Um das Experiment durchzuführen, benötigen wir zwei unabhängige und eine abhängige Variable. Als unabhängige Variablen nehmen wir unsere Implementierungen und die Längen der zu sortierenden Listen. Als abhängige Variable wird die Ausführungszeit genommen.

Wir nehmen sechs Listen, die jeweils 10, 100, 1000, 5000, 10000 und 50000 Elementen groß sind, und befüllen sie mit zufälligen Elementen. Jede Liste wird von jeder Implementierung sortiert. Die zu drei Zeitpunkten gewonnenen Messungen werden protokolliert.

Die daraus resultierenden Werte sind in der folgenden Tabelle aufgeführt.

Tabelle 1 – bei den Messungen gewonnene Werte

		Sequenziell			Runnable		
		M1	M2	M3	M1	M2	M3
Länge	10	0	1	0	6	7	6
	100	1	2	2	8	4	7
	1000	15	16	15	12	16	11
	5000	362	373	359	140	151	159
	10000	1464	1426	1468	608	553	555
	50000	103119	110200	99616	16561	15716	17214

Tabelle 1 - bei den Messungen gewonnene Werte (Fortsetzung)

		ThreadPoolExecutor			ForkJoinPool		
		M1	M2	M3	M1	M2	M3
Länge	10	16	13	12	4	5	5
	100	4	5	4	7	3	7
	1000	11	12	10	15	17	15
	5000	144	149	139	120	131	132
	10000	529	532	528	546	481	532
	50000	17103	19147	17005	16601	22818	19761

Die aus drei Messungen gebildeten Mittelwerte sind in der nächsten Tabelle aufgeführt.

Tabelle 2 - Mittelwerte

	Sequenziell	Runnable	ThreadPoolExecutor	ForkJoinPool
10	0,3	6,3	13,7	4,7
100	1,7	6,3	4,3	5,7
1000	15,3	13,0	11,3	15,7
5000	364,7	150,0	144,0	127,7
10000	1452,7	572,0	529,7	519,7
50000	104311,7	16497,0	17751,7	16090,3

Um unsere Implementierungen besser vergleichen zu können, stellen wir sie in Form eines Balkendiagramms dar.

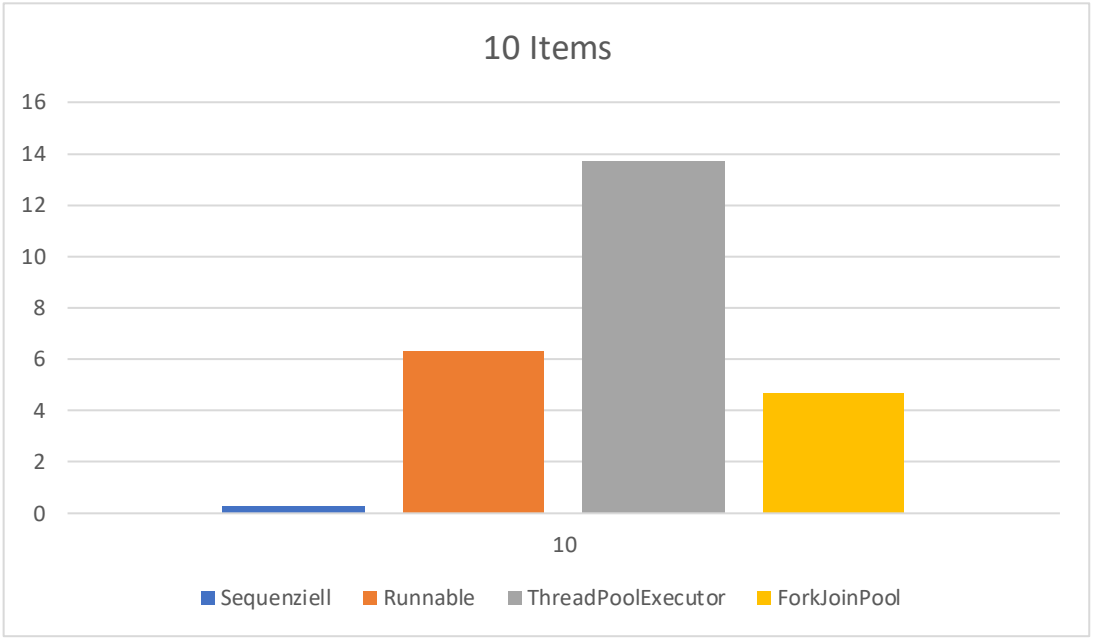

Abb. 1 - Ausführungszeit einer Liste in der Länge von 10 Items in Millisekunden

Bei nur 10 Items ist der Favorit die sequenzielle Implementierung. Die langsamste Variante ist der ThreadPoolExecutor.

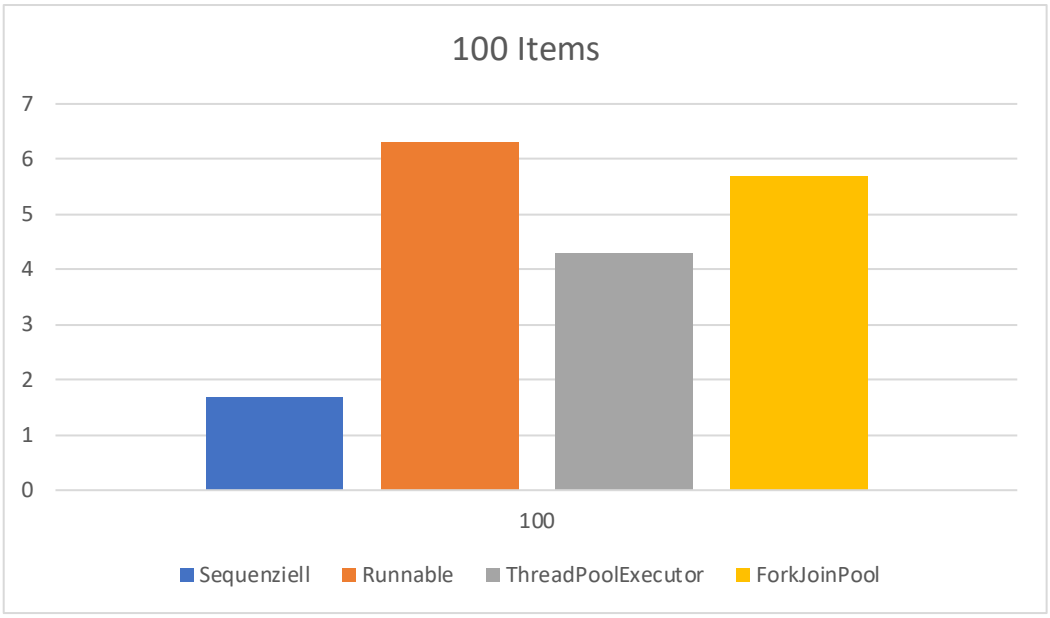

Abb. 2 - Ausführungszeit einer Liste in der Länge von 100 Items in Millisekunden

Bei den Listen mit 100 Items ist der Favorit immer noch die sequenzielle Implementierung.

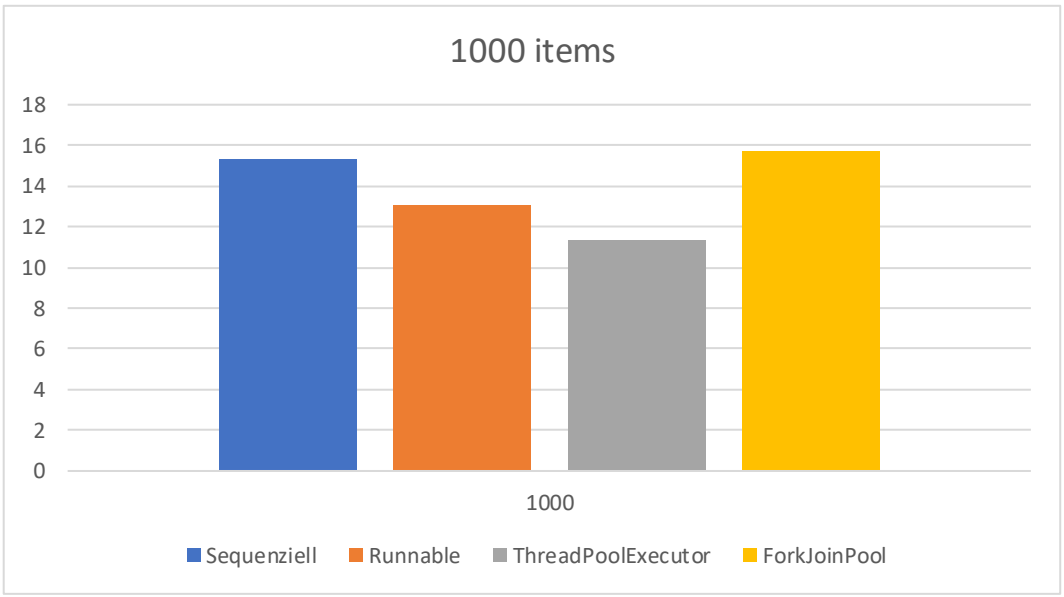

Abb. 3 - Ausführungszeit einer Liste in der Länge von 1000 Items in Millisekunden

Bei den Listen mit 1000 Items ist der Aufwand fast gleich. Der Unterschied liegt bei Millisekunden.

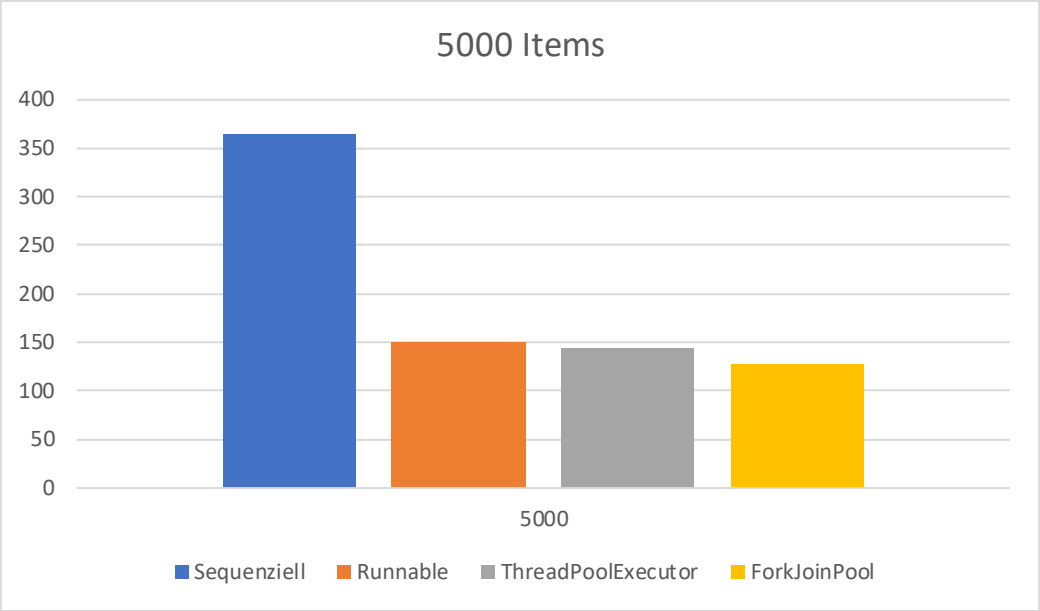

Abb. 4 - Ausführungszeit einer Liste in der Länge von 5000 Items in Millisekunden

Hier sehen wir, dass die sequenzielle Variante die langsamste ist. Die anderen Implementierungen haben fast den gleichen Aufwand.

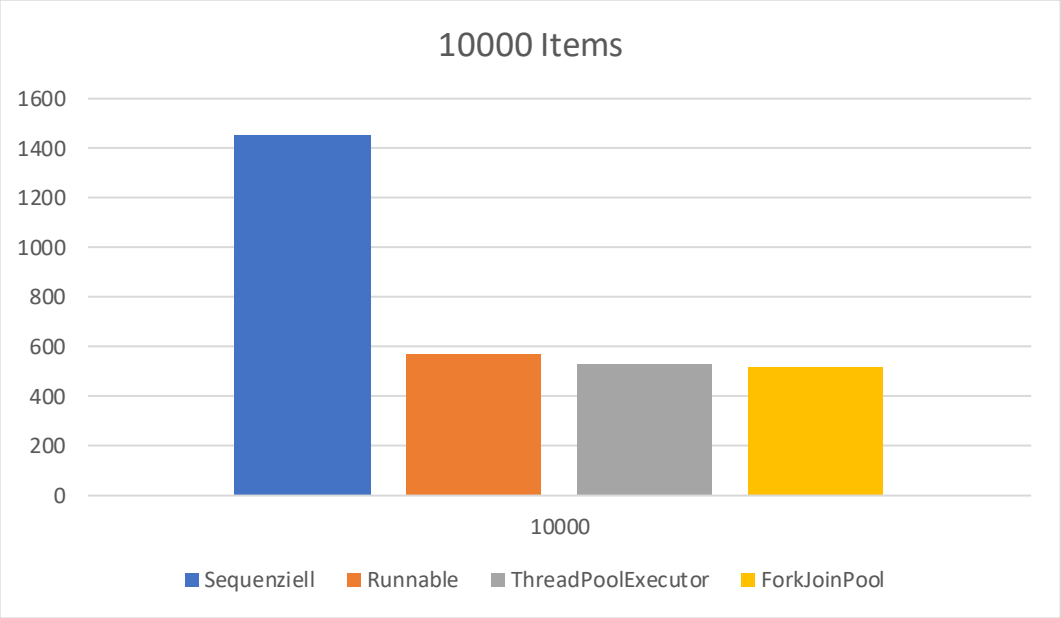

Abb. 5 - Ausführungszeit einer Liste in der Länge von 10000 Items in Millisekunden

Beim Sortieren von einer Liste mit 10000 Items (genauso wie mit 5000 Items) ist die langsamste Variante die sequenzielle Implementierung.

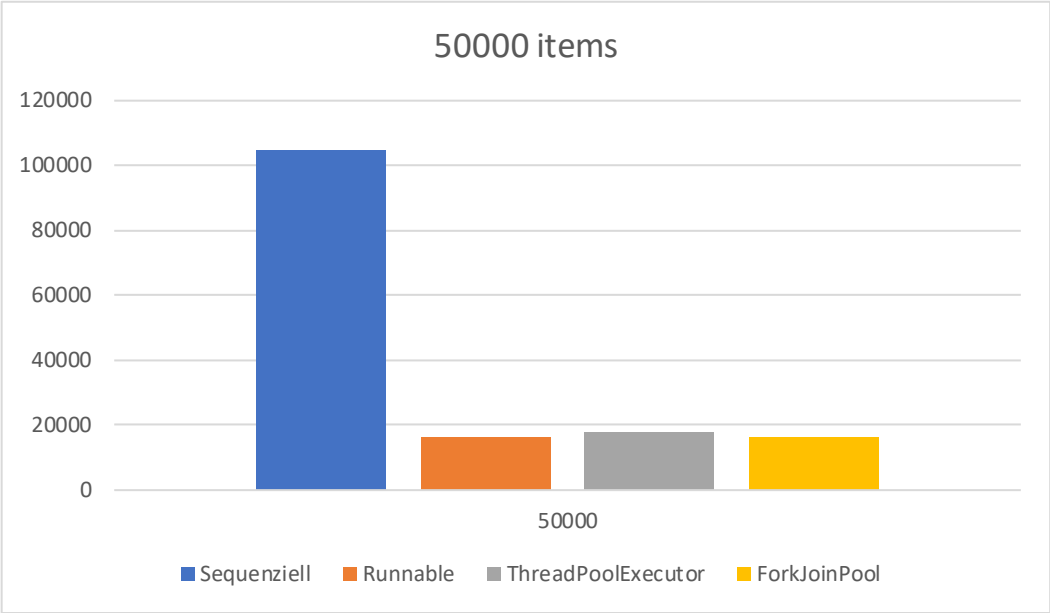

Abb. 6 - Ausführungszeit einer Liste in der Länge von 50000 Items in Millisekunden

Bei 50000 Items ist die Situation nicht anders.

Aus oben dargestellten Diagrammen ist zu erkennen, dass die sequenzielle Implementierung die beste Variante nur bei kleinen Aufgaben ist. Sind also nur wenige Daten zu bearbeiten, lohnt es sich nicht die parallele Implementierung. Je größer die Liste ist, desto ist es sinnvoller die Aufgabe parallel zu lösen.