3. Test SWP-OP Fragen

1. Was ist die O-Notation?

A: Mit dem Symbol O (ein großes Omikron) wird das Verhalten von Funktionen beschrieben. Mit O(n) wird das Wachstumsverhalten eines Algorithmus (nicht eines Programms!) in Abhängigkeit von der Eingabegröße erfasst.

1. Beschreibe den Algorithmus von Bubblesort!

A: Bei Bubblesort werden die Elemente einer Liste oder des Arrays von links nach rechts durchlaufen. Dabei wird bei jedem Schritt das aktuelle Element mit dem rechts davon liegenden Nachbaren verglichen. Ist das rechte Element (bei aufsteigender Sortierung kleiner als das aktuelle Element, dann werden diese beiden Elemente in der Liste/ in dem Array vertauscht. Wenn das rechte Element (bei aufsteigender Sortierung) jedoch größer als das aktuelle Element wird nicht getauscht. Dieser Vergleich wird dann so lange durchgeführt bis das größte Element sich bei aufsteigender Sortierung bzw. das kleinste Element bei absteigender Sortierung sich ganz rechts befindet. Wenn dies der Fall ist wird das letzte Element der Liste/ des Arrays beim nächsten Durchlauf nicht mehr beachtet, da dieses Element nicht mehr getauscht werden muss. Das bedeutet, dass sich bei der Liste/ beim Array der Wertebereich um 1 Element verkleinert wird.

Beim 2. Durchlauf werden wieder alle Elemente (bis auf das letzte) miteinander verglichen und der Wertebereich wieder um ein Element verkleinert.

Der Algorithmus läuft dann so lange durch bis die komplette Liste aufsteigend oder absteigend sortiert ist.

Bsp Wikipedia:

Eine Reihe von fünf Zahlen soll aufsteigend sortiert werden.

Die fett gedruckten Zahlen werden jeweils verglichen. Ist die linke größer als die rechte, so werden beide vertauscht; das Zahlenpaar ist dann blau markiert. Im ersten Durchlauf wandert somit die größte Zahl ganz nach rechts. Der zweite Durchlauf braucht somit die letzte und vorletzte Position nicht mehr zu vergleichen. → Dritter Durchlauf: kein Vergleich letzte/vorletzte/vorvorletzte…

**55 07** 78 12 42   1. Durchlauf  
07 **55 78** 12 42  
07 55 **78 12** 42  
07 55 12 **78 42**   Letzter Vergleich  
**07 55** 12 42 78   2. Durchlauf  
07 **55 12** 42 78  
07 12 **55 42** 78   Letzter Vergleich  
**07 12** 42 55 78   3. Durchlauf  
07 **12 42** 55 78   Letzter Vergleich  
**07 12** 42 55 78   4. Durchlauf + Letzter Vergleich  
07 12 42 55 78   Fertig sortiert.

1. Komplexität des Bubblesort Algorithmus!

A: Best Case: Alle Elemente sind bereits geordnet und es muss nur 1 Durchlauf durchgeführt werden, der das linke und rechte Element miteinander zu vergleichen.  
  
Onotation: O(n)

Worst Case: Beim Worst Case handelt sich um eine umgekehrt sortierten Liste (das beudetet Sortierung absteigend, wollen aber aufsteigend sortieren)

Onotation: O(n2)

Average Case: Falls die Elemente der Liste bereits nah den Stellen sind, die sie nach der Sortierung bekommen sollen, ist die Laufzeit erheblich besser als O(n2).

1. Eigenschaften von Bubblesort!

A: Bubblesort ist ein in-place Algorithmus, da es für die Speicherung der zu bearbeitenden Daten benötigten Speicher nur eine konstant, also von der Datenmenge unabhängige Menge an Speicherplatz benötigt.

Bubblesort ist ein stabiler Sortieralgorithmus, da er die Reihenfolge der Datensätze, deren Sortiertschlüssel gleich sind, bewahrt.

(Bei Sortierung nach Datum wird alphabetische Ordnung beibehalten.)

1. Erkläre den Algorithmus bei Insertionsort!

A: Da das 1.Element in einer Liste/einem Array immer geordnet ist, vergleicht beim ersten Schritt die ersten beiden Elemente der jeweiligen Ordnungsstruktur. Ist das rechte Element bei aufsteigender Sortierung kleiner als das aktuelle Element, wird dieses an die 1. Stelle der Datenstruktur verschoben und die danach folgenden Elemente werden in der Datenstruktur nach hinten verschoben. Danach sind die ersten beiden Elemente sortiert und man überprüft das 3. Element der Datenstruktur. Ist dieses größer als das 1. Element aber kleiner als das 2. Element wird es in der Mitte eingeordnet. In diesem Fall müsste das vorherige 2. Element jetzt auf die 3. Position aufrücken. Die nachfolgenden Elemente werden auch wieder um 1 Element verschoben. Wenn das Element am kleinsten ist wird es an die 1. Stelle gesetzt. (Wieder Aufrücken der nachfolgenden Stellen!)   
Ist das Element jedoch größer wird dieses übersprungen und das nächste Element angeschaut. Dieses wird dann wieder eingeordnet oder stehen gelassen. Der Vorgang wird so lange wiederholt bis die Datenstruktur geordnet ist.

Siehe Beispiel Wikipedia!

1. Komplexität des Insertionsort Algorithmus!

A: Best Case: bereits geordnete Liste 🡪 O(n)

Worst Case: andere Sortierung (aufsteigend statt absteigend) 🡪 0(n2)

Average Case: O(n2)

1. Eigenschaften des Insertionsort!  
   A: Insertionsort ist ein in-place Algorithmus, da es für die Speicherung der zu bearbeitenden Daten benötigten Speicher nur eine konstant, also von der Datenmenge unabhängige Menge an Speicherplatz benötigt.

Insertionsort ist ein stabiler Sortieralgorithmus, da er die Reihenfolge der Datensätze, deren Sortiertschlüssel gleich sind, bewahrt.

(Bei Sortierung nach Datum wird alphabetische Ordnung beibehalten.)

1. Erkläre den Algorithmus von Selectionsort!

A: Bei Selectionsort wird beim 1. Schritt das kleinste Element gesucht und dieses an den Anfang der Datenstruktur gesetzt. Danach weiß man, dass es kein kleineres Element mehr in der Liste gibt und die Liste beginnt an der 2.Stelle und sucht dann in der verkleinerten Datenstruktur das kleinste Element und schiebt dieses dann an die 2. Stelle des Arrays. Der sortierte Bereich wird wiederum größer und man beginnt dann ab dem 3. Element erneut das kleinste Element zu suchen usw. Dies passiert dann so lange bis es nur ein Element in Frage kommt, welches dann automatisch das höchste bzw. niedrigste Element darstellt.

Siehe Bsp Wikipedia!

1. Komplexität von Selectionsort!

A: O(n2)

1. Eigenschaften von Selectionsort!

A: Selectionsort ist ein in-place Algorithmus, da es für die Speicherung der zu bearbeitenden Daten benötigten Speicher nur eine konstant, also von der Datenmenge unabhängige Menge an Speicherplatz benötigt.

Selectionsort ist in seiner Grundform kein stabiler Sortieralgorithmus, da er die Reihenfolge der Datensätze, deren Sortiertschlüssel gleich sind, nicht bewahrt.

(Bei Sortierung nach Datum wird alphabetische Ordnung beibehalten.)

1. Was ist ein in place Algorithmus und was sind seine Eigenschaften?

A: Ein Algorithmus arbeitet in-place bzw. in situ, wenn er außer dem für die Speicherung der zu bearbeitenden Daten benötigten Speicher nur eine konstante, also von der zu bearbeitenden Datenmenge unabhängige Menge von Speicher benötigt.

Beispiele: Bubblesort, Selectionsort, Insertionsort

1. Was ist ein out of place Algorithmus und was sind seine Eigenschaften?  
   A: Ein Algorithmus arbeitet out-of-place, wenn die Ausgabedaten gesondert gespeichert und nicht die Eingabedaten damit überschrieben werden. Algorithmen, die direkt auf den Eingabedaten arbeiten und diese modifizieren, arbeiten in-place.

Durch mehr vollständige oder teilweise Speicherrepräsentationen der zu bearbeitenden Daten verbraucht ein Out-of-Place-Algorithmus normalerweise mehr Arbeitsspeicher als ein In-Place-Algorithmus.

Beispiele: Mergesort

1. Was ist ein stabiler Sortieralgorithmus?  
   A: Ein stabiles Sortierverfahren ist ein Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, bewahrt.

Wenn bspw. eine Liste alphabetisch sortierter Personendateien nach dem Geburtsdatum neu sortiert wird, dann bleiben unter einem stabilen Sortierverfahren alle Personen mit gleichem Geburtsdatum alphabetisch sortiert.

Beispiele: Bubblesort, Insertionsort, Mergesort, …

1. Was ist ein instabiler Sortieralorithmus?  
   A: Ein instabiles Sortierverfahren ist ein Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, nicht bewahrt.

Wenn bspw. eine Liste alphabetisch sortierter Personendateien nach dem Geburtsdatum neu sortiert wird, dann bleiben unter einem instabilen Sortierverhalten die Sortierung nach Personendateien nicht erhalten.

Beispiele: Selectionsort, Quicksort, Heapsort, …

1. Was ist ein wahlfreier Zugriff?  
   A: Unter wahlfreiem Zugriff (englisch random access, auch „direkter Zugriff“, „Direktzugriff“) wird in der Informatik die Möglichkeit verstanden, in konstanter (oder unter-linearer) Zeit einen lesenden und/oder schreibenden Speicherzugriff auf ein beliebiges Element eines Datenspeichers oder einer Datenstruktur durchführen zu können.

Beispiele: Array

1. Was ist ein sequenzieller Zugriff?

A: Der Begriff sequentieller Zugriff (auch sequenzieller Zugriff) bezeichnet eine Zugriffsart auf einen Datenspeicher oder eine Datenstruktur, bei der die Datensätze aufeinanderfolgend angeordnet sind. Um auf einen bestimmten Datensatz zugreifen zu können, müssen zunächst alle zwischen Ausgangs- und Zielposition befindlichen Datensätze aufgesucht werden. Die Zugriffszeit ist dabei vor allem von der Entfernung der Datensätze abhängig.

Beispiele: Listen

1. Welche Datenstruktur eignet sich besser für Insertionsort?  
   Ein Array oder eine Liste und warum tut sie das?  
   A:
2. Wann sollte man Listen und wann Arrays verwenden und was sind die Vorteile bzw. Nachteile von Arrays und Listen?

A:

1. Können Elemente aus einem Array bzw. aus einer Liste gelöscht werden? Wenn, ja wie funktioniert das?

A: