Übung 3 - Suchen und Sortieren

3.1. Selection Sort

3.1.1. Sortieren Sie die folgenden Zahlen mit **Selection Sort** in **absteigender** Reihenfolge,
Die Sortierung kann in n-1 Runden durchgeführt werden. Vervollständigen Sie die Tabelle, indem Sie die Array Elemente nach jeder Runde eintragen.

	A[1]		Array A			A[6]
Runde	3	9	6	1	5	4
1	9	3	6	1	5	4
2	9	6	3	1	5	4
3	9	6	5	1	3	4
4	9	6	5	4	3	1
5	9	6	5	4	3	1

3.1.2. Sortieren Sie die folgenden Zahlen mit **Selection Sort** in **aufsteigender** Reihenfolge
Die Sortierung kann in n-1 Runden durchgeführt werden. Vervollständigen Sie die Tabelle, indem Sie die Array Elemente nach jeder Runde eintragen.

	A[1]		Array A			A[6]
Runde	3	9	6	1	5	4
1	1	9	6	3	5	4
2	1	3	6	9	5	4
3	1	3	4	9	5	6
4	1	3	4	5	9	6
5	1	3	4	5	6	9

3.1.3. Schreiben Sie den Algorithmus **Selection Sort** in Pseudocode auf, der ein Array A[1..n] der Länge n aufsteigend sortiert. Es sollen dabei keine Unterfunktionen aufgerufen werden. Wie viele **Vergleiche** führt der Algorithmus aus, wenn das Array

a) die Länge n=1 hat 0
b) die Länge n=2 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält 1
c) die Länge n=3 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält 3
d) die Länge n=4 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält 6
e) die Länge n=3 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält 3
f) die Länge n=4 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält 6

```
func selection_sort(A[1..n])
  for i = 1 to n-1
    var j = i
    for k = i +1 to n
      if A[k] < A[j]
        j=k
    var t = A[i]
    A[i] = A[j]
    A[j] = t
```

3.2. Insertion Sort

3.2.1. Sortieren Sie die folgenden Zahlen mit **Insertion Sort** in **absteigendender** Reihenfolge.
Die Sortierung soll in n-1 Runden durchgeführt werden. Vervollständigen Sie die Tabelle, indem Sie die Array Elemente nach jeder Runde eintragen.

	A[1]		Array A			A[6]
Runde	3	9	6	1	5	4
1	9	3	6	1	5	4
2	9	6	3	1	5	4
3	9	6	3	1	5	4
4	9	6	5	3	1	4
5	9	6	5	4	3	1

3.2.2. Sortieren Sie die folgenden Zahlen mit **Insertion Sort** in **aufsteigendender** Reihenfolge.
Die Sortierung soll in n-1 Runden durchgeführt werden. Vervollständigen Sie die Tabelle, indem Sie die Array Elemente nach jeder Runde eintragen.

	A[1]		Array A			A[6]
Runde	3	9	6	1	5	4
1	3	6	9	1	5	4
2	1	3	6	9	5	4
3	1	3	5	6	9	4
4	1	3	4	5	6	9
5	1	3	4	5	6	9

3.2.3. Schreiben Sie den Algorithmus **Insertion Sort** in Pseudocode auf, der ein Array A[1..n] der Länge n aufsteigend sortiert. Es sollen dabei keine Unterfunktionen aufgerufen werden. Wie viele **Vergleiche** führt der Algorithmus aus, wenn das Array

a) die Länge n=1 hat 0
b) die Länge n=2 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält 1
c) die Länge n=3 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält 2
d) die Länge n=4 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält 3
e) die Länge n=3 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält 4
f) die Länge n=4 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält 9

3.3. Sortieren

3.3.1. Wie viele elementare Schritte (1 Zeile == 1 Zeiteinheit) macht in Groß-$\mathcal{O}$ Notation:

a) Selection Sort im besten Fall (best-case) O(n^2)
b) Selection Sort im schlechtesten Fall (worst-case) O(n^2)
c) Insertion Sort im besten Fall (best-case) O(n)
d) Insertion Sort im schlechtesten Fall (worst-case) O(n^2)

3.3.2. Nennen Sie je ein Beispiel für ein Sortierverfahren aus der Vorlesung, bei der die Anzahl der elementaren Schritte

a) im schlechtesten Fall quadratisch $\mathcal{O}(n^2)$ ist Selection Sort / Insertion sort
b) im schlechtesten Fall $\mathcal{O}(n \cdot \log(n))$ ist Merge Sort
c) von den Eingabedaten unabhängig ist Selection Sort / Merge Sort
d) von den Eingabedaten abhängig ist Insertion Sort
e) im besten Fall linear $\mathcal{O}(n)$ ist Insertion Sort
f) im besten Fall quadratisch $\mathcal{O}(n^2)$ ist Selection Sort

3.3.3. Betrachten Sie den folgenden Algorithmus:

```
maybe4( A[1..4] ):
        if A[1] > A[2]
                tausche( A, 1, 2 )
        if A[3] > A[4]
                tausche( A, 3, 4 )
        if A[1] > A[4]
                tausche( A, 1, 4 )
        if A[2] > A[3]
                tausche( A, 2, 3 )
```

a) Nennen Sie eine Eingabe, die dieser Algorithmus korrekt in aufsteigender Reihenfolge sortiert.
P= [2,1,4,3]

b) Sortiert dieser Algorithmus *jede Eingabe* korrekt in aufsteigender Reihenfolge? Nein
Begründen Sie kurz ihre Entscheidung! der Algorithmus sortiert manche Eingaben wie P= [2,4,3,1] nicht richtig

3.3.4. Schreiben Sie einen Algorithmus **twosort(A[1..n])** in Pseudocode, der jedes Array korrekt aufsteigend sortiert, das nur die Zahlen 0 und 1 enthält und der im schlechtesten Fall lineare Laufzeit $\mathcal{O}(n)$ hat. Sie dürfen nur $\mathcal{O}(1)$ zusätzlichen Speicher benutzen.

3.3.5.* Für die vorangehende Aufgabe gibt es mindestens zwei grundsätzlich verschiedene Lösungsansätze. Lösen Sie die Aufgabe erneut mit einem anderen Lösungsansatz.

3.3.6. Schreiben Sie einen Algorithmus **sorted(A[1..n])** in Pseudocode, der prüft ob ein beliebiges Array A, dessen Elemente ganze Zahlen sind, bei der Eingabe bereits aufsteigend sortiert ist (Ergebnis: 1) oder nicht (Ergebnis: 0). Was ist die Laufzeit des Algorithmus im worst-case in Groß-$\mathcal{O}$ Notation? O(n)

3.4. Suchen

3.4.1 Ein Array A[1..n] der Länge n=15 enthalte die folgenden Zahlen:

Index i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A[i]	12	28	96	13	43	29	54	68	93	17	23	39	42	88	72

a) Wie viele Vergleiche benötigt man für eine lineare Suche von links nach rechts nach 29? 6

b) Wie viele Vergleiche benötigt man für eine lineare Suche von links nach rechts nach 88? 14

c) Sortieren Sie das Array in aufsteigender Reihenfolge, tragen Sie das Ergebnis ein:

Index i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A[i]	12	13	17	23	28	29	39	42	43	54	68	72	88	93	96

d) Tragen Sie den Verlauf einer binären Suche nach der Zahl z=29 in folgende Tabelle ein, notieren Sie dabei für jeden Suchschritt die untere Grenze a, die obere Grenze b und die Mitte m. Unter Vergleich tragen Sie bitte die beiden zu vergleichenden Werte und <, = oder > ein, je nachdem wie der Vergleich in diesem Schritt ausgeht:

Schritt	a	b	m	Vergleich (<, = oder >)
1	1	15	8	42 > 29
2	1	7	4	24<29
3	5	7	6	29=29
4				
5				
6				

e) Tragen Sie den Verlauf einer binären Suche nach der Zahl z=88 in folgende Tabelle ein, notieren Sie dabei für jeden Suchschritt die untere Grenze a, die obere Grenze b und die Mitte m. Unter Vergleich tragen Sie bitte die beiden zu vergleichenden Werte und <, = oder > ein, je nachdem wie der Vergleich in diesem Schritt ausgeht:

Schritt	a	b	M	Vergleich (<, = oder >)
1	1	15	8	42<88
2	9	15	12	72<88
3	13	15	14	93>88
4	13	13	13	88=88

3.5.1. Anzahl der Nullen in einem sortierten Array

a) Gesucht ist ein Algorithmus **nullen(A(1..n])** in Pseudocode der die Anzahl der in A enthaltenen Nullen berechnet. Dabei ist A ein bereits *aufsteigend sortiertes* Array natürlicher Zahlen ≥0. Beispiel: Für A = [0|0|2|7] ist das Ergebnis 2 und für A=[1|2|3|5|7] ist das Ergebnis 0. Die Laufzeit des Algorithmus soll deutlich besser als im allgemeinen Fall eines unsortierten Arrays sein. Welche Laufzeit hat der von Ihnen entwickelte Algorithmus größenordnungsmäßig?

b)* Erweitern Sie den Algorithmus so, dass er eine zusätzliche Zahl x als Eingabe hat und die Anzahl der Vorkommen von x im aufsteigend sortierten Array A ermittelt.

3.5.2.** Gleichzeitiges Suchen von Minimum und Maximum in einem Array

Gegeben ist ein Array A[1..n] ganzer Zahlen. Gesucht sind das Minimum und das Maximum unter den Elementen von A. Vereinfachend sei n eine Zweierpotenz, also $n = 2^k$ für eine natürliche Zahl k. Der einfache Ansatz, die Algorithmen min(A[1..n]) und max(A[1..n]) hintereinander Aufzurufen funktioniert und benötigt insgesamt $2\cdot(n\text{-}1)$ Vergleiche. Finden Sie einen Algorithmus, der deutlich weniger Vergleiche benötigt! Hinweis: Eine optimale Lösung benötigt nur ca. $1{,}5\cdot n$ Vergleiche.

3.5.3.** Größte durch 3 teilbare Zahl aus einem Array von Ziffern

Eingabe ist ein Array A[1..n]. Jedes Array Element ist eine der Ziffern 0, 1, 2 ... 9. Gesucht ist die größte durch 3 teilbare Zahl, die aus den Ziffern im Array A gebildet werden kann. Der Ziffernvorrat im Array A darf dabei in beliebiger Reihenfolge genutzt werden, nicht alle Ziffern müssen verwendet werden. Beispiel: A = {7, 1, 6, 8, 0, 6} Ausgabe: 87660
Wie groß wäre die Laufzeit, wenn Sie eine Lösung durch systematisches Probieren (exhaustive search) ermitteln würden? Finden Sie eine effizientere Lösung!
<u>Hinweise:</u> 1. Benutzen Sie die Teilbarkeitsregel für die Teilbarkeit durch 3.
2. Welche Rolle spielt die Reihenfolge der Ziffern einer Zahl für deren Teilbarkeit durch 3?
3. Benutzen Sie Sortieren und drei Queues um eine effiziente Lösung zu erhalten.

Quelle mit Lösung: http://www.geeksforgeeks.org/find-the-largest-number-multiple-of-3/