2.Übung

## Aufgabe 1:

1. **Why does the output from the different algorithms differ between the sorted and the shuffled list?**

FOR und DELTA beziehen sich immer auf die vorherigen Elemente in der Liste. Wenn die Liste also nicht mehr sortiert ist, ändert sich auch das Ergebnis erheblich. Gerade weil dadurch auch negative Zahlen auftreten können, komprimieren wir nicht wirklich, sondern verschlimmern das Ganze.

1. **Why does Delta encoding needs fewer bits than Dictionary encoding? Is this always the case?**

In dieser Value Liste liegen die Werte nah beieinander, deswegen komprimiert DELTA besser als DICT. DICT hängt nur von der Länge der Liste ab, während DELTA zusätzlich von der Verteilung der Werte abhängt. Also in einem Fall mit einer kurzen Liste und sich stark ändernden Werten würde DICT um einiges besser abschneiden als DELTA.

1. **What is the optimal frame size for FOR (minimal number of needed bits)?**

Die Länge der Liste ist die Optimale frame size. Dadurch verhält es sich Analog zu DELTA.

1. **Do we save memory by using those algorithms?**

Bei FOR und DELTA schaffen wir es auf jeden Fall Speicher zu sparen, weil die Zahlen anstatt zu direkt im Speicher, relativ zueinanderstehen und das verbraucht in den meisten fällen weniger Speicher. Bei DICT wiederrum muss man ja auch das DICT irgendwo speichern, um die eigentlichen Werte wieder zu bekommen. Also muss man für die Encoding-Liste erstmal Speicher investieren, kann aber bei z.B. sich sehr oft wiederholenden großen Zahlen auch zu mehr freien Speicher führen.

1. **What happens if negative numbers are part of the input data, and why does it happen?**

Negative Zahlen brauchen um einiges mehr Bits um dargestellt zu werden. Und da die Zahlen relativ zueinanderstehen, kann es auch schnell passieren, dass man mehr negative Zahlen erzeugt als nötig und damit wieder viel Speicher verbraucht.

## Aufgabe 2:

1. **Why do we need caching in the first place?**

Wir brauchen Caches damit wir auf Daten die häufig genutzt werden, auch schnell wieder zugreifen können. Wenn wir keinen Cache hätten, müssten wir die Daten immer aus dem Hauptspeicher holen und das ist sehr langsam. Durch diesen Flaschenhals würde die Performance signifikant sinken. Das Lokalitätsprinzip von Daten unterstützt die Notwendigkeit von Caches.

1. **What happens when k in LRU-K becomes bigger?**Man speichert den k-ten Zugriff auf die Page. Wenn wir ein höheres K verwendet können wir besser entscheiden welche Page am wenigsten genutzt wird, weil wir einen größeren Zeitraum in Betracht ziehen, aber zum Preis, dass bei einem größeren k mehr Zugriffe gespeichert werden müssen, und das führt zu mehr Speicherverbrauch. Wenn z.B. k=4 wäre müssten die letzten 4 Zugriffe einer Page gespeichert werden, damit wenn die Page nochmal aufgerufen wird, man das nächste k zur Verfügung hat.
2. **In which scenario LIFO performs better than FIFO?**

## Aufgabe 3:

**Uncompressed:**  
Execution time - 12c.sql: 1.4357197284698486 Seconds.

Execution time - 21a.sql: 1.6578071117401123 Seconds.

Execution time - 25a.sql: 3.037773370742798 Seconds.

Execution time - 6a.sql: 1.9201064109802246 Seconds.

Execution time - 9a.sql: 2.0303149223327637 Seconds.

**Compressed:**

Ich konnte leider keinen Tablespace mit der „compressed=true“ Eigenschaft erstellen…