Adrian Cruz - 10154816
José Comprés - 10123259

Explicación del programa para el ejercicio 1

Este programa utiliza un algoritmo para rotar T veces las posiciones de los elementos de un arreglo de tamaño n. El valor n siempre debe ser par y el valor T debe estar comprendido entre los valores 1 y n-1. Posteriormente, el arreglo se divide en dos mitades, para los valores impares de T, se intercambia el ultimo valor de la primera mitad con la primera posición y los demás valores van rotando, para los valores pares se intercambia el último valor del arreglo con el primer valor de la segunda mitad y los demás valores van rotando, esto se repite hasta llegar al valor de T.

Instrucciones para compilar el programa y ejecutar

Para compilar y ejecutar el programa únicamente hay que presionar el botón Run en la clase ListasVSArrays.

Ejemplo de entrada y salida

```
Ingrese la cantidad de valores (Debe ser par): 6
Ingrese el valor T:
3
Ingrese el valor 1:
1
Ingrese el valor 2:
2
Ingrese el valor 3:
3
Ingrese el valor 4:
4
Ingrese el valor 5:
5
Ingrese el valor 6:
6
1 2 3 4 5 6
2 3 1 6 4 5
Process finished with exit code 0
```

Análisis del rendimiento en función del número de procesos utilizados

Mejor caso: O(n/2)

Peor caso: O(n)

EXPLICACIÓN DE LA SOLUCIÓN DEL PROBLEMA 2 (Método ordenarQS en Solve.java)

Para la solución de este problema se implementó el algoritmo de ordenamiento al estilo de Quicksort. Es un algoritmo recursivo que va ordenando sublistas de la lista total. Su caso base es que la longitud de la lista recibida sea menor que 2, en cuyo caso no hay nada que ordenar, pues la lista sería de un elemento o sería vacía, en este caso, se retorna la lista recibida, pues prácticamente ya estaría ordenada.

En este método se recibe una lista de estudiantes sin ordenar en cuanto al índice académico y el parámetro booleano ascendente, el cuál, como su nombre lo indica, determinará cómo se ordenará la lista (ascendente o descendentemente). El algoritmo empieza creando tres listas: ordenada, mayores y menores. Luego, elige un "pivote" que será un estudiante cuyo índice determinará cómo se dividirá la lista total entre las listas mayores y menores. En este caso, siempre se elige el primer estudiante de la lista como el pivote, y se toma su índice para compararlo con los del resto de los estudiantes. Para comparar, se empieza a iterar desde el segundo estudiante de la lista, ya que el primero ya se está evaluando como pivote. Los estudiantes con índice menor o igual al del pivote se irán a la lista de menores, y los que tengan un índice mayor se irán a la lista de mayores. Luego, se pasa a ordenar estas sublistas menores y mayores con la misma función, y como se puede notar, mientras más avance el algoritmo, más se irán reduciendo el tamaño de las listas, hasta que dar con un elemento o cero (en el caso de que una lista de mayores o menores quede vacía en la comparación con su respectivo pivote). Al final, solo quedarán 1 elemento en cualquier lado del pivote, por ejemplo, si quedan tres elementos y resulta que uno es menor y otro es mayor al pivote, entonces se queda el menor a la izquierda y el mayor a la derecha, en caso de ser true el parámetro ascendente, en caso de que el parámetro ascendente sea false, entonces se invertirán los lados. También puede darse el caso de que queden dos elementos, y solo se pasará a ordenar estos dos elementos. Al final quedarán ordenadas las listas en estas situaciones y se irán retornando a las respectivas instancias en que fueron llevadas a ordenarQS.

INSTRUCCIONES DE EJECUCIÓN DE LA SOLUCIÓN 2:
Ejecute la clase Run del paquete com.algyests.practica2.problema2

EJEMPLO DE ENTRADA Y SALIDA

```
Digite la cantidad de estudiantes que desea generar: 10

Estudiantes generados:
4587  -  Estudiante #1  -  1.2985808198373898
5140  -  Estudiante #2  -  2.0495891920053735
4574  -  Estudiante #3  -  3.046004893840092
3600  -  Estudiante #4  -  3.7295994273298767
5594  -  Estudiante #5  -  0.072584878408124
2793  -  Estudiante #6  -  3.33316082667117
5819  -  Estudiante #7  -  1.0406589805376143
4773  -  Estudiante #8  -  2.745038372968155
3278  -  Estudiante #9  -  3.56073835873124
4210  -  Estudiante #10  -  1.5440672454428426

Desea que el orden sea (1)ascendente o (2)descendente? 1

Estudiantes ordenados
5594  -  Estudiante #5  -  0.072584878408124
5819  -  Estudiante #7  -  1.0406589805376143
4587  -  Estudiante #1  -  1.2985808198373898
4210  -  Estudiante #10  -  1.5440672454428426
5140  -  Estudiante #2  -  2.0495891920053735
4773  -  Estudiante #8  -  2.745038372968155
4574  -  Estudiante #3  -  3.046004893840092
2793  -  Estudiante #6  -  3.33316082667117
3278  -  Estudiante #9  -  3.56073835873124
3600  -  Estudiante #4  -  3.7295994273298767
```

```
Digite la cantidad de estudiantes que desea generar: 10

Estudiantes generados:
2496  -  Estudiante #1  -  2.9476615125231937
6042  -  Estudiante #2  -  0.4572808541351847
2205  -  Estudiante #3  -  1.4049122466693538
2914  -  Estudiante #4  -  0.11960181850720009
6080  -  Estudiante #5  -  0.6664933009250635
6932  -  Estudiante #6  -  2.0012215329579
2448  -  Estudiante #7  -  2.0487958155868844
2209  -  Estudiante #8  -  0.15493522816622285
4207  -  Estudiante #9  -  3.6831484718387135
5380  -  Estudiante #10  -  0.15135073738535 62

Desea que el orden sea (1)ascendente o (2)descendente? 2

Estudiantes ordenados
4207  -  Estudiante #9  -  3.6831484718387135
2496  -  Estudiante #1  -  2.9476615125231937
2448  -  Estudiante #7  -  2.0487958155868844
6932  -  Estudiante #6  -  2.0012215329579
2205  -  Estudiante #3  -  1.4049122466693538
6080  -  Estudiante #5  -  0.6664933009250635
6042  -  Estudiante #2  -  0.4572808541351847
2209  -  Estudiante #8  -  0.15493522816622285
5380  -  Estudiante #10  -  0.15135073738535 62
2914  -  Estudiante #4  -  0.11960181850720009
```

Análisis del rendimiento en función del número de procesos utilizados

Promedio: O(n log n)
Peor caso O(n^2)