Universidad ICESI
Facultad de Ingeniería
Ingeniería de Sistemas

Nicolas Biojo Bermeo (A00137580), Jose A. Galvis Nieto (A00302328), Juan M. Lopez (A00130500), Christian F. López (A00134296), Sara Ortiz Drada (A00302324)

Análisis de Complejidad

QuickSort: `n = r-p+1`

Línea	Instrucción	# veces que se ejecuta		
	QuickSort(A,p,r)	Mejor	Peor	Promedio
1	**if** p < r	n-1	n-1	n-1
2	q = Partition(A,p,r)	2n+1	4n+3	2n+1
3	QuickSort(A,p,q-1)	T(n/2)	T(n-1)	T(i-1)
4	QuickSort(A,q+1,r)	T(n/2)	T(0)	T(n-i)
Total		T(n)=2T(n/2)+3n T(n)=O(n*lgn)	T(n)=T(n-1)+T(0)+θ(5n+2) T(n)=θ(n^2)	T(n)=T(n-i)+T(i-1)+3n T(n)=O(n*lgn)

Quicksort **peor caso**: T(n)=θ(n^2)

Utilizando el método de sustitución:
T(n)=T(n-1) +T(0)+ □ (5n+2), T(0)=0 y □ (5n+2) O(n)
T(n)=T(n-1) +n
T(n-1) =T(n-2)+2n-1
T(n-2) =T(n-3)+3n-3
T(n-3) =T(n-4)+4n-6
.
.
.
T(n-k-1) =T(n-k) + kn-(k*(k-1))/2, k=n-1
n-k=1, T(n-k) =T (1)
T(n-k-1) =T (1) +(n^2-n) -(n^2-3n+2)/2
T(n-k-1) =1+(n^2+n-2)/2 O(n^2)

Así, se concluye que en el mejor caso Quicksort se ejecuta en un tiempo T(n)=O(n^2)
Quicksort **mejor caso**: T(n)=2T(n/2)+3n

Aplicando el teorema del maestro:

f(n)=n^lg2=n^1, entonces T(n)= (n^1*logn)
Así, se concluye que en el mejor caso Quicksort se ejecuta en un tiempo T(n)=O(n*logn)

Quicksort **caso promedio:** T(n)=T(n-i)+T(i-1)+(3n)

Por medio del método de sustitución es posible resolver este caso.
.
.
.
T(n)=O(n*logn)

Línea	Instrucción	# veces que se ejecuta	
	Partition(A,p,r)	Mejor	Peor
1	x = A[r]	1	1
2	i = p - 1	1	1
3	**for** j = p to r - 1	n-1	n-1
4	**if** A[j] ≤ x	n-2	n-2
5	i = i + 1	0	n-2
6	A[i] ↔ A[j]	0	n-2
7	A[i + 1] ↔ A[r]	1	1
8	return i + 1	1	1
Total		T(n)=2n+1	T(n)=4n-3

QuickSort No Randomized

Randomized-QS: n = r-p+1
0 ≤ q ≤ n-1
1 ≤ i ≤ n-1

Línea	Instrucción	# veces que se ejecuta		
	Randomized-QS(A,p,r)	Mejor	Peor	Promedio
1	**if** p < r	n-1	n-1	n-1
2	q = Rand-Parti(A,p,r)	2n+3	4n-1	2n+3
3	Randomized-QS(A,p,q-1)	T(i-1)	T(q)	T(n/2)
4	Randomized-QS(A,q+1,r)	T(n-i)	T(n-1-q)	T(n/2)
Total		T(n)=T(n-i)+T	T(n)=T(q)+T(	T(n)=2T(n/2)+

	(i-1)+3n+2 T(n)=O(n*lgn)	n-1-q)+θ(5n-2) T(n)=θ(n^2)	3n+2 T(n)=O(n*lgn)

Randomized Quicksort **peor caso:** T(n)=T(q)+T(n-1-q)+□(5n-2), $0 \leq q \leq n-1$
(El proceso partition, producirá dos subproblemas con tamaño total de n-1) y □(5n-2) O(n).

Asumimos que T(n)<=cn^2 para alguna constante c. Así,

$$T(n) \leq ((cq^2)+c(n-1-q)^2)+n = c*((q^2)+(n-1-q)^2)+n$$

Como la expresión ((q^2)+(n-1-q)^2) alcanza un máximo en q=n-1, también se cumple que:

$$((q^2)+(n-1-q)^2)<=(n-1)^2 = n^2-2n+1$$

De esta manera, al igual que en el caso anterior, podemos continuar de la siguiente manera
T(n)<=cn^2-c(2n-1)+n.

Como se puede tener un término c tan grande que n sea despreciable, tendríamos:
T(n)<=cn^2.

Por tanto, T(n)=O(n^2)

Para este algoritmo, el mejor y el caso promedio son muy similares sino es que iguales al de Quicksort.

Línea	Instrucción	# veces que se ejecuta	
	Rand-Parti(A,p,r)	Mejor	Peor
1	i = Random(p,r)	1	1
2	A[r] ↔ A[i]	1	1
3	**return** Partition(A,p,r)	2n+1	4n-3
Total		2n+3	4n-1

Variante del QuickSort Randomized

Con el resultado de análisis de complejidad previo, se formula la siguiente función como hipótesis para cada caso:

	Factores de Entrada		
# de Prueba	**Variante del QS X_1**	**Estado del Arreglo X_2**	**Tiempo (en ms) Y**
1	No Randomized	Ordenado No Descendente	$O(n \cdot \log n)$
2	Randomized	Ordenado No Descendente	$O(n^2)$
3	No Randomized	Ordenado No Ascendente	$O(n \cdot \log n)$
4	Randomized	Ordenado No Ascendente	$O(n^2)$
5	No Randomized	No ordenado (en orden aleatorio)	$O(n \cdot \log n)$.
6	Randomized	No ordenado (en orden aleatorio)	$O(n \cdot \log n)$.

Resultados

	Resultados QS (milis)	Resultados QSR (milis)
10^1		
aleatorio	0,008	0,002
ascendente	0,001	0,006
descendente	0,001	0,001
10^2		
aleatorio	0,001	0,02
ascendente	0,031	0,011
descendente	0,049	0,004
10^3		
aleatorio	0,077	0,313
ascendente	3,788	0,255
descendente	3,439	0,26
10^4		
aleatorio	1,841	10,313
ascendente	354,34	9,497
descendente	353,885	9,536
10^5		
aleatorio		112,847
ascendente		103,446
descendente		103,836

Diseño de pruebas

➔ Entradas Pequeñas: arreglos con tamaños de 10^1 y 10^2

Objetivo: Verificar que el método ordena un arreglo (sin importar su orden inicial) de forma ascendente.				
Clase: Sorts		**Método:** `RandomizedQuickSort(int[] input, int left, int right)`		
Caso #	**Descripción de la prueba**	**Escenario**	**Valores de entrada**	**Resultado**
1	Dado un arreglo no ordenado con tamaño n = 10^1, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	1	`input = {10, 1324, 356, 0, 4875430, 1000000000, 1198, 7, 3210, 345435}`	Retorna el arreglo ordenado de forma ascendente: `{0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`
2	Dado un arreglo ordenado no descendente con tamaño n = 10^1, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	2	`input = {0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`	Retorna el arreglo ordenado de forma ascendente: `{0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`
3	Dado un arreglo ordenado no ascendente con tamaño n = 10^1, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	3	`input = {1000000000, 4875430, 345435, 3210, 1324, 1198, 356, 10, 7, 0}`	Retorna el arreglo ordenado de forma ascendente: `{0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`
4	Dado un arreglo no ordenado con tamaño n = 990, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	4	input: arreglo de enteros de tamaño n = 990.	Retorna el arreglo ordenado de forma ascendente.
5	Dado un arreglo ordenado no descendente con tamaño n = 990, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	5	input: arreglo de enteros de tamaño n = 990.	Retorna el arreglo ordenado de forma ascendente.
6	Dado un arreglo ordenado no ascendente con tamaño n = 990, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	6	input: arreglo de enteros de tamaño n = 990.	Retorna el arreglo ordenado de forma ascendente.

Objetivo: Verificar que el método ordena un arreglo (sin importar su orden inicial) de forma ascendente.				
Clase: Sorts		**Método:** `QuickSort(int[] arr, int low, int high)`		
Caso #	**Descripción de la prueba**	**Escenario**	**Valores de entrada**	**Resultado**
7	Dado un arreglo no ordenado con tamaño n = 10^1, verifica que el algoritmo `QuickSort` ordene el arreglo de forma ascendente correctamente.	1	`arr= {10, 1324, 356, 0, 4875430, 1000000000, 1198, 7, 3210, 345435}`	Retorna el arreglo ordenado de forma ascendente: `{0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`
8	Dado un arreglo ordenado no descendente con tamaño n = 10^1, verifica que el algoritmo `QuickSort` ordene el arreglo de forma ascendente correctamente.	2	`arr = {0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`	Retorna el arreglo ordenado de forma ascendente: `{0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`
9	Dado un arreglo ordenado no ascendente con tamaño n = 10^1, verifica que el algoritmo `QuickSort` ordene el arreglo de forma ascendente correctamente.	3	`arr = {1000000000, 4875430, 345435, 3210, 1324, 1198, 356, 10, 7, 0}`	Retorna el arreglo ordenado de forma ascendente: `{0, 7, 10, 356, 1198, 1324, 3210, 345435, 4875430, 1000000000}`
10	Dado un arreglo no ordenado con tamaño n = 990, verifica que el algoritmo `QuickSort` ordene el arreglo de forma ascendente correctamente.	4	arr: arreglo de enteros de tamaño n = 990.	Retorna el arreglo ordenado de forma ascendente.
11	Dado un arreglo ordenado no descendente con tamaño n = 990, verifica que el algoritmo `QuickSort` ordene el arreglo de forma ascendente correctamente.	5	arr: arreglo de enteros de tamaño n = 990.	Retorna el arreglo ordenado de forma ascendente.
12	Dado un arreglo ordenado no ascendente con tamaño n = 990, verifica que el algoritmo `QuickSort` ordene el arreglo de forma ascendente correctamente.	6	arr: arreglo de enteros de tamaño n = 990.	Retorna el arreglo ordenado de forma ascendente.

➔ Entradas Grandes: arreglos con tamaños de 10^3, 10^4 y 10^5.

Objetivo: Verificar que el método ordena un arreglo (sin importar su orden inicial) de forma ascendente.				
Clase: Sorts		**Método:** RandomizedQuickSort(int[] input, int left, int right)		
Caso #	**Descripción de la prueba**	**Escenario**	**Valores de entrada**	**Resultado**

13	Dado un arreglo no ordenado con tamaño n = 5298, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`7`	input: arreglo de enteros de tamaño n = 5298.	Retorna el arreglo ordenado de forma ascendente.
14	Dado un arreglo ordenado no descendente con tamaño n = 5298, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`8`	input: arreglo de enteros de tamaño n = 5298.	Retorna el arreglo ordenado de forma ascendente.
15	Dado un arreglo ordenado no ascendente con tamaño n = 5298, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`9`	input: arreglo de enteros de tamaño n = 5298.	Retorna el arreglo ordenado de forma ascendente.
16	Dado un arreglo no ordenado con tamaño n = 17634, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`10`	input: arreglo de enteros de tamaño n = 17634.	Retorna el arreglo ordenado de forma ascendente.
17	Dado un arreglo ordenado no descendente con tamaño n = 17634, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`11`	input: arreglo de enteros de tamaño n = 17634.	Retorna el arreglo ordenado de forma ascendente.
18	Dado un arreglo ordenado no ascendente con tamaño n = 17634, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`12`	input: arreglo de enteros de tamaño n = 17634.	Retorna el arreglo ordenado de forma ascendente.
19	Dado un arreglo no ordenado con tamaño n = 999999, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`13`	input: arreglo de enteros de tamaño n = 999999.	Retorna el arreglo ordenado de forma ascendente.
20	Dado un arreglo ordenado no descendente con tamaño n = 999999, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`14`	input: arreglo de enteros de tamaño n = 999999.	Retorna el arreglo ordenado de forma ascendente.
21	Dado un arreglo ordenado no ascendente con tamaño n = 999999, verifica que el algoritmo `RandomizedQuickSort` ordene el arreglo de forma ascendente correctamente.	`15`	input: arreglo de enteros de tamaño n = 999999.	Retorna el arreglo ordenado de forma ascendente.

Objetivo: Verificar que el método ordena un arreglo (sin importar su orden inicial) de forma ascendente.				
Clase: Sorts		**Método:** QuickSort(int[] arr, int low, int high)		
Caso #	**Descripción de la prueba**	**Escenario**	**Valores de entrada**	**Resultado**
22	Dado un arreglo no ordenado con tamaño n = 5298, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	7	arr: arreglo de enteros de tamaño n = 5298.	Retorna el arreglo ordenado de forma ascendente.
23	Dado un arreglo ordenado no descendente con tamaño n = 5298, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	8	arr: arreglo de enteros de tamaño n = 5298.	Retorna el arreglo ordenado de forma ascendente.
24	Dado un arreglo ordenado no ascendente con tamaño n = 5298, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	9	arr: arreglo de enteros de tamaño n = 5298.	Retorna el arreglo ordenado de forma ascendente.
25	Dado un arreglo no ordenado con tamaño n = 17634, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	10	arr: arreglo de enteros de tamaño n = 17634.	Retorna el arreglo ordenado de forma ascendente.
26	Dado un arreglo ordenado no descendente con tamaño n = 17634, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	11	arr: arreglo de enteros de tamaño n = 17634.	Retorna el arreglo ordenado de forma ascendente.
27	Dado un arreglo ordenado no ascendente con tamaño n = 17634, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	12	arr: arreglo de enteros de tamaño n = 17634.	Retorna el arreglo ordenado de forma ascendente.
28	Dado un arreglo no ordenado con tamaño n = 999999, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	13	arr: arreglo de enteros de tamaño n = 999999.	Retorna el arreglo ordenado de forma ascendente.
29	Dado un arreglo ordenado no descendente con tamaño n = 999999, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	14	arr: arreglo de enteros de tamaño n = 999999.	Retorna el arreglo ordenado de forma ascendente.
30	Dado un arreglo ordenado no ascendente con tamaño n = 999999, verifica que el algoritmo	15	arr: arreglo de enteros de tamaño n = 999999.	Retorna el arreglo ordenado de forma ascendente.

	QuickSort ordene el arreglo de forma ascendente correctamente.			

➔ Entrada Aleatoria: arreglo con tamaño aleatorio entre 10^1 y 10^5

Objetivo: Verificar que el método ordena un arreglo (sin importar su orden inicial) de forma ascendente.				
Clase: Sorts		**Método:** RandomizedQuickSort(int[] input, int left, int right)		
Caso #	**Descripción de la prueba**	**Escenario**	**Valores de entrada**	**Resultado**
31	Dado un arreglo no ordenado con tamaño n = 384759, verifica que el algoritmo RandomizedQuickSort ordene el arreglo de forma ascendente correctamente.	16	input: arreglo de enteros de tamaño n = 384759.	Retorna el arreglo ordenado de forma ascendente.
32	Dado un arreglo ordenado no descendente con tamaño n = 384759, verifica que el algoritmo RandomizedQuickSort ordene el arreglo de forma ascendente correctamente.	17	input: arreglo de enteros de tamaño n = 384759.	Retorna el arreglo ordenado de forma ascendente.
33	Dado un arreglo ordenado no ascendente con tamaño n = 384759, verifica que el algoritmo RandomizedQuickSort ordene el arreglo de forma ascendente correctamente.	18	input: arreglo de enteros de tamaño n = 384759.	Retorna el arreglo ordenado de forma ascendente.

Objetivo: Verificar que el método ordena un arreglo (sin importar su orden inicial) de forma ascendente.				
Clase: Sorts		**Método:** QuickSort(int[] arr, int low, int high)		
Caso #	**Descripción de la prueba**	**Escenario**	**Valores de entrada**	**Resultado**
34	Dado un arreglo no ordenado con tamaño n = 384759, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	16	arr: arreglo de enteros de tamaño n = 384759.	Retorna el arreglo ordenado de forma ascendente.
35	Dado un arreglo ordenado no descendente con tamaño n = 384759, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	17	arr: arreglo de enteros de tamaño n = 384759.	Retorna el arreglo ordenado de forma ascendente.
36	Dado un arreglo ordenado no ascendente con tamaño n = 384759, verifica que el algoritmo QuickSort ordene el arreglo de forma ascendente correctamente.	18	arr: arreglo de enteros de tamaño n = 384759.	Retorna el arreglo ordenado de forma ascendente.

Escenarios

Escenario	**Representación**
1: No Ordenado	**10 1,324 356 0 4,875,430 1,000,000,000 1,198 7 3,210 345,435**
2: Ordenado No Descendente	**0 7 10 356 1,198 1,324 3,210 345,435 4,875,430 1,000,000,000**
3: Ordenado No Ascendente	**1,000,000,000 4,875,430 345,435 3,210 1,324 1,198 356 10 7 0**
4: No Ordenado	a_1 a_2 a_3 ... a_n Con n = 990, $\{ a_1, a_2, a_3, \dots, a_n \mid a_i$ *no tenga orden* $\}$ y $0 \leq a_i \leq 10^9$
5: Ordenado No Descendente	a_1 a_2 a_3 ... a_n Con n = 990, $\{ a_1, a_2, a_3, \dots, a_n \mid a_1 \leq a_2 \leq a_3 \leq \dots \leq a_n \}$ y $0 \leq a_i \leq 10^9$
6: Ordenado No Ascendente	a_1 a_2 a_3 ... a_n Con n = 990, $\{ a_1, a_2, a_3, \dots, a_n \mid a_1 \geq a_2 \geq a_3 \geq \dots \geq a_n \}$ y $0 \leq a_i \leq 10^9$
7: No Ordenado	a_1 a_2 a_3 ... a_n Con n = 5298, $\{ a_1, a_2, a_3, \dots, a_n \mid a_i$ *no tenga orden* $\}$ y $0 \leq a_i \leq 10^9$
8: Ordenado No Descendente	a_1 a_2 a_3 ... a_n Con n = 5298, $\{ a_1, a_2, a_3, \dots, a_n \mid a_1 \leq a_2 \leq a_3 \leq \dots \leq a_n \}$ y $0 \leq a_i \leq 10^9$
9: Ordenado No Ascendente	a_1 a_2 a_3 ... a_n Con n = 5298, $\{ a_1, a_2, a_3, \dots, a_n \mid a_1 \geq a_2 \geq a_3 \geq \dots \geq a_n \}$ y $0 \leq a_i \leq 10^9$

10: No Ordenado	a_1 a_2 a_3 ... a_n Con n = 17634, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_i$ *no tenga orden* $\}$ y $0 \leq a_i \leq 10^9$
11: Ordenado No Descendente	a_1 a_2 a_3 ... a_n Con n = 17634, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_n \}$ y $0 \leq a_i \leq 10^9$
12: Ordenado No Ascendente	a_1 a_2 a_3 ... a_n Con n = 17634, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_n \}$ y $0 \leq a_i \leq 10^9$
13: No Ordenado	a_1 a_2 a_3 ... a_n Con n = 999999, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_i$ *no tenga orden* $\}$ y $0 \leq a_i \leq 10^9$
14: Ordenado No Descendente	a_1 a_2 a_3 ... a_n Con n = 999999, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_n \}$ y $0 \leq a_i \leq 10^9$
15: Ordenado No Ascendente	a_1 a_2 a_3 ... a_n Con n = 999999, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_n \}$ y $0 \leq a_i \leq 10^9$
16: No Ordenado	a_1 a_2 a_3 ... a_n Con n = 384759, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_i$ *no tenga orden* $\}$ y $0 \leq a_i \leq 10^9$
17: Ordenado No Descendente	a_1 a_2 a_3 ... a_n Con n = 384759, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_n \}$ y $0 \leq a_i \leq 10^9$
18: Ordenado No Ascendente	a_1 a_2 a_3 ... a_n Con n = 384759, $\{ a_1, a_2, a_3, \ldots, a_n \mid a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_n \}$ y $0 \leq a_i \leq 10^9$

Planeación y realización

I. Unidad experimental

En el presente experimento, se trabaja con dos unidades experimentales, el algoritmo `QuickSort` y el algoritmo `RandomizedQuickSort`, donde a cada uno los anteriores se les modificarán factores de entrada para validar su resultado. Entre dichos factores se encuentran la cantidad de elementos inicial y el orden inicial del arreglo; cabe mencionar que se tomará una muestra de 1000 pruebas con cada uno de los tamaños establecidos (10 ^ 1, 10 ^ 2, 10 ^ 3, 10 ^ 4, 10 ^ 5) para su posterior análisis.

II. Variables de respuesta

La variable de respuesta de la prueba de los dos algoritmos será el tiempo, es decir, que esta será la variable que, en teoría, cambiará su valor al alterar los diversos factores de entrada de los algoritmos.

III. Factores controlables

Los factores que se pueden controlar dentro del presente experimento son:

- Número de elementos de entrada.
- Orden inicial de los elementos de entrada.

IV. Factores no controlables

Los factores que no pueden ser controlados dentro de este experimento y, por tal, alteran el resultado de las pruebas, son las características propias del equipo o computador donde se ejecutan las pruebas, entre estas se encuentran:

- Memoria ram.
- Fragmentación de la memoria.
- Núcleos de equipo.
- Tareas realizadas en paralelo con la prueba.

V. Factores estudiados

Los factores estudiados que influyen en la variable de respuesta son:

- Número de elementos a ordenar.
- Orden inicial de los elementos.

VI. Niveles

La siguiente tabla presenta los diversos niveles que serán utilizados en el presente experimento.

Número de elementos de entrada	Nivel de elementos
10	1

10 ^ 2	2
10 ^ 3	3
10 ^ 4	4
10 ^ 5	5

Niveles utilizados para el número de elementos de entrada.

Orden inicial de elementos	Nivel de orden
Randomized	1
No ascendente	2
No descendente	3

Niveles utilizados para el orden inicial de los elementos.

VII. Tratamientos

A continuación se presenta la combinación de factores estudiados.

Nivel de elementos	Nivel de orden	Tratamiento
1	1	1
	2	2
	3	3
2	1	4
	2	5
	3	6
3	1	7
	2	8

	3	9
4	1	10
	2	11
	3	12
5	1	13
	2	14
	3	15

Análisis

Etapas en el diseño de experimentos

Hasta este momento, se han llevado a cabo tres de las cuatro etapas del diseño de experimentos, con el objetivo de demostrar que el método de ordenamiento `RandomizedQuickSort` es más rápido que el `QuickSort` normal. A continuación se describe el desarrollo realizado de cada una.

Para la etapa de planeación, se delimitó el objeto de estudio como el comportamiento de los algoritmos, `QuickSort` y `RandomizedQuickSort`, posteriormente, se estableció la variable que sería medida mediante la experimentación, la cual es el tiempo de ejecución, así mismo, se seleccionaron los factores que, en teoría, tienen influencia sobre la variable de respuesta, en este caso se optó por la cantidad de elementos a ordenar y el orden inicial de los mismos. Ya con lo anterior, fue necesario establecer el número de repeticiones que se llevarían a cabo con cada tratamiento, donde se eligieron 1000 repeticiones.
Para el desarrollo de la experimentación se seleccionó a José Galvis como el encargado de esta, sin embargo en las pruebas llevadas a cabo con un 10 ^ 5 elementos, fue necesario asignar otro individuo, Nicolás Biojo, esto se hizo, pues el computador de José Galvis lanzaba una StackOverFlowException, haciendo referencia a que la pila de ejecución se desbordó.

En la etapa de análisis se utilizó la técnica de análisis de experimentos ANOVA (Analysis Of Variance/análisis de la varianza), esto con el objetivo de validar el hecho de que la versión aleatoria del QuickSort es más rápida que su versión no aleatoria.