

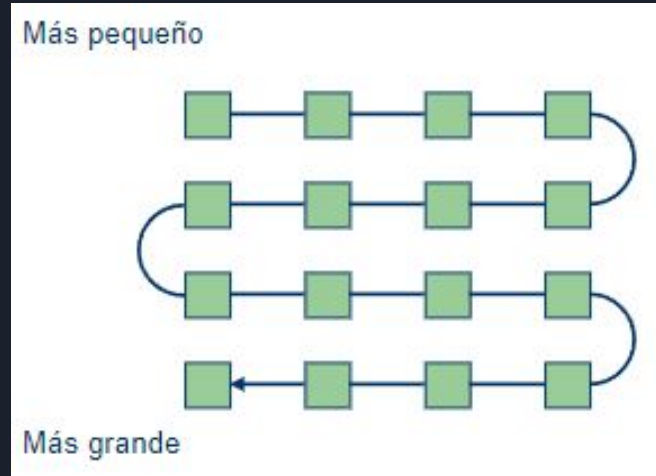
Algoritmos de ordenamiento paralelos

García Lopez Erik
Ramírez Medina Daniel

Snake sort (Shearsort)

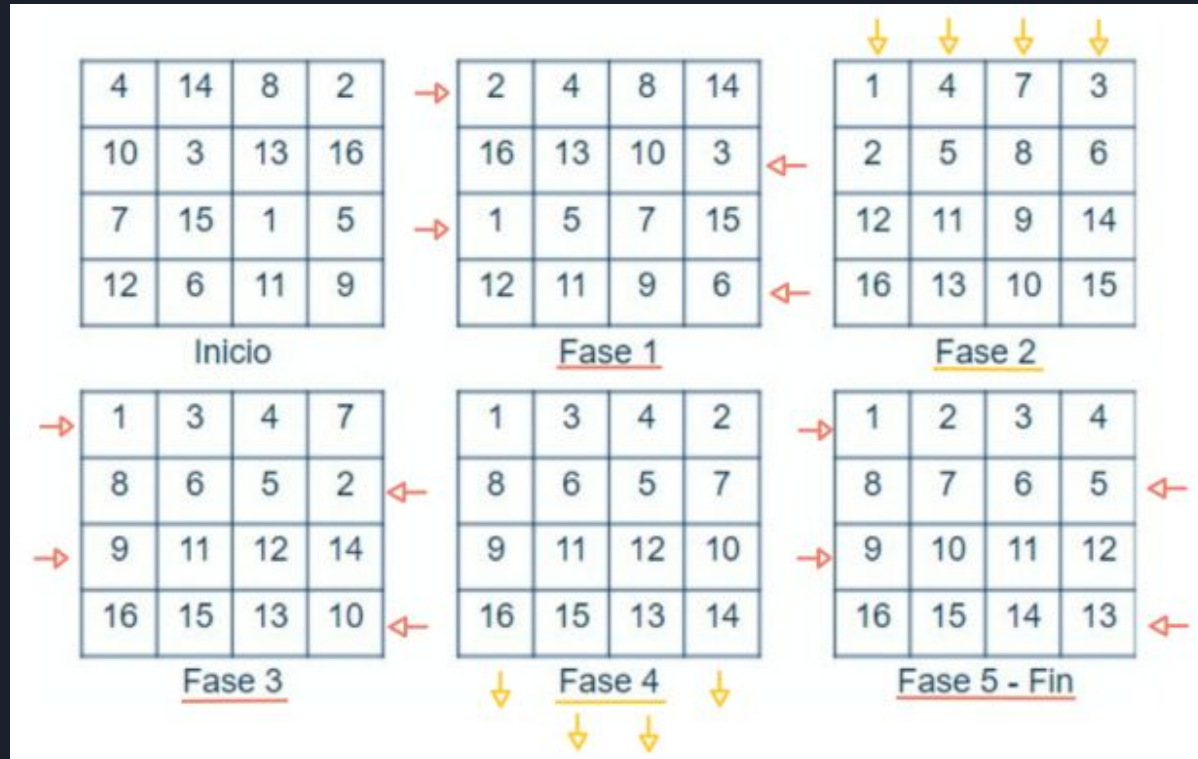
¿Qué es?

Es un algoritmo de ordenación donde los elementos de una matriz son ordenados en patrón de serpiente



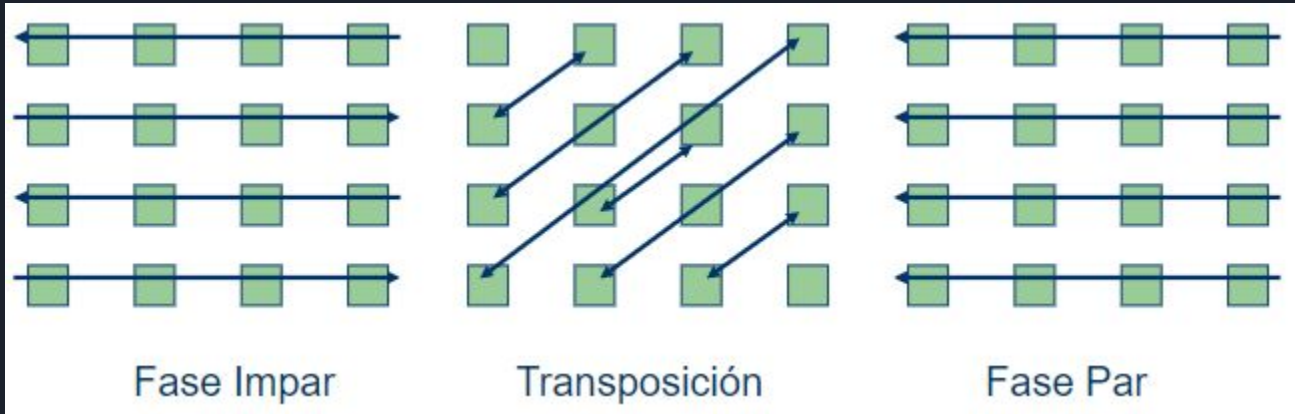
¿Cómo funciona la versión secuencial?

El algoritmo lo que hace es alternar entre ordenar las filas y ordenar las columnas



¿Cómo funciona la versión paralela?

- Teniendo un procesador por fila podemos paralelizar las fases impares
- Usando transposición entre fases podemos paralelizar también las fases impares



4	14	8	2
10	3	13	16
7	15	1	5
12	6	11	9

Inicio



2	4	8	14
16	13	10	3
1	5	7	15
12	11	9	6

Fase 1

2	16	1	12
4	13	5	11
8	10	7	9
14	3	15	6

Transposición



1	2	12	16
4	5	11	13
7	8	9	10
3	6	14	15

Fase 2

1	4	7	3
2	5	8	6
12	11	9	14
16	13	10	15

Transposición



1	3	4	7
8	6	5	2
9	11	12	14
16	15	13	10

Fase 3

1	8	9	16
3	6	11	15
4	5	12	13
7	2	14	10

Transposición



1	8	9	16
3	6	11	15
4	5	12	13
2	7	10	14

Fase 4

1	3	4	2
8	6	5	7
9	11	12	10
16	15	13	14

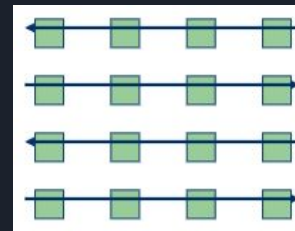
Transposición

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

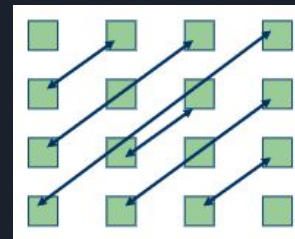
Fase 5 - Fin

Paralelización

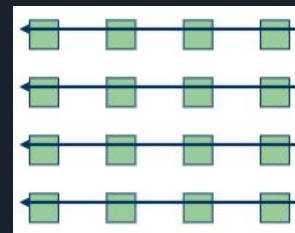
Fase impar: Cada procesador compara y ordena los elementos de su fila. Como cada procesador trabaja de forma independiente en su propia fila, esta fase se puede paralelizar sin necesidad de comunicación entre procesadores.



Fase de transposición: La matriz se transpone para permitir la comparación y ordenamiento de los elementos en las columnas. Para paralelizar esta fase, se pueden usar técnicas de transposición eficientes como la transposición en bloques, que permite la transposición paralela de submatrices.



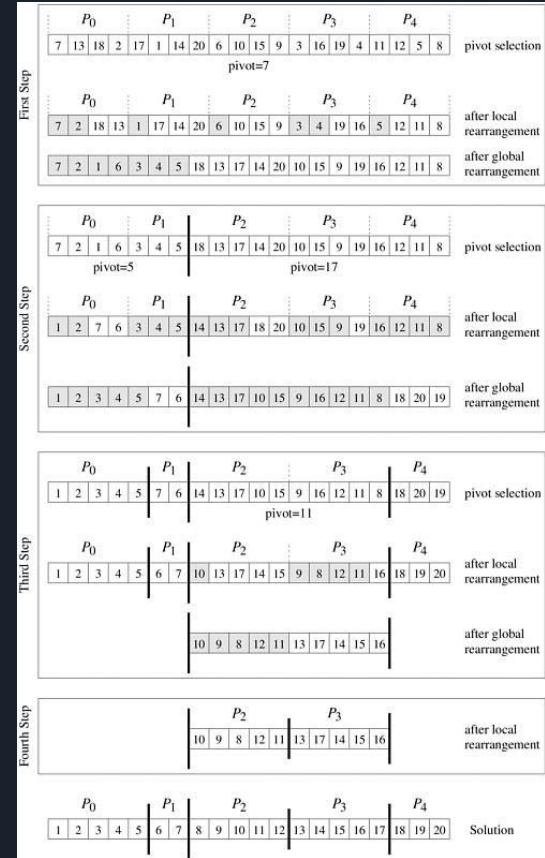
Fase par: los procesadores vuelven a comparar y ordenar los elementos de su fila. Como en la fase impar, esta fase se puede paralelizar sin necesidad de comunicación entre procesadores.



Parallel Quick Sort

¿Qué es?

El Parallel Quick Sort u “Ordenamiento Rápido Paralelo”, es una variante del algoritmo de ordenamiento rápido (QS) que utiliza técnicas de paralelismo para mejorar el rendimiento en sistemas multi-core o distribuidos.



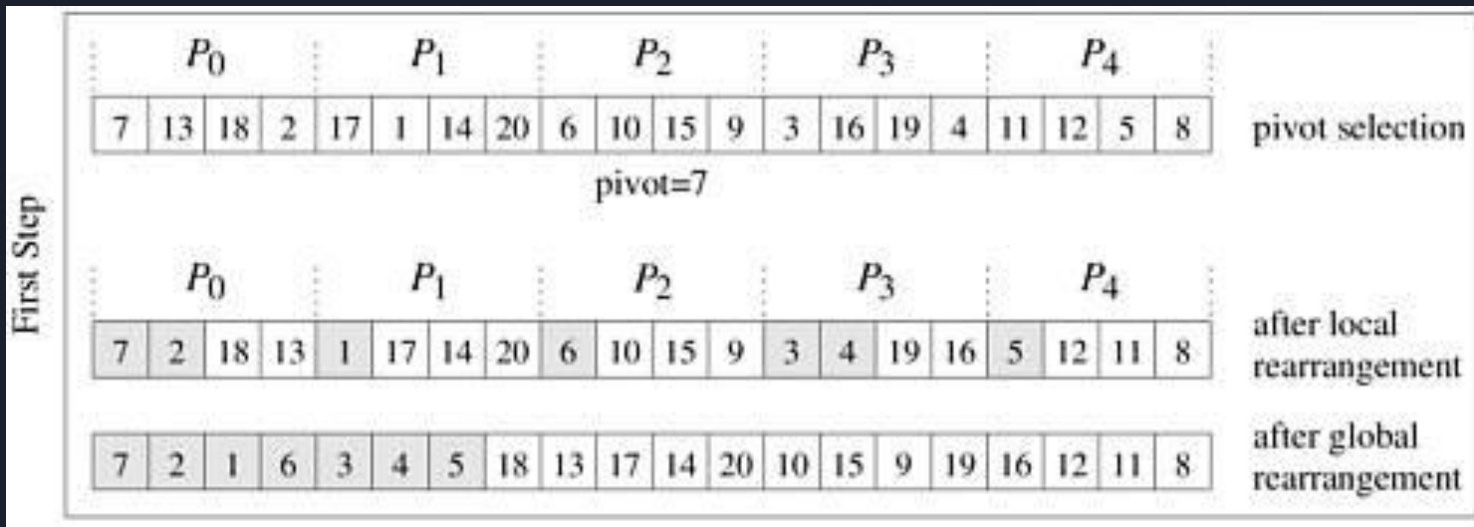


Algoritmo del Parallel Quick Sort

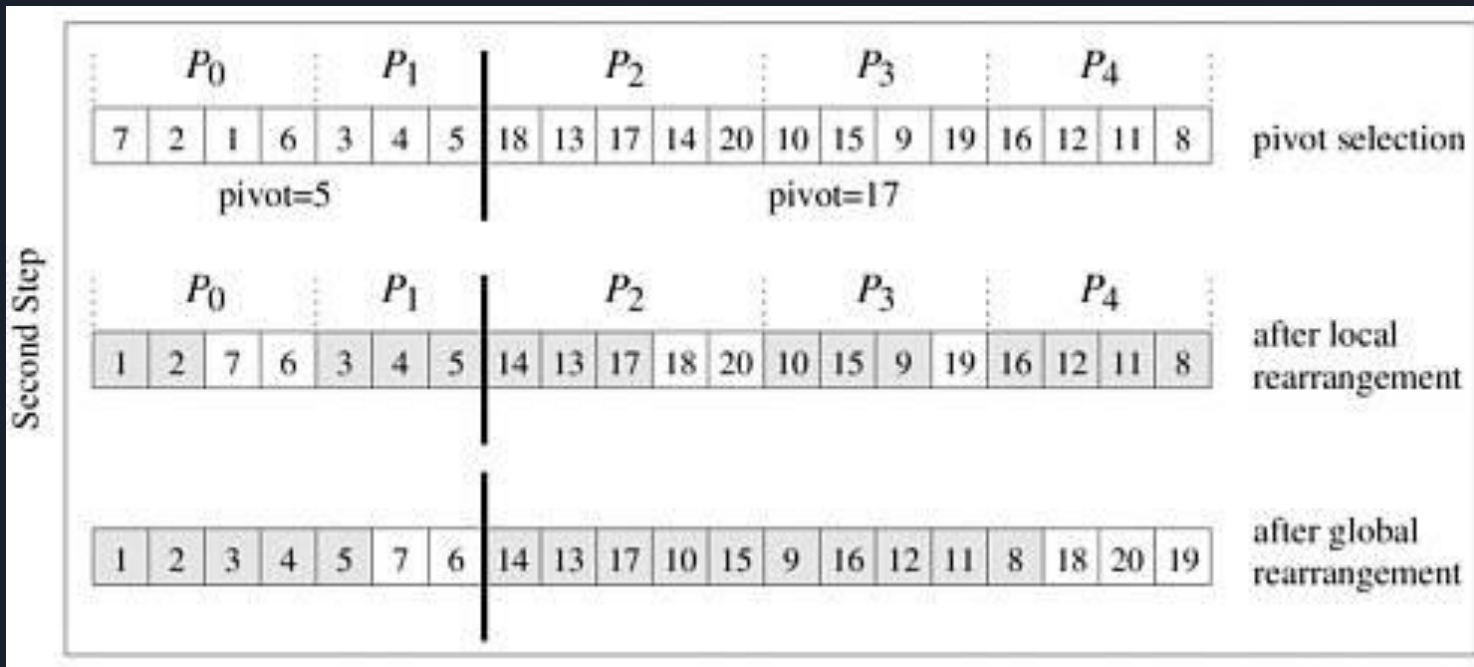
La clave para lograr el paralelismo en el Quick Sort es elegir un pivote adecuado y dividir la lista en subconjuntos que se puedan procesar de forma independiente. Una vez que se hayan ordenado los subconjuntos de manera individual, se puede realizar una etapa de fusión o combinación para obtener la lista finalmente ordenada.

Complejidad: $O(n \log n)$.

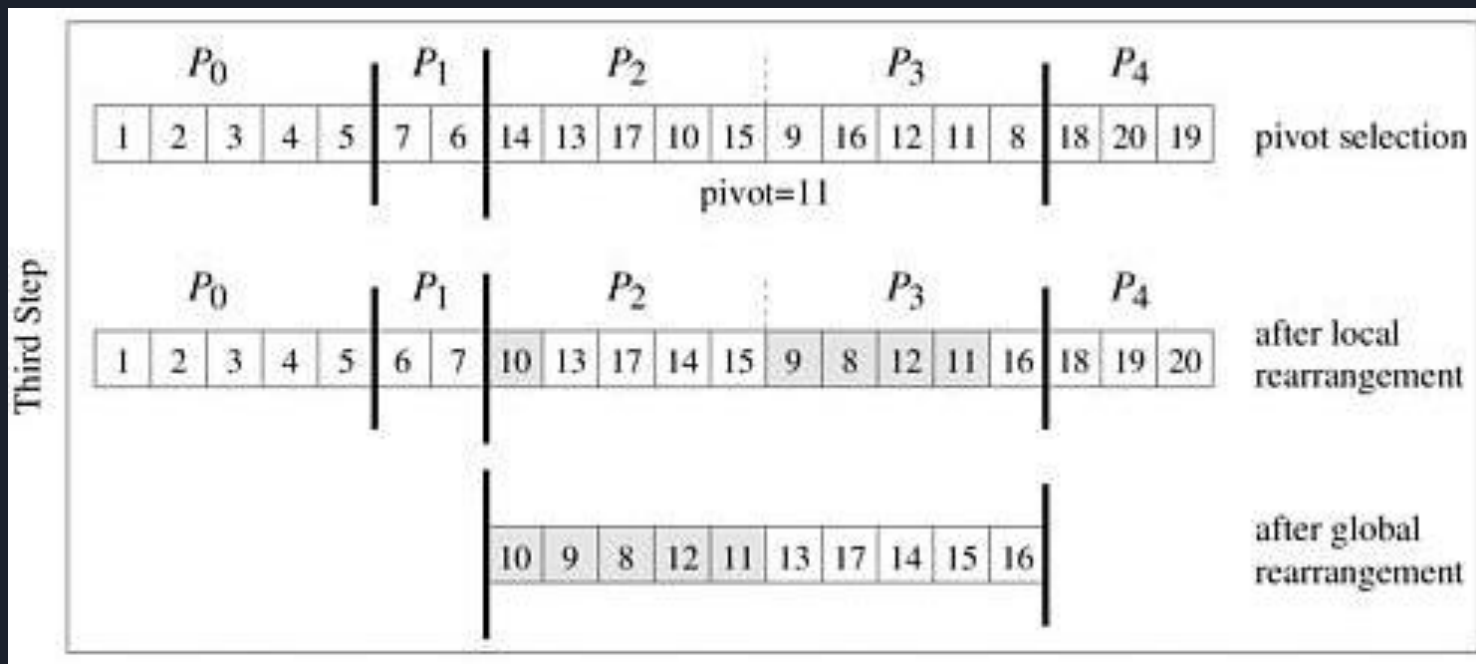
Algoritmo



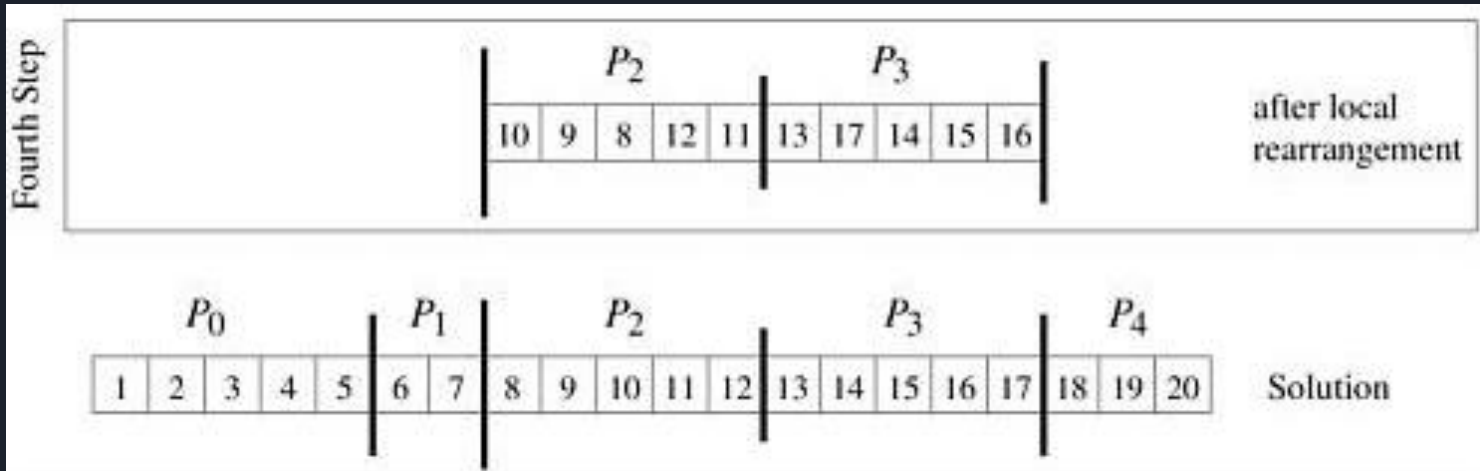
Algoritmo



Algoritmo



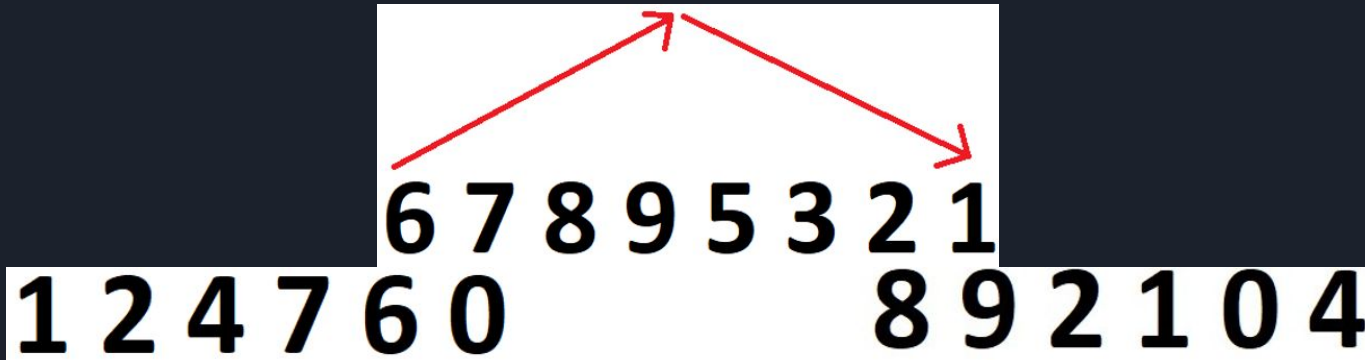
Algoritmo



Bitonic Sort

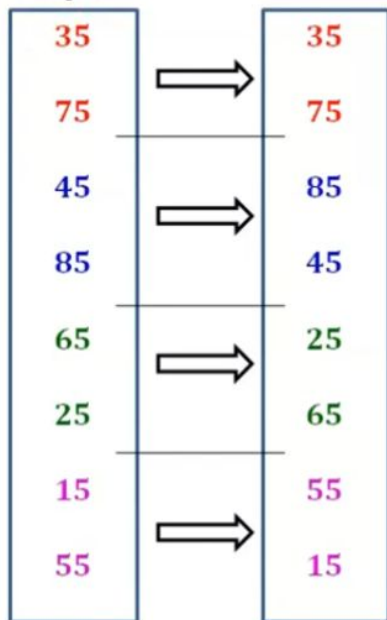
¿Qué es una secuencia bitónica?

Una secuencia es llamada bitónica cuando los números primero aumentan y luego disminuyen o viceversa.



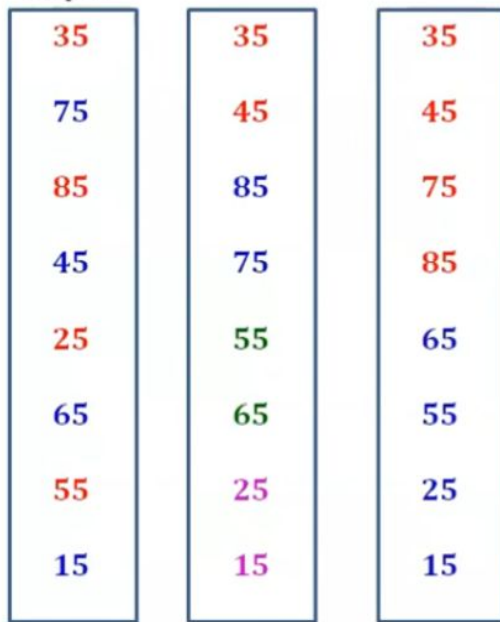
Es un algoritmo de ordenamiento utilizado para convertir una secuencia de números aleatorios a una secuencia bitónica y posteriormente en una secuencia de números ordenada

Step 1



Stage 1

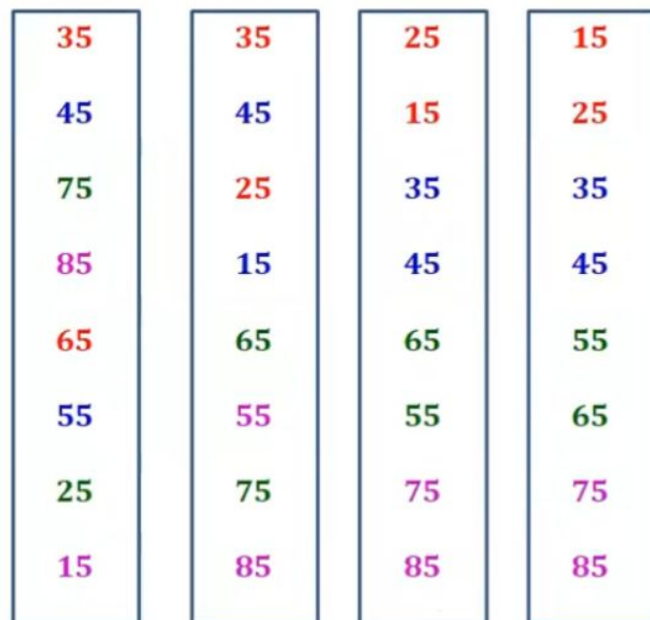
Step 2



Stage 2

Stage 1

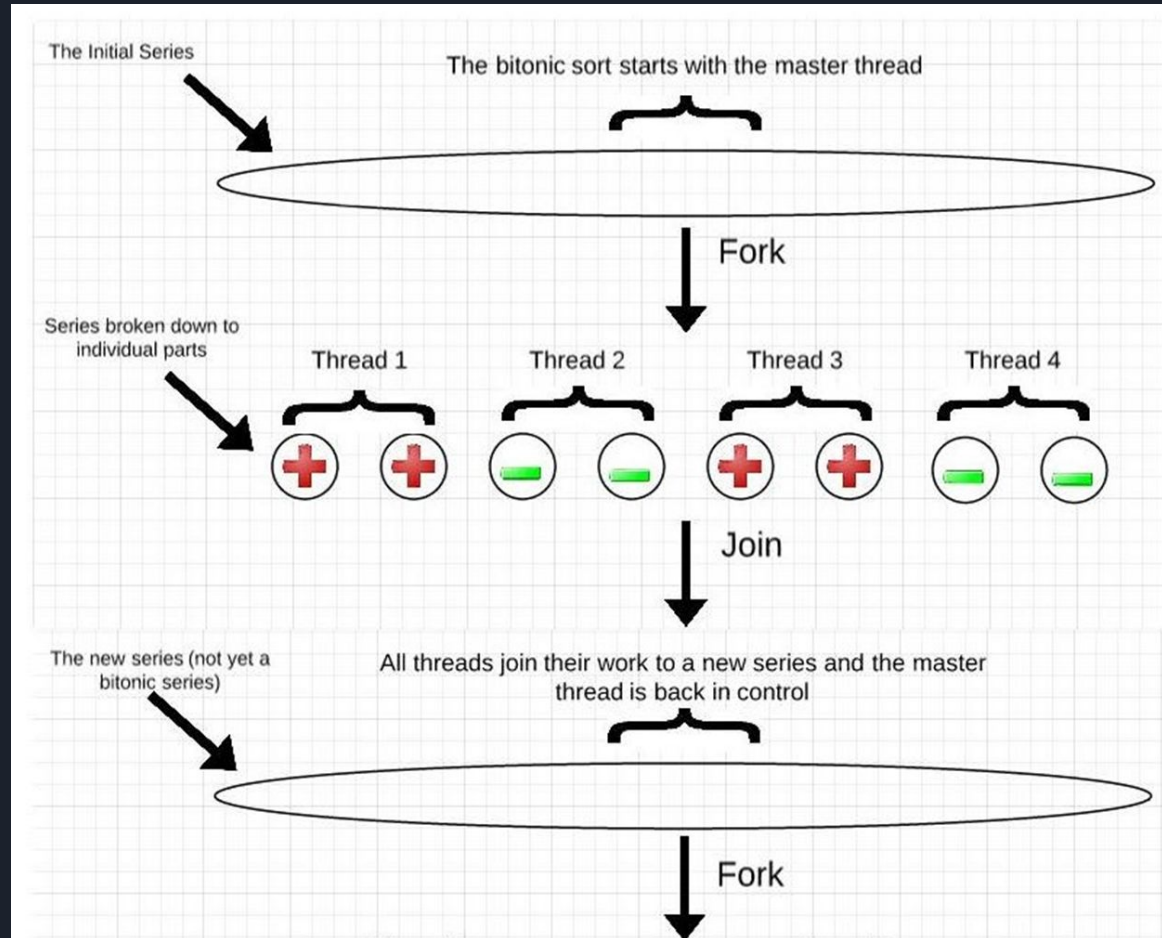
Step 3:

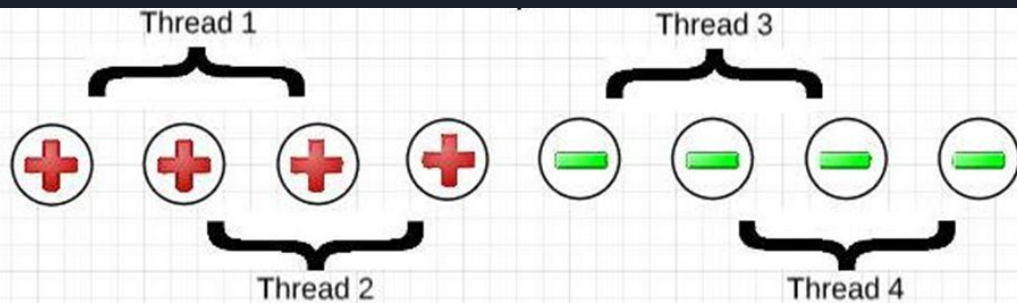


Stage 3

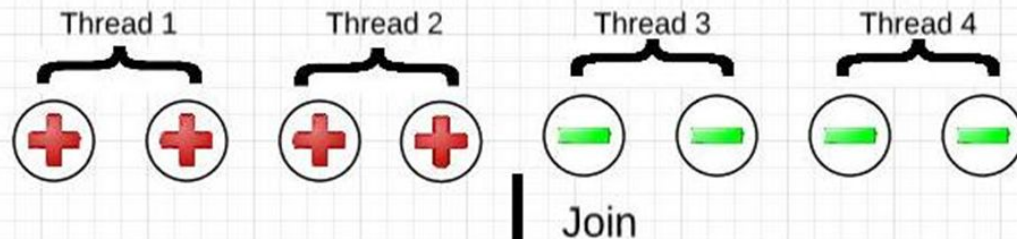
Stage 2

Stage 1





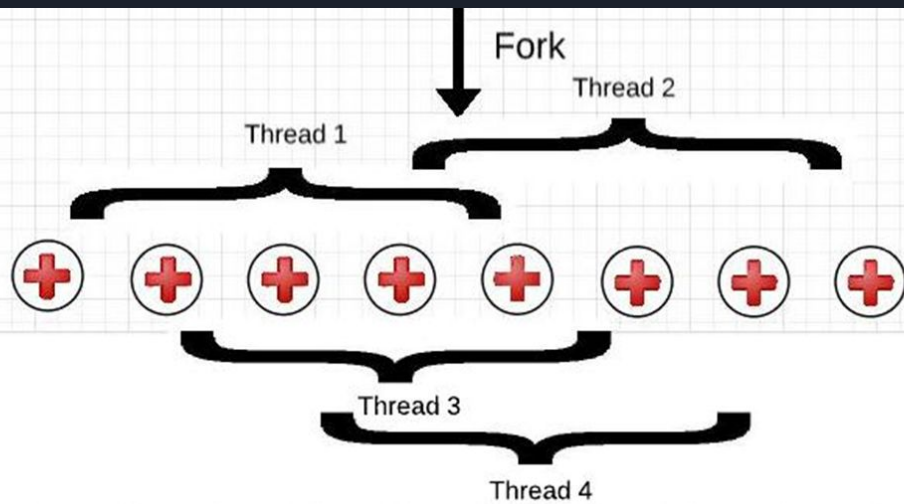
Wait until all thread are done



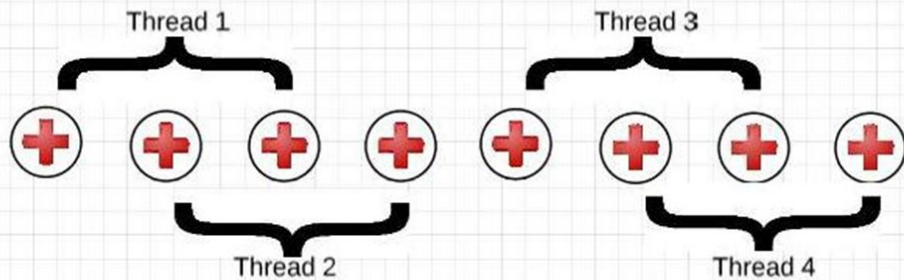
The new series (now a bitonic series)

All threads join their work to a new series and the master thread is back in control





Wait until all thread are done



Wait until all thread are done

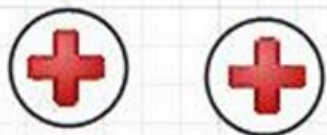
Wait until all thread are done

Thread 1

Thread 2

Thread 3

Thread 4



Join

Sorted Series!!!!!

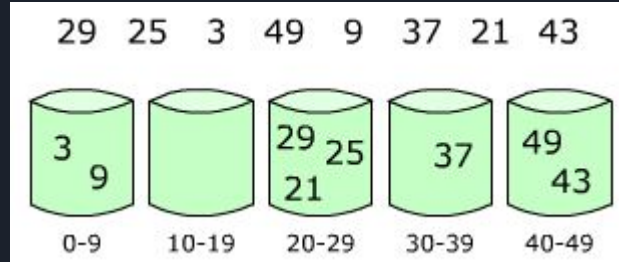
Parallel Bucket Sort

¿Qué es?

Bucket Sort u “Ordenamiento por Casillero” es un algoritmo bastante fácil de implementar cuando se habla de algoritmos paralelos. Su lógica principal es reservar diferentes partes de una matriz en diferentes buckets, luego ordenarlos al mismo tiempo con otro algoritmo y volver a juntarlos en una gran matriz de resultados.

A diferencia del bucket sort clásico, que es un algoritmo secuencial, el Parallel Bucket Sort está diseñado específicamente para aprovechar el paralelismo y distribuir la carga de trabajo en múltiples hilos o procesadores.

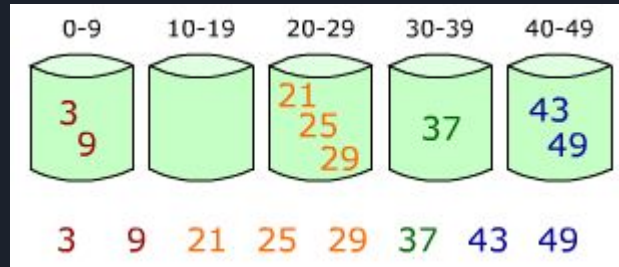
complejidad: $O(n)$.

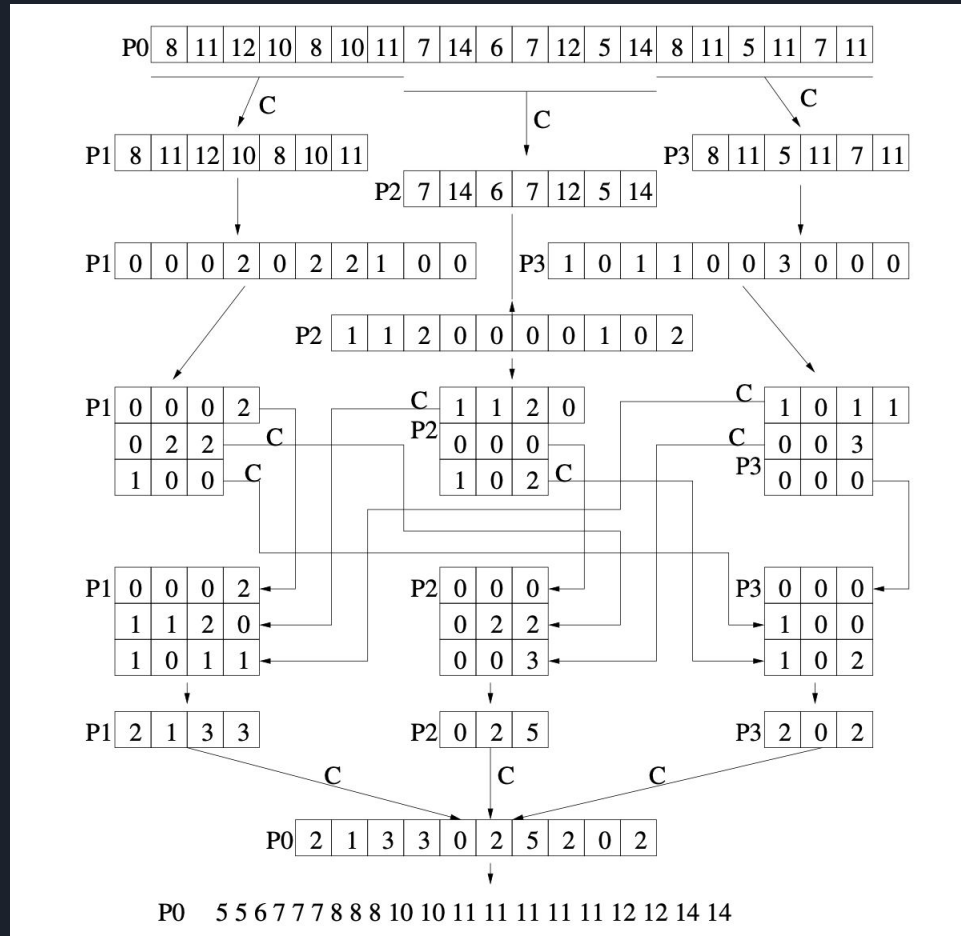


Parallel Bucket Sort

Ponemos los números en los buckets según el intervalo que se repite desde el valor mínimo hasta el final de todos los números, que es el número máximo en la matriz. En este caso, es posible que vea que todos los números negativos estarán en el primer cubo, lo que podría ser en algún caso una desventaja de esta implementación en particular de este algoritmo.

Al principio, los números se calculan con la misma lógica y el proceso principal envía información de cuántos números necesitará cada cubo para analizar y obtener. Al hacer la misma rutina de nuevo, el proceso principal envía todos los números a un cubo en particular y los cubos están recibiendo información hasta que obtiene todos los números que se le deben enviar.







Master generates random numbers:

P0: 8 11 12 10 8 10 11 7 14 6 7 12 5 14 8 11 5 11 7 11

Slaves receive their part of the unsorted array:

P1: 8 11 12 10 8 10 11

P2: 7 14 6 7 12 5 14

P3: 8 11 5 11 7 11

Slaves perform bucket-sort into arrays of size $\max - \min + 1$

		content of local buckets												
		5	6	7	8		9	10	11		12	13	14	
P1:	8 8 10 10 11 11 12	=	[0	0	0	2	--	0	2	2	--	1	0	0]
P2:	5 6 7 7 12 14 14	=	[1	1	2	0	--	0	0	0	--	1	0	2]
P3:	5 7 8 11 11 11	=	[1	0	1	1	--	0	0	3	--	0	0	0]

P1 handles [5,6,7,8]

P2 handles [9,10,11]

P3 handles [12,13,14]



9 10 11

P1: Send [0 2 2] to P2

12 13 14

P1: Send [1 0 0] to P3


5 6 7 8

P1: Receive [1 1 2 0] from P2

P1: Receive [1 0 1 1] from P3

5 6 7 8

P1: Send [2 1 3 3] to P0



5 6 7 8
P2: Send [1 1 2 0] to P1

12 13 14
P2: Send [1 0 2] to P3

9 10 11
P2: Receive [0 2 2] from P1
P2: Receive [0 0 3] from P3

9 10 11
P2: Send [0 2 5] to P0



5 6 7 8

P3: Send [1 0 1 1] to P1

9 10 11

P3: Send [0 0 3] to P2


12 13 14

P3: Receive [1 0 0] from P1

P3: Receive [1 0 2] from P2

12 13 14

P3: Send [2 0 2] to P0



5 6 7 8
P0: Receive [2 1 3 3] from P1

9 10 11
P0: Receive [0 2 5] from P2

12 13 14
P0: Receive [2 0 2] from P3

5 6 7 8 9 10 11 12 13 14
P0: Now has [2 1 3 3 0 2 5 2 0 2]

P0: Output: 5 5 6 7 7 7 8 8 8 10 10 11 11 11 11 11 12 12 14 14



Bibliografía

<https://slideplayer.es/slide/3173643/>

<https://iq.opengenus.org/parallel-merge-sort/>

https://www.youtube.com/watch?v=32NLIL_6WJg

<https://www.geeksforgeeks.org/bitonic-sort/>

<http://users.atw.hu/parallelcomp/ch09lev1sec4.html>

<https://www.smaizys.com/programing/bucket-sort-parallel-algorithm-using-c-openmpi/>

<http://www.egr.unlv.edu/~matt/teaching/CSC789/assignment6.pdf>