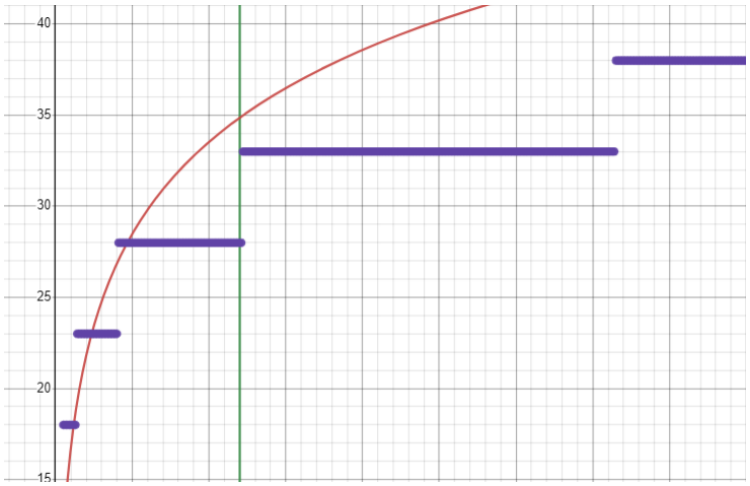


Figura 8. Gráfica para valor n_0.

$$\therefore T(n) \in O(log_3 n)$$

La comparación del algoritmo recursivo e iterativo tienen la misma complejidad temporal ya que se desmuestra en lo anterior que ambos son big $O(log_3 n)$, por lo que para la búsqueda se puede implementar tanto de la forma recursiva o de la forma iterativa ya que tienen a misma eficiencia en complejidad temporal. Otro cosa que se toma en cuenta para el análisis de la complejidad espacial, de los algoritmos anteriores solo se muestra las complejidad temporal.

4. Conclusiones

Texto ejemplo, conclusión general.

Conclusiones Catonga Tecla Daniel Isaí 1
Texto ejemplo.

Conclusiones Olguin Castillo Victor Manuel 2
Texto ejemplo.

5. Bibliografía

Referencias

[1] Gary Briceño. *Recursivida o Iteración.* 2013. URL: https://www.clubdetecnologia.net/blog/2013/recursividad-iteracion/.

[2] REAL ACADEMIA ESPAÑOLA. *Diccionario de la lengua española, 23.ª ed.* 2023. URL: https://dle.rae.es/algoritmo.

[3] R. C. t. Lee. *Introducción al diseño y análisis de algoritmos: un enfoque estratégico*. Ed. por Mc Graw Hill. 2014.