**1MPR06\_Simona\_Bļinova sb24037**

Visos uzdevumos lietoju savu moduli ar funkcijām. (programmu mapē *funkcijas.py*)

**1.uzdevums**

Programma, kas veic masīva kārtošanu dilstoša secībā ar trim metodēm.

**Kods:**

Funkcijas no *funkcijas.py*:

# Funkcija pārbaudei, lai skaitlis būtu naturāls skaitlis

def naturals\_skaitlis(a):

# meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)

meginajumi = 1

while meginajumi <= 3:

try:

a = int(a)

if a > 0:

return int(a)

else:

raise Exception

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet naturālo skaitli vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

# Masīva izveides funkcija

def masiva\_izveide(n):

return numpy.arange(n)

*1MPR06\_1\_Simona\_Blinova.py*:

import funkcijas

import random

import copy

import numpy

''' Funkciju bloks '''

def aizpildit\_masivu(a, elementu\_skaits):

for i in range(elementu\_skaits, 2, -1):

b = random.randint(1, i-1)

c = a[i]

a[i] = a[b]

a[b] = c

return a

def izvade(a):

garums = len(a)

virkne = ''

for i in range(1, garums):

if i == garums - 1:

virkne += str(a[i])

else:

virkne += str(a[i]) + ', '

return f'{a[0]} salīdzināšanas - {virkne}'

##### Ātrā jeb Hoara kārtošana #####

def hoara(a, sakuma\_vertiba, beigu\_vertiba):

if sakuma\_vertiba < beigu\_vertiba:

i = sakuma\_vertiba

j = beigu\_vertiba

solis = -1

lv = True

while i != j:

a[0] += 1

if lv == (a[i] < a[j]):

x = a[i]

a[i] = a[j]

a[j] = x

x = i

i = j

j = x

lv = not lv

solis = -solis

j += solis

hoara(a, sakuma\_vertiba, i - 1)

hoara(a, i + 1, beigu\_vertiba)

return a

##### Saliešanas metode #####

def saliesana(a, sakuma\_vertiba, beigu\_vertiba):

garums = len(a)

b = funkcijas.masiva\_izveide(garums)

if sakuma\_vertiba < beigu\_vertiba:

vidus\_vertiba = (sakuma\_vertiba + beigu\_vertiba) // 2

saliesana(a, sakuma\_vertiba, vidus\_vertiba)

saliesana(a, vidus\_vertiba + 1, beigu\_vertiba)

for i in range(sakuma\_vertiba, beigu\_vertiba + 1):

b[i] = a[i]

i = sakuma\_vertiba

j = vidus\_vertiba + 1

k = sakuma\_vertiba

while i <= vidus\_vertiba and j <= beigu\_vertiba:

a[0] += 1

if b[i] > b[j]:

a[k] = b[i]

i += 1

else:

a[k] = b[j]

j += 1

k += 1

if j > beigu\_vertiba:

while i <= vidus\_vertiba:

a[k] = b[i]

i += 1

k += 1

return a

##### Timsort #####

def iev\_met(a, sakums, beigas):

for i in range(sakums+1, beigas+1):

elem = a[i]

j = i-1

while j >= sakums and a[j] < elem:

a[0] += 1

a[j+1] = a[j]

j = j - 1

a[j+1] = elem

def apvienosana(a, sakums, vidus, beigas):

g1 = vidus - sakums + 1

g2 = beigas - vidus

viens\_a = numpy.copy(a[sakums:sakums+g1])

otrs\_a = numpy.copy(a[vidus+1:vidus+1+g2])

i = 0

j = 0

k = sakums

while i < g1 and j < g2:

a[0] += 1

if viens\_a[i] >= otrs\_a[j]:

a[k] = viens\_a[i]

i += 1

else:

a[k] = otrs\_a[j]

j += 1

k += 1

while i < g1:

a[k] = viens\_a[i]

i += 1

k += 1

while j < g2:

a[k] = otrs\_a[j]

j += 1

k += 1

def tim\_sort(a):

INTERVALA\_GARUMS = 2

n = len(a)

#Atsevišķu intervālu sakārtošana

for i in range(1, n, INTERVALA\_GARUMS):

iev\_met(a, i, min(i+INTERVALA\_GARUMS - 1, n-1))

izmers = INTERVALA\_GARUMS

while izmers < n:

for sakums in range(1, n, 2\*izmers):

vidus = min(n-1, sakums+izmers-1)

beigas = min(n-1, sakums + 2\*izmers-1)

if vidus < beigas:

apvienosana(a, sakums, vidus, beigas)

izmers = izmers \* 2

return a

''' Masīva izveides un aizpildes bloks '''

n = input('Ievadiet masīva izmēru --> ')

n = funkcijas.naturals\_skaitlis(n)

masivs = funkcijas.masiva\_izveide(n+1)

masivs = aizpildit\_masivu(masivs, n)

print(masivs)

'''Masīva kopijas kārtošana trīs veidos'''

hoara\_masivs = hoara(copy.deepcopy(masivs), 1, len(masivs)-1)

saliesanas\_masivs = saliesana(copy.deepcopy(masivs), 1, len(masivs)-1)

timsort\_masivs = tim\_sort(copy.deepcopy(masivs))

''' Rezultātu izvade '''

print(f'Ātrā jeb Hoara kārtošanas metode: {izvade(hoara\_masivs)}')

print(f'Saliešanas metode: {izvade(saliesanas\_masivs)}')

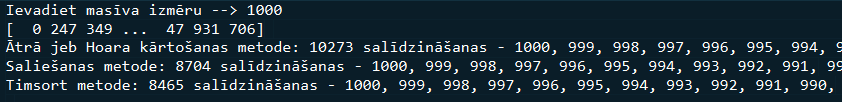
print(f'Timsort metode: {izvade(timsort\_masivs)}')

Testa piemērs(1)

A screenshot of a computer

AI-generated content may be incorrect.

Testa piemērs(2)



Testa piemērs(3)

A computer screen with numbers

AI-generated content may be incorrect.

**2.uzdevums**

Programma, kas veic trīs dažāda izmēra masīvu apvienošanu.

**Kods:**

Funkcijas no *funkcijas.py*:

# Funkcija pārbaudei, lai skaitlis būtu naturāls skaitlis

def naturals\_skaitlis(a):

# meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)

meginajumi = 1

while meginajumi <= 3:

try:

a = int(a)

if a > 0:

return int(a)

else:

raise Exception

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet naturālo skaitli vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

# Masīva izveides funkcija

def masiva\_izveide(n):

return numpy.arange(n)

# Funkcija pārbaudei, lai skaitlis būtu reāls skaitlis

def reals\_skaitlis(a):

# meģinajumu skaitītājs (3 meģinājumi pareizi ievadīt)

meginajumi = 1

while meginajumi <= 3:

try:

a = float(a)

return float(a)

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet reālo skaitli vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

*1MPR06\_2\_Simona\_Blinova.py*:

import funkcijas

import math

def izveidot\_masivu():

n = input('Ievadiet masīva izmēru --> ')

n = funkcijas.naturals\_skaitlis(n)

a = funkcijas.masiva\_izveide(n)

for i in range(n):

skaitlis = input('Ievadiet skaitli masīvā --> ')

skaitlis = funkcijas.reals\_skaitlis(skaitlis)

a[i] = skaitlis

return a

def shell\_metode\_augosa(a):

garums = len(a)

solis = (3\*\*math.floor(math.log(2\*garums+1, 3))-1)//2

while solis >= 1:

for i in range(0, solis):

for j in range(solis+i, garums, solis):

if a[j-solis] > a[j]:

b = a[j]

k = j

while a[k-solis] > b:

a[k] = a[k-solis]

k -= solis

if k == i:

break

a[k] = b

solis = (solis - 1) // 3

return a

def shell\_metode\_dilstosa(a):

garums = len(a)

solis = (3\*\*math.floor(math.log(2\*garums+1, 3))-1)//2

while solis >= 1:

for i in range(0, solis):

for j in range(solis+i, garums, solis):

if a[j-solis] < a[j]:

b = a[j]

k = j

while a[k-solis] < b:

a[k] = a[k-solis]

k -= solis

if k == i:

break

a[k] = b

solis = (solis - 1) // 3

return a

def apvienosana(a, b):

garums\_a = len(a)

garums\_b = len(b)

garums\_c = garums\_a + garums\_b

c = funkcijas.masiva\_izveide(garums\_c)

ia = indeksa\_noteiksana(a)

ib = indeksa\_noteiksana(b)

ic = 0

if ia != 0:

a = shell\_metode\_augosa(a)

ia = 0

if ib != 0:

b = shell\_metode\_augosa(b)

ib = 0

while ia < garums\_a and ib < garums\_b:

if a[ia] < b[ib]:

c[ic] = a[ia]

ia += 1

else:

c[ic] = b[ib]

ib += 1

ic += 1

if ia < garums\_a:

for i in range(ia, garums\_a):

c[ic] = a[i]

ic += 1

else:

for i in range(ib, garums\_b):

c[ic] = b[i]

ic += 1

return c

def indeksa\_noteiksana(a):

garums = len(a)

if a[0] < a[garums-1]:

i = 0

else:

i = garums-1

return i

masivs1 = izveidot\_masivu()

print(masivs1)

masivs2 = izveidot\_masivu()

print(masivs2)

masivs3 = izveidot\_masivu()

print(masivs3)

masivs1 = shell\_metode\_augosa(masivs1)

masivs2 = shell\_metode\_dilstosa(masivs2)

masivs3 = shell\_metode\_augosa(masivs3)

print(masivs1, masivs2, masivs3)

masivs12 = apvienosana(masivs1, masivs2)

masivs = apvienosana(masivs12, masivs3)

print(masivs)

Testa piemērs(1)

A screen shot of a computer program

AI-generated content may be incorrect.

Testa piemērs(2)

A screenshot of a computer program

AI-generated content may be incorrect.

Testa piemērs(3)

A screenshot of a computer program

AI-generated content may be incorrect.

**PU**

Programma, kas veic trīs dažāda izmēra masīvu apvienošanu.

**Kods:**

Funkcijas no *funkcijas.py*:

# Funkcija pārbaudei, lai skaitlis būtu naturāls skaitlis

def naturals\_skaitlis(a):

# meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)

meginajumi = 1

while meginajumi <= 3:

try:

a = int(a)

if a > 0:

return int(a)

else:

raise Exception

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet naturālo skaitli vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

# Masīva izveides funkcija

def masiva\_izveide(n):

return numpy.arange(n)

# Funkcija pārbaudei, lai skaitlis būtu reāls skaitlis

def reals\_skaitlis(a):

# meģinajumu skaitītājs (3 meģinājumi pareizi ievadīt)

meginajumi = 1

while meginajumi <= 3:

try:

a = float(a)

return float(a)

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet reālo skaitli vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

*1MPR06\_PU\_Simona\_Blinova.py*:

import funkcijas

import math

import random

def izveidot\_masivu():

n = input('Ievadiet masīva izmēru --> ')

n = funkcijas.naturals\_skaitlis(n)

a = funkcijas.masiva\_izveide(n)

for i in range(n):

skaitlis = input('Ievadiet skaitli masīvā --> ')

skaitlis = funkcijas.reals\_skaitlis(skaitlis)

a[i] = skaitlis

return a

def shell\_metode\_augosa(a):

garums = len(a)

solis = (3\*\*math.floor(math.log(2\*garums+1, 3))-1)//2

while solis >= 1:

for i in range(0, solis):

for j in range(solis+i, garums, solis):

if a[j-solis] > a[j]:

b = a[j]

k = j

while a[k-solis] > b:

a[k] = a[k-solis]

k -= solis

if k == i:

break

a[k] = b

solis = (solis - 1) // 3

return a

def shell\_metode\_dilstosa(a):

garums = len(a)

solis = (3\*\*math.floor(math.log(2\*garums+1, 3))-1)//2

while solis >= 1:

for i in range(0, solis):

for j in range(solis+i, garums, solis):

if a[j-solis] < a[j]:

b = a[j]

k = j

while a[k-solis] < b:

a[k] = a[k-solis]

k -= solis

if k == i:

break

a[k] = b

solis = (solis - 1) // 3

return a

def apvienosana(a, b):

garums\_a = len(a)

garums\_b = len(b)

garums\_c = garums\_a + garums\_b

c = funkcijas.masiva\_izveide(garums\_c)

# Nosaka indeksu (vai ir augoša vai dilstoša)

ia = indeksa\_noteiksana(a)

ib = indeksa\_noteiksana(b)

ic = 0

# pārveido masīvus par augošu, ja noteikts, ka ir dilstoša

if ia != 0:

a = shell\_metode\_augosa(a)

ia = 0

if ib != 0:

b = shell\_metode\_augosa(b)

ib = 0

while ia < garums\_a and ib < garums\_b:

if a[ia] < b[ib]:

c[ic] = a[ia]

ia += 1

else:

c[ic] = b[ib]

ib += 1

ic += 1

if ia < garums\_a:

for i in range(ia, garums\_a):

c[ic] = a[i]

ic += 1

else:

for i in range(ib, garums\_b):

c[ic] = b[i]

ic += 1

return c

def indeksa\_noteiksana(a):

garums = len(a)

if a[0] < a[garums-1]:

i = 0

else:

i = garums-1

return i

def random\_sort(a):

# Ja 0, tad dilstoša

# Ja 1, tad augoša

pazime = random.randint(0, 1)

if pazime == 0:

return shell\_metode\_dilstosa(a)

else:

return shell\_metode\_augosa(a)

masivs1 = izveidot\_masivu()

print(masivs1)

masivs2 = izveidot\_masivu()

print(masivs2)

masivs3 = izveidot\_masivu()

print(masivs3)

masivs1 = random\_sort(masivs1)

masivs2 = random\_sort(masivs2)

masivs3 = random\_sort(masivs3)

print(masivs1, masivs2, masivs3)

masivs12 = apvienosana(masivs1, masivs2)

masivs = apvienosana(masivs12, masivs3)

print(masivs)

Testa piemērs(1)

A screenshot of a computer program

AI-generated content may be incorrect.

Testa piemērs(2)

A screenshot of a computer program

AI-generated content may be incorrect.

Testa piemērs(3)

A screenshot of a computer program

AI-generated content may be incorrect.