NitscheSahoo - AD Praktikum für 17.10.13 – Aufgabe 1 - Listen

Das Interface:

```
public interface IList<T> {
	// add elem to the front
	void cons(T elem);

	// removes first element and return first
	T head();

	// Not in interface but a simple get(index) method
	// throws outofbound when n > arrlength-1 && n < 0
	T get(int index);

	// get first element
	T first();

	// how many elements has a list?
	int length();

	// is the list empty?
	boolean isempty();

	// insert element after index n so between n and n+2 if there is a current
	// n+2
	void insert(T elem, int n);

}
```

Akkumulation:

„Müssen Sie in Ihrer Lösung alle bisherigen Messwerte mitspeichern? Wenn Ja: Geht das auch anders? Wie?“

Unter Verwendung des Satz von Steiner brauchen wir akkumulierend nur zwei Werte zu speichern und zur Rückgabe der Varianz/des Durchschnitts nur kleine Berechnungen zu machen. Zum Vergleich der expliziten und iterativen Implementation haben wir jedoch dennoch die Speicherung aller Werte und die explizite Varianten implementiert. Wir setzten diese jedoch nicht ein

Am Anfang hinzufügen:

```
	// 15 elements need: 6414 ns.
	// AccessCount: 29
	// 150 elements need: 62860 ns.
	// AccessCount: 299

	// 15 elements need: 7270 ns.
	// AccessCount: 29
	// 150 elements need: 65426 ns.
	// AccessCount: 299

	// 15 elements need: 6842 ns.
	// AccessCount: 29
	// 150 elements need: 59867 ns.
	// AccessCount: 299

	// These observations fit to a O(n)-Algorithm. Indeed, the time needed for
	// inserting elements at the front grows proportionaly to the number of elements.
	// 10x the elements needs approx. 10x the time.
	// The initial overhead can also be recognized.
```

Am Ende inserten:

```
	// 15 elements need:  36775 ns.
	// AccessCount: 153
	// 150 elements need:  546501 ns.
	// AccessCount: 11628
	//
	// 15 elements need:  33354 ns.
	// AccessCount: 153
	// 150 elements need:  484923 ns.
	// AccessCount: 11628
	//
	// 15 elements need:  23946 ns.
```

```
        // AccessCount: 153
        // 150 elements need:  426339 ns.
        // AccessCount: 11628

        // The number of Accesses suspects a quadratic complexity.
        // It can be shown, that inserting at the end in a
        // linked list is indeed of quadratic complexity.
```

An einer zufälligen Stelle hinzufügen:

```
T=1:
369039.0 nanoseconds on average
with a variance of 0.0 nanoseconds*nanoseconds

T=5:
168911.4 nanoseconds on average
with a variance of 107824.53600317506 nanoseconds*nanoseconds

T=10:
139790.2 nanoseconds on average
with a variance of 78926.47220152295 nanoseconds*nanoseconds

T=50:
131511.4 nanoseconds on average
with a variance of 309610.4800676396 nanoseconds*nanoseconds

T=100:
76395.01 nanoseconds on average
with a variance of 265690.87653643073 nanoseconds*nanoseconds

T=500:
25400.826 nanoseconds on average
with a variance of 105097.84907858778 nanoseconds*nanoseconds

T=1000:
15841.725 nanoseconds on average
with a variance of 72155.23965360272 nanoseconds*nanoseconds

T=5000:
9664.2108 nanoseconds on average
with a variance of 46556.242115225985 nanoseconds*nanoseconds
```

Aufwandsabschätzung:

Arbeitszeit: 9h gemeinsam – viel davon ist organisatorisches

Zeitverteilung: 10% AvgVarianz - 60% Listen und Experimente - 30% Tests und Javadoc

Vorschau der Javadoc: