Análisis de Algoritmos, Sem: 2018-1, 3CV2 Práctica 4, 14 de Septiembre del 2017

Práctica 4: Divide y Venceras: Algoritmo QuickSort.

Luis Daniel Martinez Berumen

Escuela Superior de Computo
Instituto Politecnico Nacional
dany.berumen@gmail.com

Abstract

En esta practica vamos a demostrar, tanto de manera analitica como de manera practica con graficas de los experimentos, como los algoritmos Partition y el QuickSort tienen complejidad complejidad lineal, para el Partition y el QuickSort con complejidad θ (nlogn), todo esto con ayuda del paradigma de Divide y Venceras el cual nos ayudara a el analisis de cada uno de los algoritmos, ademas de mostrar la resolucion de 2 problemas en la parte de anexos.

Palabras Clave

- Algoritmo
- Funcion
- Orden
- Recursividad
- Paradigma

1. Introducción

Divide y Vencerás es una frase quye hemos escuchado todos, al menos, una vez en nuestra vida, para nosotros es técnica de diseño de algoritmos, siendo de gran utilidad para nuestra carrera, ya que, los problemas a los que nos enfrentamos día con día son mas faciles de resolver si aplciamos una tecnica de este tipo. De hecho, suele ser considerada una filosofía general para resolver problemas, no solo del termino informatico, sino que también se utiliza en muchos otros ámbitos

2. Conceptos Básicos

Para la correcta comprension de este trabajo, es necesario definir algunos terminos tales como θ, O y Ω.
θ(n):
Sea g(n) una función. Se define θ (g(n)) como:

$\theta(\text{g(n)}) = \{f(n) \quad | \quad \exists c1, c2 > 0 \quad \& \quad n_0 > 0 \quad | \quad \forall n >= n_0 \quad 0 <= c1g(n) <= f(n) <= c2g(n)\}$

O(n):
Sea g(n) una función, O(n) (el pero de los casos) se define como:

$$\text{O(n)}=\{f(n) \quad | \quad \exists c > 0 \quad \& \quad n_0 > 0 \quad | \quad f(n) <= Cg(n) \quad \forall n >= n_0\}$$

Ω(n):
Sea g(n) una función. Se define Ω (g(n)) (el mejor de los casos) como:

$$\Omega(\text{g(n)}) =\{f(n) \quad | \quad \exists c > 0 \quad \& \quad n_0 > 0 \quad | \quad 0 <= cg(n) <= f(n) \quad \forall n >= n_0\}$$

El ordenamiento rápido (quicksort en inglés) es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log . Esta es la técnica de ordenamiento más rápida conocida. Fue desarrollada por C. Antony R. Hoare en 1960. El algoritmo original es recursivo, pero se utilizan versiones iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos, y consumen más recursos). El algoritmo fundamental es el siguiente:

- Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados. Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.
- En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es O(n·log n).
- En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $0(n^2)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas.
- En el caso promedio, el orden es O(n·log n).

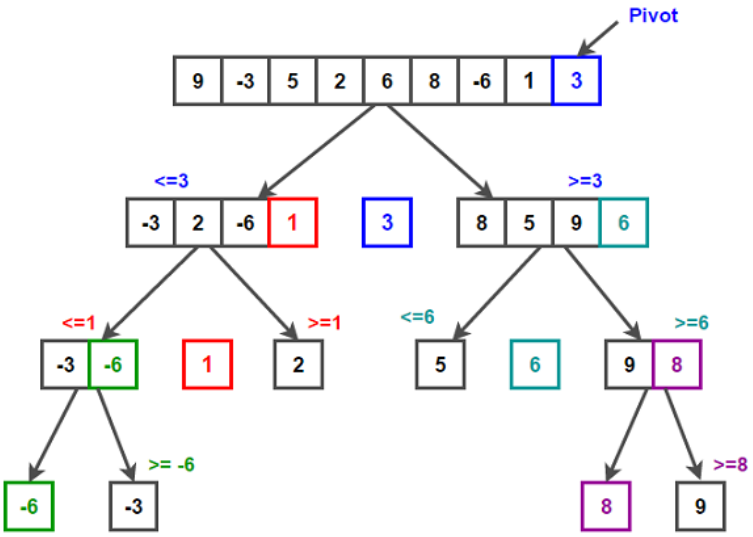

Figura 1: QuickSort

3. Experimentación y Resultados

3.1. Implemente el algortimo QuickSort

i)Mediante graficas, muestre que el algoritmo **Partition** tiene complejidad lineal.

Para empezar este ejercicio tomamos arreglos creciendo iterativamente de tamaño 1 hasta el 50, llenandolos con valores aleatorios no repetidos segun su tamaño. Para esta prueba, el algoritmo Partition arrojo los siguientes resultados

1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
15	14
16	15
17	16
18	17
19	18
20	19
21	20
22	21
23	22
24	23
25	24
26	25
27	26
28	27
29	28
30	29
31	30
32	31
33	32
34	33
35	34
36	35
37	36
38	37
39	38
40	39
41	40
42	41
43	42
44	43
45	44
46	45
47	46
48	47
49	48
50	49

Figura 2: Tabla de Partition.

Para estos resultados, se realizo una grafica confrontando los resultados tabulados anteriormente, la cual quedo de la siguiente manera.

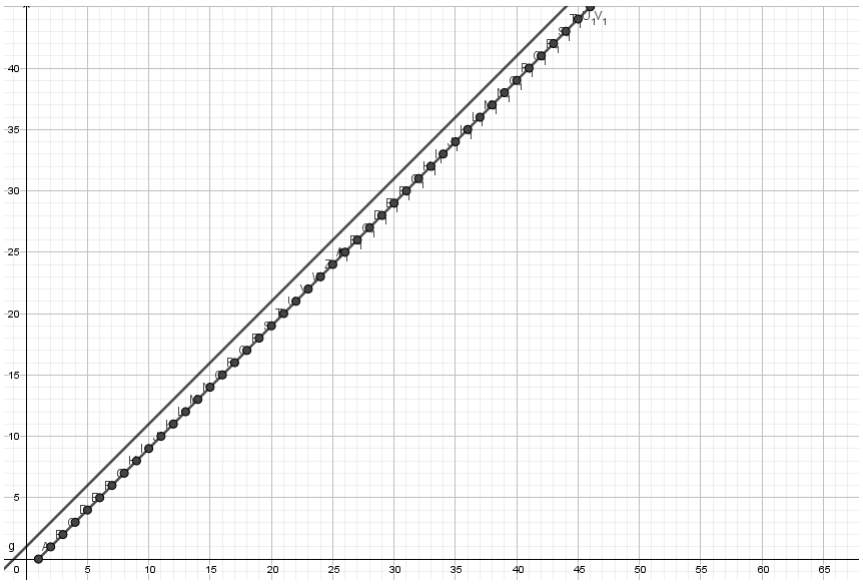

Figura 3: Grafica de Partition.

Podemos concluir que nuestra funcion es lineal ya que al compararala con una funcion que se multiplica con una constante es evidente el caracter de la funcion no solo por los resultados obtenidos en la tabla de paso, si no, tambien por la grafica que generamos.

ii)Demostrar analiticamente que el algortimo **Partition** tiene complejidad lineal.

Sea n = r-q-1;

Partition
Entrada: A[p,...,r],p,r;
Salida : q

Partition(A,p,r)

$$\left.\begin{array}{l} 1.- x = A[r-1] \\ 2.- i = \quad p-1 \end{array}\right\}\theta(1);$$

$$\left.\begin{array}{l} 3.- for \quad j = p \quad to \quad j <= r-2 \quad do \end{array}\right\}\theta(n);$$

$$\left.\begin{array}{l} 4.- \qquad if \quad A[j] <= x \\ 5.- \qquad\quad i = i+1 \\ 6.- \qquad\quad intercambio(A[j], A[i]) \end{array}\right\}\theta(1);$$

$$\left.\begin{array}{l} 7.- intercambio(A[i+1], A[r-1]) \\ 8.- return \quad i+1 \end{array}\right\}\theta(1);$$

$$\left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array}\right\}\theta(1*n) = O(n);$$

$O(1) + O(n) + O(1) = O(1) + O(n) = O(max\{1, n\}) = O(n)$
Como podemos notar para el mejor de los casos el orden de complejidad tambien es lineal.
$\therefore T(n) = \theta(n)$.

iii)Mostrar mediante graficas, que el algoritmo **QuickSort** tiene complejidad $\theta(nlogn)$.

Para empezar este ejercicio tomamos arreglos creciendo iterativamente de tamaño 1 hasta el 50, llenandolos con valores aleatorios no repetidos segun su tamaño. Para esta prueba, el algoritmo QuickSort arrojo los siguientes resultados

1	1
2	3
3	3
4	5
5	7
6	7
7	9
8	11
9	15
10	13
11	13
12	17
13	17
14	19
15	21
16	21
17	21
18	23
19	23
20	29
21	27
22	31
23	35
24	31
25	35
26	35
27	35
28	39
29	39
30	37
31	41
32	43
33	45
34	45
35	45
36	49
37	51
38	49
39	49
40	51
41	55
42	57
43	57
44	53
45	57
46	59
47	65
48	63
49	69
50	63

Figura 4: Tabla de QuickSort.

Para estos resultados, se realizo una grafica confrontando los resultados tabulados anteriormente, la cual quedo de la siguiente manera.

Figura 5: Grafica de QuickSort.

La figura numero 5 es la conclucoin a todas nuestras pruebas y a nuestros desarrollos podemos observar que el inicio de las curvas es el mismo mientras que cuando los valores de nuestro quicksort empiezan a desvariar, creemos que esta variacion es debido a los valores obtenidos por el random para llenar los arreglos, muchas veces se generan complejidades menores , como seria el caso de 1,2,3,4,5,6,..., n donde el arreglo se encuentra, si no en su totalidad ordenado, si en una parte, lo cual, creo que podria interferir con los resultados, sin embargo, se deduce que la complejidad del algoritmo QuickSort es de θ(nlogn), debido a las graficas obtenidas y su similitud.

iv)Demostrar analiticamente que el algoritmo **QuickSort** tiene complejidad $\theta(nlogn)$ cuando el pivote divide el arreglo por la mitad.

Sea n = r-p;

QUICKSORT
Entrada: A[p,...,r],p,r;
Salida : A[p,...,r] ordenado
QuickSort(A,p,r)
1.- if(p!=r)
2.- $q \leftarrow Partition(A, p, r)$;
3.- QuickSort(A,p,q);
4.- QuickSort(A,q+1,r);

Sea G(n) la complejidad del algoritmo de Partition demostrado anteriormente, es decir $\theta(n)$.
Entonces sea T(n) la complejidad del algoritmo de Quicksort la cual se define como:
T(n) = G(n) + T(q) + T(n-q) cuando $n > 1$.
Como el pivote divide el arreglo a la mitad entonces q=n/2.

Con ello
$$\text{T(n)} = \begin{cases} \theta(1) \quad si \quad n = 1; \\ 2T(n/2) + G(n) \quad si \quad n > 1 \quad con \quad n = 2^k \end{cases}$$

Es decir
$$\text{T(n)} \begin{cases} c \quad si \quad n = 1; \\ 2T(n/2) + c * n \quad si \quad n > 1 \quad con \quad n = 2^k \end{cases}$$

Como $n = 2^k$ entonces:
$T(2^k) = 2T(2^{k-1}) + c(2^k)$
$T(2^k) = 2[2T(2^{k-2}) + c(2^{k-1})] + c(2^k)$
$T(2^k) = 4T(2^{k-2}) + 2c(2^k)$
$T(2^k) = 4[2T(2^{k-3}) + c(2^{k-2})] + 2c(2^k)$
$T(2^k) = 8T(2^{k-3}) + 3c(2^k)$
$\vdots$
$T(2^k) = (2^i)T(2^{k-i}) + ic(2^k)$
Cuando i = k
$T(2^k) = (2^k)T(2^{k-k}) + kc(2^k)$
$T(2^k) = (2^k)c + kc(2^k)$
Remplazando $2^k = n \quad y \quad k = \log_2(n)$
$T(n) = cn + c(n \log_2(n))$
$T(n) = c(n + n \log_2(n))$
$\therefore \quad T(n) = \theta(n \log_2(n))$

v)Mediante graficas, proponga el orden de complejidad de QuickSort cuando todos los elementos del arreglo son distintos y estan ordenados en forma decreciente.
Para empezar este ejercicio tomamos arreglos creciendo iterativamente de tamaño 1 hasta el 50,

llenandolos con valores decrecientes con el mayor igual al tamaño del arreglo. Para esta prueba, el algoritmo QuickSort arrojo los siguientes resultados

0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	20
20	21
21	22
22	23
23	24
24	25
25	26
26	27
27	28
28	29
29	30
30	31
31	32
32	33
33	34
34	35
35	36
36	37
37	38
38	39
39	40
40	41
41	42
42	43
43	44
44	45
45	46
46	47
47	48
48	49
49	50

Figura 6: Tabla de QuickSort con orden decreciente.

Para estos resultados, se realizo una grafica confrontando los resultados tabulados anteriormente, la cual quedo de la siguiente manera.

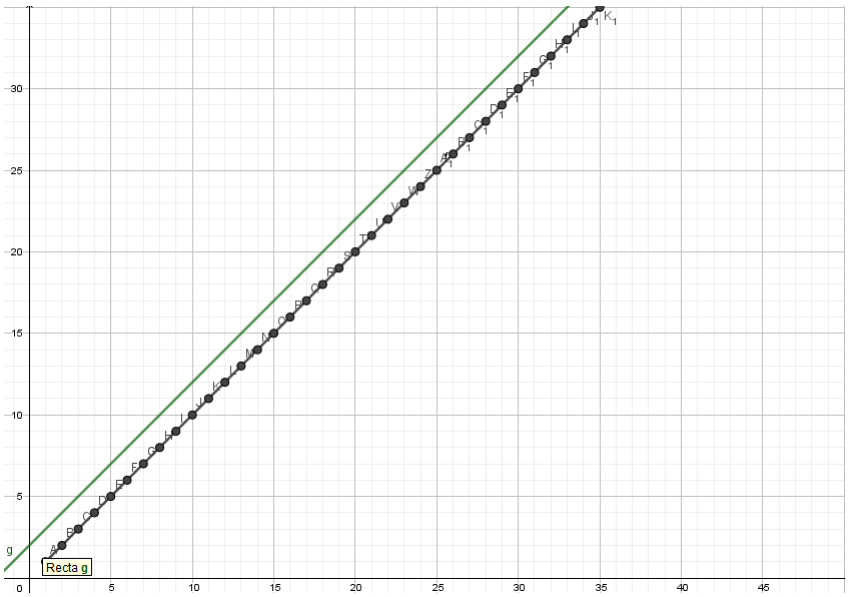

Figura 7: Grafica de QuickSort con orden decreciente.

Para estos resultados llegue a la conclusion de que el orden es lineal, ya que es un caso especial del quicksort, por la forma en la que estan acomodados los valores, esmas sencillo para el algoritmo lograr acomodarlos, lo que nos deja con una complejidad de tipo lineal, debido a los datos obtenidos con la comparacion de graficas, se llega a la conclucion de que el algoritmo QuickSort con datos decrecientes es de complejidad lineal.

4. Conclusión

En esta pracitca observe que dependiendo de el tipo de datos que se introduzcan a un algoritmo de ordenacion como QuickSort, dependera el tiempo computacional que tarde en ordenarlo, debido a su funcionamiento.
Comenzamos a ver algoritmos mas y mas complejops, que, ademas de su complejidad debido a la recusividad, son ahora dependientes de los valores con los que se va a probar.
Tuve algunos problemas debido al las graficas de quicksort, intentaba tabular 200 resultados, pero los valores cambiaban demasiado, no lograba entender que sucedia hasta que decidi imprimir cada arreglo antes y despues de ordenarlo, imprimiendo igual las veces que se ejecuto el algoritmo QuickSort, fue entonces cuando note que lops algoritmos que tenian valores ya ordenados, aunque fuera en una parte solamente, tardaban menos pasos en ser ordenados en su totalidad, por lo tanto, reduje el tamaño de las pruebas para lograr reducir este fallo.

5. Anexos:

5.1. Resolver los siguientes problemas:

1. Que valor de q retorna **Partition** cuando los elementos del arreglo $A[p, ..., r]$ tienen el mismo valor?

Con este experimento, donde pusimos los arreglos llenos de 1, pudimos ver que el valor que Partition nos regresa es igual al tamaño del arreglo mismo. Esto lo podemos garantizar por lo que vemos en la figura 8, los arreglos en este experimento fueron tomados del 1 al 50.

2.Cual es el tiempo de ejecucion de **QuickSort** cuando todos los elementos del arreglo tienen el mismo valor?

Del mismo modo, la tabulacion obtenida suguiere que la complejidad obtenida por el algoritmo QuickSort con datos iguales, es de complejidad lineal.

```
C:\Python27\python.exe "D:/ESCOM/5to/analisis de algoritmos/practica5/ej1.py"
Despues de ordenar
[1]
Pasos de QuickSort: 1
Pasos de Partition: 0

Despues de ordenar
[1, 1]
Pasos de QuickSort: 2
Pasos de Partition: 1

Despues de ordenar
[1, 1, 1]
Pasos de QuickSort: 3
Pasos de Partition: 2

Despues de ordenar
[1, 1, 1, 1]
Pasos de QuickSort: 4
Pasos de Partition: 3

Despues de ordenar
[1, 1, 1, 1, 1]
Pasos de QuickSort: 5
Pasos de Partition: 4

Despues de ordenar
[1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 6
Pasos de Partition: 5

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 10
Pasos de Partition: 9

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 11
Pasos de Partition: 10

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 12
Pasos de Partition: 11

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 13
Pasos de Partition: 12

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 14
Pasos de Partition: 13

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 15
Pasos de Partition: 14

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 20
Pasos de Partition: 19

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 21
Pasos de Partition: 20

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 22
Pasos de Partition: 21

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 23
Pasos de Partition: 22

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 24
Pasos de Partition: 23

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 25
Pasos de Partition: 24

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 31
Pasos de Partition: 30

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 32
Pasos de Partition: 31

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 33
Pasos de Partition: 32

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 34
Pasos de Partition: 33

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 35
Pasos de Partition: 34

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 36
Pasos de Partition: 35

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 45
Pasos de Partition: 44

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 46
Pasos de Partition: 45

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 47
Pasos de Partition: 46

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 48
Pasos de Partition: 47

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 49
Pasos de Partition: 48

Despues de ordenar
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Pasos de QuickSort: 50
Pasos de Partition: 49
```

Figura 8: Tabla de QuickSort con orden numeros iguales.