Universidad del Valle de Guatemala
Programación de Microprocesadores
Sección No. 10
Docente: Juan Celada

Proyecto #2 de desarrollo:
Algoritmo de clasificación Par-Impar

Alejandro José Gomez Hernández 20347
Ana Paola de León Molina 20361
Cristian Eduardo Aguirre Duarte 20231
Gabriela Paola Contreras Guerra 20213

I. Índice

II. Introducción

Como parte del curso de programación de microprocesadores, se plantea el presente trabajo, el cual tiene como finalidad identificar y explotar las características de la paralelización potencial en algoritmos usando pthreads y el lenguaje C++. El objetivo principal fue aplicar los conceptos teóricos vistos a lo largo del curso. Además, de implementar operaciones del tipo mutex, variables de condición y barreras.

El temario asignado corresponde a la clasificación par-impar, para la realización de operaciones paralelas sin dependencia hacia los resultados de otros hilos. Este algoritmo se encarga de establecer las fases del proceso, en la cual se cambian los valores locales, que son clasificados posteriormente por el vecino correspondiente. Finalmente, se clasifican de forma local secuencialmente las llaves que se intercambiaron, para finalizar con un número limitado de fases, en la cual las llaves se encontrarán clasificadas.

Se logró concluir que el paralelismo es una forma eficaz de realizar un programa puesto a que este logra distribuir las tareas en varios bloques, permitiendo así, la realización de tareas en menos tiempo. Se recomienda correr el programa en una Raspberry Pi para evitar cualquier tipo de inconveniente al compilar.

III. Antecedentes del proyecto

Los números siempre han jugado un papel fundamental en la vida del ser humano, siendo utilizados desde el lejano año de 7.000 a.C., durante la época egipcia. Sin embargo, no fue hasta el año 1926 que se estableció un principio matemático. Este, fue llamado principio de paridad y se dio a conocer en el libro de Modern Puzzles de Henrry Ernest Dudeney. Henrry nunca realizó estudios universitarios de matemáticas, aunque provenía de una familia de maestros matemáticos. Este principio Henrry lo definió como "Si *un objeto o elemento cualquiera que pueda tener asociados dos estados diferentes. Si cambia su estado un número par de veces, volverá a su estado original. Por el contrario, si cambia un número impar de veces, entonces se modificará su estado"*. En palabras más simples, este principio permite establecer si un número cuenta con atributos de ser par o impar.

Un número par se puede definir como aquellos números que pueden dividirse dentro de dos, dentro de estos se encuentran los números 2,4,6,8,10,12, etc. Una forma de identificar estos números es que la cifra de las unidades de estos resultados siempre es cero o un número par. Por otro lado, los números impares son aquellos que no pueden agruparse de dos en dos como lo son los números: 1, 3, 5, 7, 9, 11… etc.

Como se mencionó anteriormente, los números son parte de casi todas las actividades que se realizan. Entre todas las que se encuentran, existe una, muy importante para la carrera de Ciencias de la Computación, que es toda la parte de la programación. Estos son fundamentales para realizar la comunicación interna de los dispositivos. Adicionalmente, se puede mencionar que permiten la realización de cálculos. Esto es de suma importancia debido a que, al ser una computadora, las operaciones matemáticas a realizar con números no tienen mayor límite más que la capacidad y potencia de la computadora a utilizar.

El paralelismo es una técnica de computación que se basa en dividir las tareas en varias tareas pequeñas y resolverlas al mismo tiempo. Esto permite la ejecución de instrucciones en un tiempo menor. Este se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente. Así mismo, es importante destacar que esta técnica ha sido empleada durante muchos años, sobre todo para la computación de alto rendimiento, teniendo en cuenta las generaciones de procesadores. Este permite resolver problemas en un tiempo razonable y ofrece un mejor balance entre el rendimiento y el costo

computacional. Sin embargo, la implementación de esta es compleja al momento de programar y dificulta la sincronización y comunicación entre tareas. El paralelismo se ha utilizado para muchas temáticas diferentes, desde bioinformática para hacer plegamiento de proteínas, hasta economía para hacer simulaciones en matemática financiera.

En cuanto al principio de paridad, se puede mencionar que este ha sido implementado en muchísimas ocasiones en ejercicios y ejemplos de estructuras de datos. Inclusive, llegando al punto de crear un algoritmo llamado clasificación impar-par, como el que fue presentado para el desarrollo de este proyecto. Este algoritmo se divide en dos fases, siendo estas la fase impar y la par. El algoritmo se ejecuta hasta que los elementos de la matriz se ordenan y en cada iteración se producen las dos fases. La fase impar, es la encargada de realizar una clasificación del tipo bubblesort de los elementos indexados impares y la fase par realiza una clasificación del tipo bubblesort en elementos indexados pares.

Finalmente, es importante mencionar que el temario a desarrollar para esta entrega cuenta con mucha ventana de oportunidad para realizar la paralelización de este. Cabe mencionar que, este algoritmo utiliza fases de intercambio para comparar elementos que ocurren de forma simultánea en otras fases. Por estos motivos, el temario presentado es una correcta aplicación de los conceptos relacionados al paralelismo.

IV. Algoritmo Descriptivo

INICIO

1. Se define el encabezado
2. Se hace llamado a librerías
3. Se declaran variables globales
4. Se define la cantidad de elementos contenidos en la lista (modificable según especificaciones del tester)
5. Se declara la cantidad de hilos a implementar
6. Se define la variable condicional pthread
7. Se define la variable de tipo mutex
8. Se define una variable temporal para implementar durante el programa
9. Se define la variable condicional de pthread de la lista vacía
10. Se define una estructura con los parámetros a implementar

11. Se declaran las funciones vacías:
    - ObtencionNum
    - *ComparacionNum
    - Bubblesort
    - ClasificacionNum
12. Se define subrutina ObtencionNum
    - Se definen variables locales
    - Se solicita al usuario el ingreso del número
    - Se define un try catch para corroborar los datos ingresados
        1. Try: aumenta contador
        2. Catch: Muestra mensaje con el error
    - Ciclo for con la cantidad de elementos de n definidos
13. Se define la subrutina de Bubblesort
    - Se definen una variable local
    - Se define un ciclo for para asignar la variable de iteración con el mínimo
        1. Se define un segundo ciclo for para restarle 1 al máximo
            a) If para comparar posiciones
                (1) Ordena el array de menor a mayor
14. Se define la subrutina ClasificacionNum
    - Se definen variables locales
    - If para comparar posiciones
        1. Se bloquea la mutex
        2. Ordena el array de menor a mayor
        3. Desbloquea mutex
    - Se define el valor de retorno
15. Se define la subrutina Main
    - Se define variable máxima
    - Se define variable mínima
    - Se define una variable para la validación de num
    - Se realiza un ciclo for con la cantidad de elementos “n” definidos:
        1. Se itera la cantidad de veces correspondiente
        2. Por cada iteración que se toma un valor del array y este es asignado
    - Se inicializa la mutex

- Se imprime mensaje para mostrar lista inicial
- Se imprime la lista original sin ordenar
- Se utiliza la variable pthread con el id de la cantidad de hilos
- Se definen dos variables de iteración
- Se realiza un ciclo for para verificar mientras sea menor a la cantidad de elementos:
    1. Se verifica si el número es par o impar
        a) En caso sea par:
            (1) Se crean los hilos con los parámetros correspondientes
        b) En caso sea impar:
            (1) Se crean los hilos con los parámetros correspondientes.
        c) En caso exista error:
            (1) Mostrar mensaje de error.
- Se muestra mensaje de lista ordenada
- Se imprime cada uno de los elementos de forma ordenada
- Se realiza un ciclo for por la cantidad de elementos existente:
    1. Se imprime cada uno de los números de cada hilo
- Destrucción de hilos y mutex

FIN

I. Control de versiones

Como parte de los requerimientos del presente proyecto, se optó por implementar la herramienta de GitHub para llevar control del manejo de versiones. Esto, facilitó bastante la realización de los procedimientos a desarrollar por cada uno de los miembros del equipo, evitando así cualquier tipo de confusión.

https://github.com/aleg001/Proyecto2-Micro

II. Cuerpo

El presente proyecto tiene un enfoque en las operaciones paralelas para la clasificación par e impar. Entre las funciones se encuentra que, al realizar estas operaciones paralelas, no depende directamente de los resultados obtenidos de otros hilos durante su ejecución. Básicamente, el algoritmo desarrollado en código consiste en varias fases de proceso, en la cual se realiza la separación de datos, los cuales posteriormente se ordenan de

forma secuencial dentro del hilo. Posteriormente, se intercambian los valores locales que se clasificaron con su vecino correspondiente, para mandarlos a una pareja. Finalmente, este algoritmo regresa localmente las llaves intercambiadas, para su clasificación.

En cuanto a los retos encontrados para la implementación del paralelismo dentro de un algoritmo de clasificación, se encontró con bastantes conflictos en cuanto a la compilación. Por la naturaleza de este proyecto, no se pudo compilar en ningún compilador en línea como método de prueba. Sin embargo, esto fácilmente se solucionó implementando el uso de una Raspberry Pi para el corrimiento del programa. Cabe destacar que, adicional a esto, también se realizaron pruebas en una Mac, utilizando Clang. A diferencia del compilador en línea, se obtuvieron resultados prácticamente iguales que al correrlos en una Raspberry Pi, haciendo de esta una alternativa para correr el código.

Para futuras réplicas de este temario de proyecto, se recomienda tener bastante presente los conceptos de paralelismo para poder realizarlo de forma correcta. Adicional a esto, se recomienda realizar el algoritmo descriptivo o diagrama de flujo, previo al desarrollo del código. Al realizarlo de esta manera, resultará mucho más fácil programar este algoritmo de clasificación par e impar. Así mismo, la implementación de herramientas como GitHub para llevar un control de versiones, resulta bastante útil. Se usó esta herramienta como método de completar las distintas tareas o conflictos existentes con el código desarrollado.

III. Conclusiones

- Por medio de la investigación y la programación en paralelo, se cumplió con los objetivos propuestos en este proyecto, logrando así, la resolución del temario asignado.
- Se recomienda para futuras réplicas de este temario de proyecto, tomar en consideración que, adicionalmente al análisis previo, es necesaria una ardua investigación para contar con amplios conocimientos sobre el paralelismo, aplicandolo en la resolución de este temario.
- Se recomienda compilar este tipo de programas en una Raspberry Pi, esto, debido a que al correrlo en un compilador en línea, muestra mensajes de error.

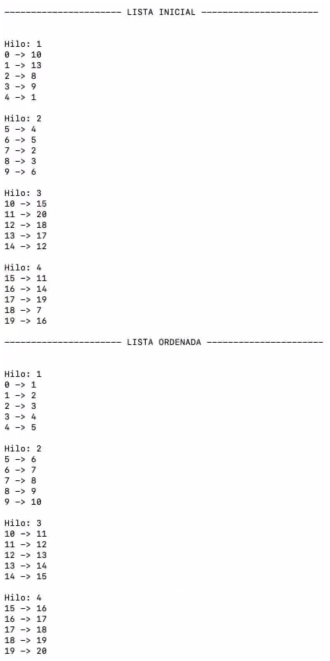
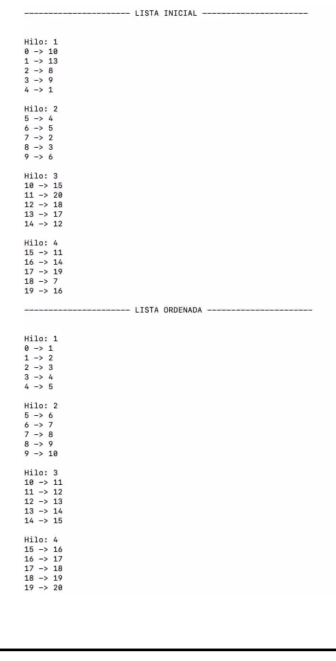
IV. Anexos

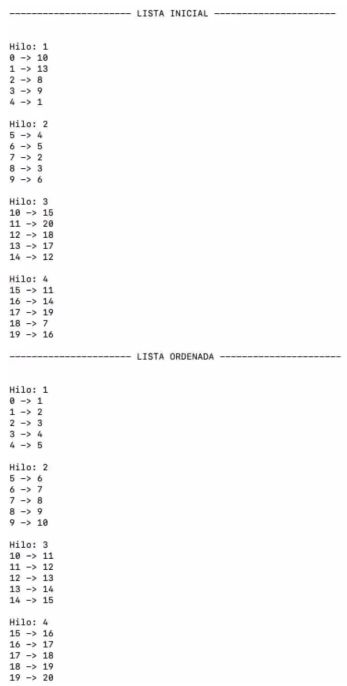
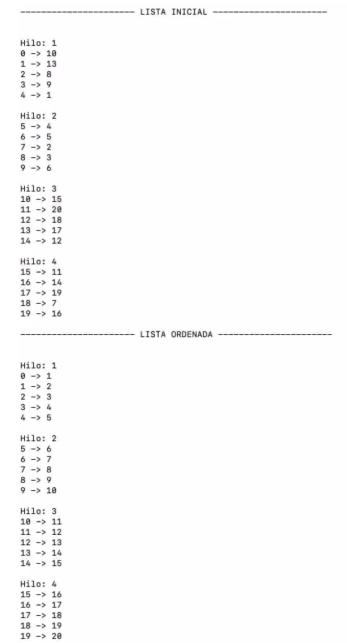
Tabla: No.1: Catálogo de funciones

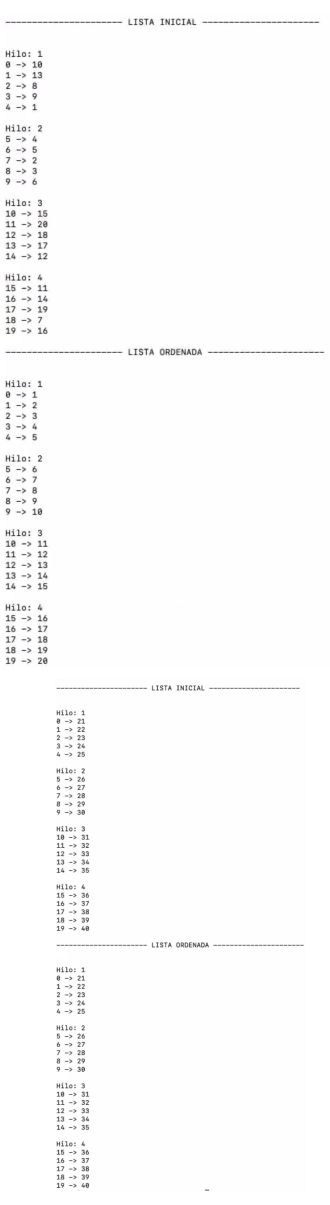
	Variables	Tipos	Descripción de uso
Entradas	n	Int	Solicita al usuario que ingrese la cantidad de números del array
Salidas	e	Exception	Utilizada como parte de la programación defensiva del programa desarrollado.
	varError	String	Utilizada para mostrar que existió un error al momento de crear los hilos
	hilo	Int	Utilizada para distintos ciclos for, con el fin de mostrar el hilo que se utiliza en el proceso pertinente.

	Nombre de la función	Descripción
1)	ObtencionNum	Esta función es utilizada para obtener el número que será ingresado en el proceso de ordenamiento del programa.
2)	ComparacionNum	En cuanto a la comparación de números, se implementa la función como método de ordenar el array utilizado de mayor a menor, implementando prácticas relacionadas al paralelismo como lo es el bloqueo y desbloqueo de mutex.
3)	Bubblesort	En cuanto a la función del Bubblesort, se puede mencionar que se encarga de ordenar utilizando uno de los algoritmos relacionados a la notación Big O, correspondiente al Bubblesor, así mismo, se encarga de revisar cada elemento de la lista a ordenar, con el siguiente elemento. Llegando a intercambiarlos de posición hasta que se encuentren en la posición correcta.
5)	Main	Se realizan todos los procesos relacionados a las funciones y requisitos del proyecto. Además, incluyendo el llamado a cada una de las subrutinas desarrolladas para la resolución de este temario.

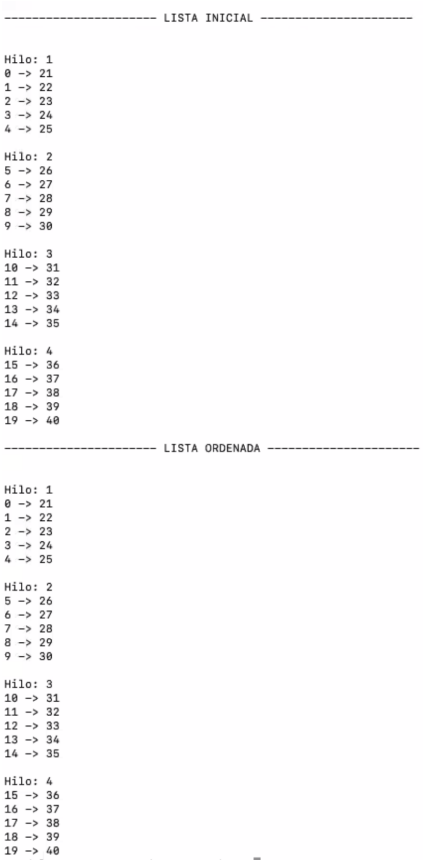
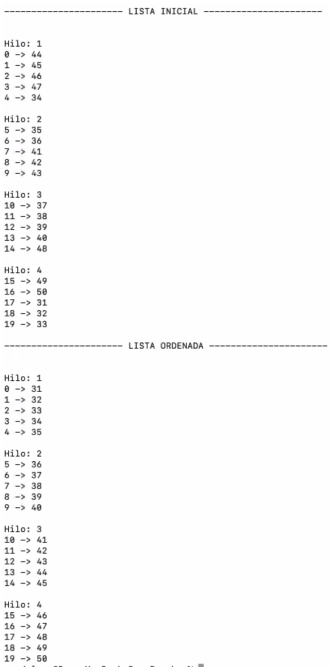
Tabla: No.2: Bitácora de pruebas:

Número de prueba	Resultado
Prueba No.1:	
Prueba No.2:	

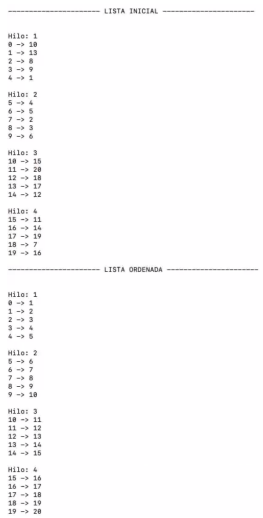
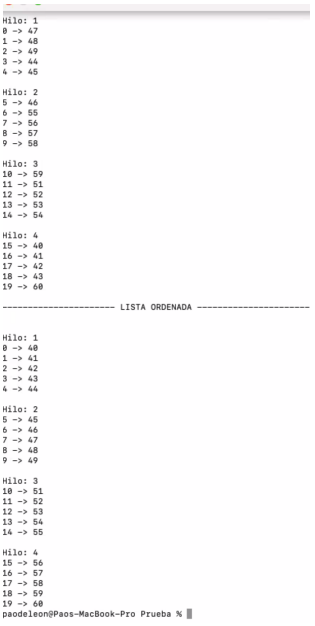
Prueba No.3:	
Prueba No.4:	
Prueba No.5:	

	 ---------- LISTA INICIAL ---------- Hilo: 1 0 -> 10 1 -> 13 2 -> 8 3 -> 9 4 -> 1 Hilo: 2 5 -> 4 6 -> 5 7 -> 2 8 -> 3 9 -> 6 Hilo: 3 10 -> 15 11 -> 20 12 -> 18 13 -> 17 14 -> 12 Hilo: 4 15 -> 11 16 -> 14 17 -> 19 18 -> 7 19 -> 16 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 1 1 -> 2 2 -> 3 3 -> 4 4 -> 5 Hilo: 2 5 -> 6 6 -> 7 7 -> 8 8 -> 9 9 -> 10 Hilo: 3 10 -> 11 11 -> 12 12 -> 13 13 -> 14 14 -> 15 Hilo: 4 15 -> 16 16 -> 17 17 -> 18 18 -> 19 19 -> 20 ---------- LISTA INICIAL ---------- Hilo: 1 0 -> 21 1 -> 22 2 -> 23 3 -> 24 4 -> 25 Hilo: 2 5 -> 26 6 -> 27 7 -> 28 8 -> 29 9 -> 30 Hilo: 3 10 -> 31 11 -> 32 12 -> 33 13 -> 34 14 -> 35 Hilo: 4 15 -> 36 16 -> 37 17 -> 38 18 -> 39 19 -> 40 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 21 1 -> 22 2 -> 23 3 -> 24 4 -> 25 Hilo: 2 5 -> 26 6 -> 27 7 -> 28 8 -> 29 9 -> 30 Hilo: 3 10 -> 31 11 -> 32 12 -> 33 13 -> 34 14 -> 35 Hilo: 4 15 -> 36 16 -> 37 17 -> 38 18 -> 39 19 -> 40
Prueba No.6:	

	Hilo: 1 0 -> 47 1 -> 48 2 -> 49 3 -> 44 4 -> 45 Hilo: 2 5 -> 46 6 -> 55 7 -> 56 8 -> 57 9 -> 58 Hilo: 3 10 -> 59 11 -> 51 12 -> 52 13 -> 53 14 -> 54 Hilo: 4 15 -> 40 16 -> 41 17 -> 42 18 -> 43 19 -> 60 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 40 1 -> 41 2 -> 42 3 -> 43 4 -> 44 Hilo: 2 5 -> 45 6 -> 46 7 -> 47 8 -> 48 9 -> 49 Hilo: 3 10 -> 51 11 -> 52 12 -> 53 13 -> 54 14 -> 55 Hilo: 4 15 -> 56 16 -> 57 17 -> 58 18 -> 59 19 -> 60
Prueba No.7:	---------- LISTA INICIAL ---------- Hilo: 1 0 -> 21 1 -> 22 2 -> 23 3 -> 24 4 -> 25 Hilo: 2 5 -> 26 6 -> 27 7 -> 28 8 -> 29 9 -> 30 Hilo: 3 10 -> 31 11 -> 32 12 -> 33 13 -> 34 14 -> 35 Hilo: 4 15 -> 36 16 -> 37 17 -> 38 18 -> 39 19 -> 40 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 21 1 -> 22 2 -> 23 3 -> 24 4 -> 25 Hilo: 2 5 -> 26 6 -> 27 7 -> 28 8 -> 29 9 -> 30 Hilo: 3 10 -> 31 11 -> 32 12 -> 33 13 -> 34 14 -> 35 Hilo: 4 15 -> 36 16 -> 37 17 -> 38 18 -> 39 19 -> 40

	
Prueba No.8:	
Prueba No.9:	

	`` ---------- LISTA INICIAL ---------- Hilo: 1 0 -> 63 1 -> 64 2 -> 65 3 -> 51 4 -> 52 Hilo: 2 5 -> 53 6 -> 54 7 -> 55 8 -> 56 9 -> 57 Hilo: 3 10 -> 58 11 -> 59 12 -> 60 13 -> 66 14 -> 67 Hilo: 4 15 -> 68 16 -> 69 17 -> 70 18 -> 61 19 -> 62 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 51 1 -> 52 2 -> 53 3 -> 54 4 -> 55 Hilo: 2 5 -> 56 6 -> 57 7 -> 58 8 -> 59 9 -> 60 Hilo: 3 10 -> 61 11 -> 62 12 -> 63 13 -> 64 14 -> 65 Hilo: 4 15 -> 66 16 -> 67 17 -> 68 18 -> 69 19 -> 70 ```
Prueba No.10:	``` Hilo: 1 0 -> 47 1 -> 48 2 -> 49 3 -> 44 4 -> 45 Hilo: 2 5 -> 46 6 -> 55 7 -> 56 8 -> 57 9 -> 58 Hilo: 3 10 -> 59 11 -> 51 12 -> 52 13 -> 53 14 -> 54 Hilo: 4 15 -> 40 16 -> 41 17 -> 42 18 -> 43 19 -> 60 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 40 1 -> 41 2 -> 42 3 -> 43 4 -> 44 Hilo: 2 5 -> 45 6 -> 46 7 -> 47 8 -> 48 9 -> 49 Hilo: 3 10 -> 51 11 -> 52 12 -> 53 13 -> 54 14 -> 55 Hilo: 4 15 -> 56 16 -> 57 17 -> 58 18 -> 59 19 -> 60 ```
Prueba No.11:	``` ---------- LISTA INICIAL ---------- Hilo: 1 0 -> 63 1 -> 64 2 -> 65 3 -> 51 4 -> 52 Hilo: 2 5 -> 53 6 -> 54 7 -> 55 8 -> 56 9 -> 57 Hilo: 3 10 -> 58 11 -> 59 12 -> 60 13 -> 66 14 -> 67 Hilo: 4 15 -> 68 16 -> 69 17 -> 70 18 -> 61 19 -> 62 ---------- LISTA ORDENADA ---------- Hilo: 1 0 -> 51 1 -> 52 2 -> 53 3 -> 54 4 -> 55 Hilo: 2 5 -> 56 6 -> 57 7 -> 58 8 -> 59 9 -> 60 Hilo: 3 10 -> 61 11 -> 62 12 -> 63 13 -> 64 14 -> 65 Hilo: 4 15 -> 66 16 -> 67 17 -> 68 18 -> 69 19 -> 70 ```

Prueba No.12:	
Prueba No.13:	

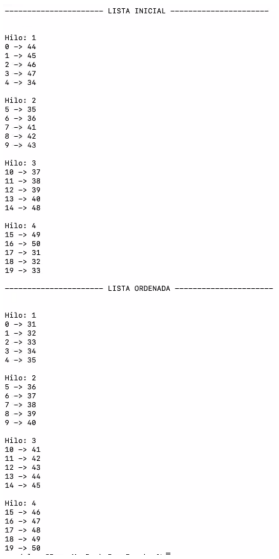
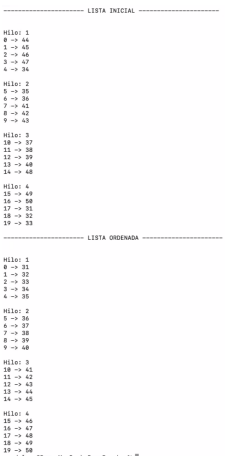
Prueba No.14:	
Prueba No.15:	

Imagen No.1: Referencia del código del libro Pachecho 3.8 & Grama et al 9.3

```
1.   procedure ODD-EVEN_PAR (n)
2.   begin
3.      id := process's label
4.      for i := 1 to n do
5.      begin
6.         if i is odd then
7.             if id is odd then
8.                 compare-exchange_min(id + 1);
9.             else
10.                compare-exchange_max(id - 1);
11.        if i is even then
12.            if id is even then
13.                compare-exchange_min(id + 1);
14.            else
15.                compare-exchange_max(id - 1);
16.     end for
17.  end ODD-EVEN_PAR
```

Imagen No.2: Referencia de código para la realización del sort

```
1  void Odd_even_sort(
2        int  a[]  /* in/out */,
3        int  n    /* in     */) {
4     int phase, i, temp;
5
6     for (phase = 0; phase < n; phase++)
7        if (phase % 2 == 0) { /* Even phase */
8           for (i = 1; i < n; i += 2)
9              if (a[i-1] > a[i]) {
10                temp = a[i];
11                a[i] = a[i-1];
12                a[i-1] = temp;
13             }
14       } else { /* Odd phase */
15          for (i = 1; i < n-1; i += 2)
16             if (a[i] > a[i+1]) {
17                temp = a[i];
18                a[i] = a[i+1];
19                a[i+1] = temp;
20             }
21       }
22 }  /* Odd_even_sort */
```

Program 3.15: Serial odd-even transposition sort

Imagen No.3: Planteamiento original del ordenamiento por fases

	Hilo 1					Hilo 2					Hilo 3					Hilo 4				
	18	14	12	2	5	10	15	3	6	9	1	7	8	4	11	13	16	20	19	17
Phase 1	0-4					5 -- 9					10--14					15--19				
	18	14	12	5	2	15	10	9	6	3	11	8	7	4	1	20	19	17	16	13
Phase 2	0--9										10--19									
	18	15	14	12	10	9	6	5	3	2	20	19	17	16	13	11	8	7	4	1
Phase 3	0-4					5--14										15--19				
	18	15	14	12	10	20	19	17	16	13	9	6	5	3	2	11	8	7	4	1
Phase 4	0--9										10--19									
	20	19	18	17	16	15	14	13	12	10	11	9	8	7	6	5	4	3	2	1

V. Bibliografía

- A. (2017, 24 octubre). *Principio de paridad*. Ilusiones Matemáticas. https://www.ilusionesmatematicas.com/principio-de-paridad/
- Grama, A. (2003). *Introduction to Parallel Computing (2nd Edition)*. TBS.
- Universidad de Carlos III Madrid. (s. f.). Programación con Hilos. Programación con Hilos. Recuperado 4 de octubre de 2021, de http://www.it.uc3m.es/pbasanta/asng/CES/M2/concurrent_1_es.pdf
- Pacheco, P. (2011). *An Introduction to Parallel Programming (English Edition)* (1.ª ed.). Morgan Kaufmann.
- Primaria, M. (2021, 14 junio). ▷ *Números pares e impares【¿Qué son? Curiosidades】*. Mundo Primaria.

https://www.mundoprimaria.com/recursos-matematicas/numeros-pares-e-impares

- GeeksforGeeks. (2021, 28 junio). *Odd-Even Sort / Brick Sort*. https://www.geeksforgeeks.org/odd-even-sort-brick-sort/
- EcuRed. (s. f.). Paralelismo (informática) - EcuRed. Recuperado 5 de octubre de 2021, de https://www.ecured.cu/Paralelismo_(_inform%C3%A1tica)
- C++ Multithreading - Tutorialspoint. (2010). tutorialspoint. Recuperado el 5 de octubre. https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm
- Rauber, T., & Runger, G. (n.d.). Parallel programming: For multicore and cluster systems. Springer Books.