Aufgabe 1

18.5.2014 - Okan Biskin & Tobias Lohse
Tutorium bei Can Göktas (Do. 14-16 Uhr)

AUFGABENTEIL A)

```
def is_sorted(L):
	'''Kontrolliert ob eine Liste sortiert ist, für ansteigend sortierte Listen
		wird 1 zurüch gegeben, für absteigend sortierted Listen -1, für
		unsortierte 0 und für Listen mit gleichem Element None.'''
	sort = None
	i = 0
	while sort == None and i+1 < len(L):
		if L[i] < L[i+1]:
			#Liste ist nicht absteigend
			sort = 1
		elif L[i] > L[i+1]:
			#Liste ist nicht aufsteigend
			sort = -1
		i += 1
	while sort == 1 and i+1 < len(L):
		if L[i] > L[i+1]:
			#Liste ist weder auf- noch absteigend
			sort = 0
		i += 1
	while sort == -1 and i+1 < len(L):
		if L[i] < L[i+1]:
			#Liste ist weder ab- noch aufsteigend
			sort = 0
		i += 1
	return sort
```

AUFGABENTEIL B)

```
from random import randint
def generate_random_list(a=0,b=99,n=50):
	'''Generiert eine Liste der Länge n mit ganzen Zufallszahlen
		im Bereich zwischen a und b.'''
	return [ randint(a,b) for i in range(n) ]
```

AUFGABENTEIL C)

```
def bubble_sort(L,lo,up):
	'''Sortiert den abschnitt zwischen lo und up der gegebene Liste
		L mit Bubble-Sort.'''
	remain_up = up-1
	done = False
	while not done:
		done = True
		for i in range(lo,remain_up):
			if L[i] > L[i+1]:
				done = False
				L[i], L[i+1] = L[i+1], L[i]
		remain_up -= 1
```

```
def merge(L,H,lo,up,mid):
	'''Merged die beiden Listenabschnitte von L zwischen lo-mid
		und mid-up in der Hilfsliste H.'''
	i = lo
	j = mid
	for k in range(lo,up):
		if i < mid and j < up:
			if L[i] <= L[j]:
				H[k] = L[i]
				i += 1
			else:
				H[k] = L[j]
				j += 1
		elif i < mid:
			H[k] = L[i]
			i += 1
		elif j < up:
			H[k] = L[j]
			j += 1

def merge_sort(L,H,lo,up,length,threshold=9):
	'''Sortiert die Liste L zwischen up und lo. Dabei wird die Hilfsliste H
		verwendet und die Liste wird an der Stelle lo+length geteilt.
		Up threshold wird Bubble-Sort verwendet.'''
	if length <= threshold:
		bubble_sort(L,lo,up)
	else:
		mid = lo+length
		L = merge_sort(L,H,lo,mid,length//2,threshold)
		L = merge_sort(L,H,mid,up,length//2,threshold)
		merge(L,H,lo,up,mid)
		L = H[:] #L updaten
	return L

def merge_sort_init(L,threshold=9):
	'''Sortiert die Liste L mit Mergesort. Optional, kann angegeben
		werden, ab welchem threshold Bubble-Sort benutzt werden soll.'''
	if len(L) < threshold:
		bubble_sort(L,0,len(L))
	else:
		H = L[:]
		length = len(L)//2 #gibt die Länge der zu mergenden Listen an
		L = merge_sort(L,H,0,len(L),length,threshold)
	return L
```

AUFGABENTEIL D)

```
def merge_sort(L,threshold=9):
	'''Sortiert die Liste L mit Mergesort. Optional, kann angegeben
		werden, ab welchem threshold Bubble-Sort benutzt werden soll.'''
	len_L = len(L)
	size = min(threshold,len_L)
	#sortiere die Teillisten
```

```
        for lo in [i*size for i in range(round(len_L/size+.5))]:
                up = min(lo+size,len_L)
                bubble_sort(L,lo,up)
        #merge die Teillisten
        while size < len_L:
                H = L[:] #H updaten
                for lo in [i*2*size for i in range(round(len_L/(2*size)+.5))]:
                        mid = lo+size
                        up = min(lo+2*size,len_L)
                        merge(L,H,lo,up,mid)
                        L = H[:] #L updaten
                size *= 2
        return L
```

Aufgabe 2

```
from textwrap import fill as wrap_text
def bubble_sort(L):
        '''Sortiert die gegebene Liste L mit Bubble-Sort.'''
        remain = len(L)-1
        done = False
        swap_L = [ [] for i in range(len(L)) ]
        while not done:
                done = True
                for i in range(remain):
                        if L[i] > L[i+1]:
                                done = False
                                # print("Swapping %s_%s and %s_%s" % (L[i],i,L[i+1],i+1))
                                L[i], L[i+1] = L[i+1], L[i]
                                swap_L[i] += [1]
                                swap_L[i+1] += [-1]
                                swap_L[i], swap_L[i+1] = swap_L[i+1], swap_L[i]
                remain -= 1
        total_swaps = sum([ len(swaps) for swaps in swap_L ])
        wrong_direction_swaps = sum([ abs( abs(sum(swaps))-len(swaps) )
                                                                for swaps in swap_L ])//2
        print(wrap_text( "Von %d Vertauschungen, gingen %d in die falsche Richtung und
nochmal soviele wurden gebraucht um diese Rückgängig zu machen. Insgesamt waren also %d
%% der Vertauschungen unnötig" %
(total_swaps,wrong_direction_swaps,round(2*wrong_direction_swaps/
total_swaps*100)) ),"\n")
        return 2*wrong_direction_swaps/total_swaps*100

# Test
from random import randint
for i in range(10):
        L = [ randint(0,99) for i in range(100) ]
        unuseful_swap = []
        unuseful_swap += [bubble_sort(L)]
        unuseful_swap_average = sum(unuseful_swap)/len(unuseful_swap)
```

```
	print("Durchschnittlich waren %.2f%% der Vertauschungen unnötig."
                                                    % unuseful_swap_average)
```

Aufgabe 3

AUFGABENTEIL A)

```
def counting_sort(L):
	'''Sortiert die gegebene Liste von ganzen Zahlen mit in-place Counting-Sort.'''
	C = [0 for i in range(max(L)+1)]
	for i in range(len(L)):
		C[L[i]] += 1
	l_i = 0
	for c_i in range(len(C)):
		while C[c_i] != 0:
			L[l_i] = c_i
			l_i += 1
			C[c_i] -= 1

# Tests
from random import randint
for i in range(10):
	L = [ randint(0,9) for i in range(20) ]
	counting_sort(L)
	print(L)
print()
```

AUFGABENTEIL C)

```
from textwrap import fill as wrap_text
print(wrap_text("Der Algorythmus besteht aus zwei Teilen. Zunächst werden alle Elemente
von L aufgerufen und in C summiert. Dabei ist die Laufzeit T1=c1*n wobei n=len(L).
Danach werden alle addierten einsen aus C n=sum(C)=len(L) wieder in L abgespeichert
T2=c2*n. Allerdings muss in diesem Schritt auch noch mindestens m=len(C)=max(L) mal
verglichen werden, ob das Element von C bereits 0 ist. Damit ergibt sich T3=c3*m.
Insgesamt haben wir damit eine Laufzeit von T=(c1+c2)*n+c3*m=O(n+m)."))
print()
print(wrap_text("Der Speicheraufwand ist durch die Länge von C, also den Betrag des
größten Elements von L, m=len(C)=max(L) gegeben S(m)=c1*m=O(m)."))
print()

# Aufgabenteil b)
print(wrap_text("Der in-place Counting-Sort Algorythmus ist nicht zum sortieren nach
Geburtstagen geeignet. Es treten zwei Probleme auf:"))
print(wrap_text("1. kann der Algorythmus nur Listen aus ganzen Zahlen sortieren, da die
gegebene Liste mit den in C gespeicherten Plätzen überschrieben wird. C speichert aber
nur Positionen von ganzen Zahlen und keine Daten. Dies lässt sich nur lösen, wenn die
in-place Eigenschaft aufgegeben wird."))
print(wrap_text("2. Außerdem ist der Algorythmus noch nicht einmal zur Sortierung von
Geburtsdaten an sich geeignet. Geburtstage sind 8-stellige Zahlen (YYYMMDD), somit
bräuchte man eine über 20-millionen-stellige Hilfsliste C und selbst, wenn man den
```

Algorythmus so verändern würde, dass C nicht bei 0 losläuft sondern einen Startwert akzeptiert, so hätte man immernoch 1 Million Stellen in C für die letzten hundert Jahre und proportionale Laufzeit und Speicherverbrauch. Die Lösung wäre eine Radix-Sort-Variante, welche nacheinander Tage, Monate und Jahre sortiert. Diese lässt sich mit diesem Algorythmus aber wegen der fehlenden Stabilität nich implementieren. “))

Aufgabe 4

AUFGABENTAIL A)

```
def heapify(H,i):
	'''Stellt die Heapbedingung des Heaps H an der Stelle i her.'''
	le, ri = i*2, i*2+1
	if le <= H[0] and H[le][:2]>H[i][:2]: maxi = le
	elif ri <= H[0] and H[ri][:2]>H[i][:2]: maxi = ri
	else: maxi = i
	if maxi != i:
		H[i], H[maxi] = H[maxi], H[i]
		heapify(H,maxi)

def build_heap(H):
	'''Wandelt eine Liste L in einen Heap um, welcher zurückgegeben wird.'''
	H[0] = len(H)-1
	for i in range(H[0]//2,0,-1):
		heapify(H,i)

def heapsort(H):
	'''Sortiert die Liste L mit Heapsort und gibt die Liste zurück.'''
	build_heap(H)
	for i in range(H[0],1,-1):
		H[i], H[1] = H[1], H[i]
		H[0] -= 1
		heapify(H,1)
	H[0] -= 1
	return H[1:]

print("Heapsort ist nicht stabil, wie man an folgendem Beispiel sehen kann.")
H = ["",(2,2,"a"),(2,2,"b"),(1,1,"a")]
print(" H: ",H[1:])
print("Der Heap wird nach den ersten beiden Tupelelementen sortiert. ")
print(" H: ",heapsort(H))
print("Die Ursprüngliche Reihenfolge von (2,2,a) und (2,2,b) wurde vertauscht.")
print()
```

AUFGABENTAIL B)

```
from random import randint
from time import time
def next_message():
	return (randint(1,50),time(),"Message:%06d" % randint(0,999999))
```

```
	# yield is not possible, because it produces a TypeError with the message:
	'generator' object is not subscriptable.
	We would have to create a class to really use yield in a meaningfull way here.
```

AUFGABENTAIL C)

```
def insert(message_queue,message):
	'''Eine neue Nachricht wird in die Prioritätswarteschlange eingefügt.'''
	message_queue[0] += 1 #Heaplgröße erhöhen
	message_queue.insert(1,message) #neue Wurzel hinzufügen
	heapify(message_queue,1) #Heapeigenschaft wiederherstellen
	return message

def is_empty(message_queue):
	'''gibt einen Wahrheitswert als Rückgabewert zurück, je nachdem, ob die
Prioritätswarteschlange leer ist oder nicht.'''
	return (True if message_queue[0] == 0 else False)

def remove_message(message_queue):
	'''Die Nachricht mit der höchsten Priorität und die, die zeitlich zu erst
produziert worden ist, wird aus der Prioritätswarteschlange entfernt und als Ergebnis
der Funktion zurückgegeben.'''
	if is_empty(message_queue):
		return None
	else:
		message_queue[0] -= 1 #Heaplgröße veringern
		message = message_queue.pop(1) #Wurzel entfernen
		heapify(message_queue,1) #Heapeigenschaft wiederherstellen)
		return message

def sort_messages(message_queue):
	'''Erstellt einen Heap aus der gegebenen message_queue.'''
	build_heap(message_queue)
```

AUFGABENTAIL D)

```
def simulate_message_traffic(N=50):
	'''Simuliert zufälligen Nachrichten Austausch.'''
	message_queue = [0]
	for i in range(N):
		if randint(0,1) == 1:
			insert(message_queue,next_message())
		else:
			print(remove_message(message_queue))

simulate_message_traffic()
```