POLITECHNIKA RZESZOWSKA

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

PROJEKT Z ALGORYTMÓW I STRUKTUR DANYCH
STUDENTA PIERWSZEGO ROKU STUDIÓW
KIERUNKU INŻYNIERIA I ANALIZA DANYCH

PORÓWNANIE ALGORYTMÓW SORTOWANIA

VITALII MORSKYI

Rzeszów 2021

SPIS TREŚCI

WSTĘP I OPIS ALGORYTMÓW

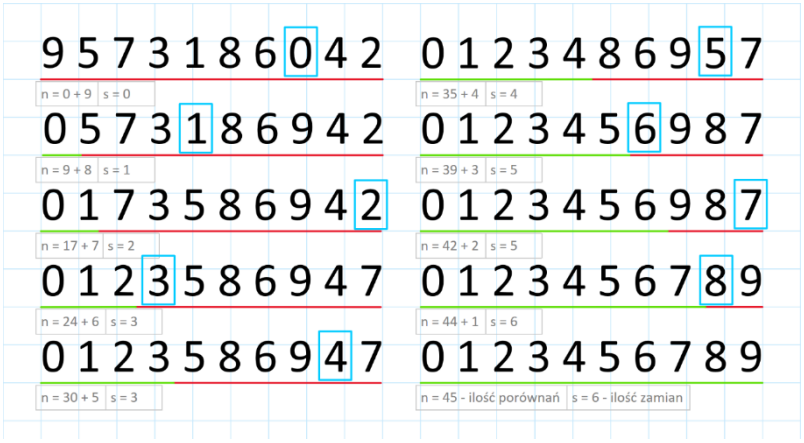
W tym rozdziale opisuję główne celi projektu oraz wykorzystane algorytmy dla ich implementacji.

Głównym zadaniem projektu było porównanie działania dwóch algorytmów sortowania, a mianowicie: sortowania przez wybieranie oraz sortowania kopcowego. Drugorzędnym zadaniem było napisanie algorytmów odczytu danych z plików i generowania danych testowych o różnej złożoności sortowania dla każdego algorytmu.

SORTOWANIE PRZEZ WYBIERANIE

Sortowanie przez wybieranie to niestabilny porównujący algorytm sortowania na miejscu. Ma złożoność czasową $\theta(n^2)$, co oznacza, że jest nieefektywnym w przypadku dużych list. Taki algorytm charakteryzuje się prostotą i ma przewagę wydajnościową nad bardziej złożonymi algorytmami w pewnych sytuacjach, zwłaszcza gdy pamięć pomocnicza jest ograniczona.

Algorytm dzieli listę wejściową na dwie partycji: posortowaną i nieposortowaną. Początkowo posortowana część jest pusta, a nieposortowana partycja zajmuje całą listę wejściową. Algorytm jest wykonywany poprzez znalezienie najmniejszego elementu w niesortowanej części, zamiane go na pierwszy nieposortowany element i przesunięcie podziału listy. Działanie algorytmu krok po kroku pokazano na rysunku (Rysunek 1).

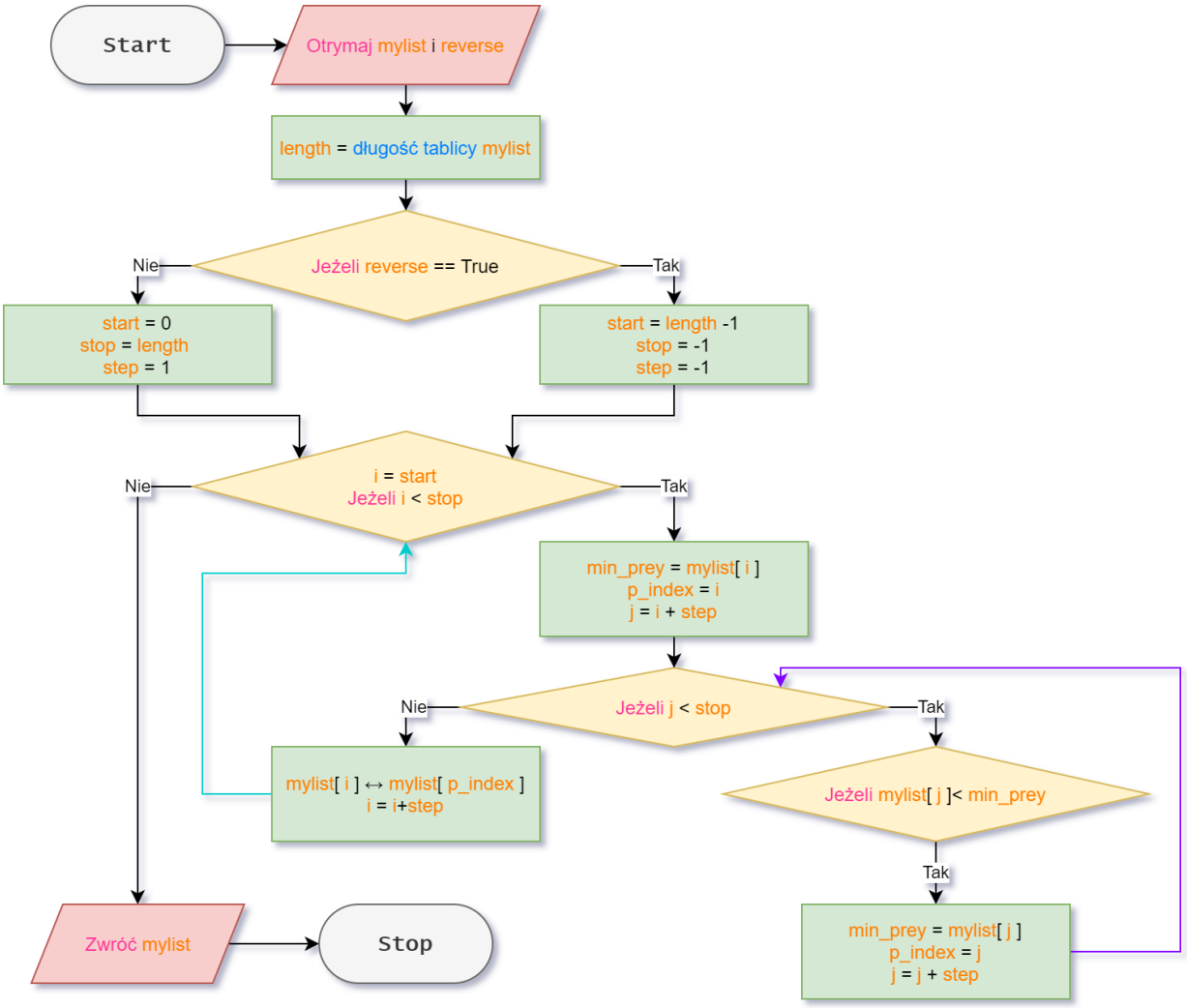


Rysunek 1 Szczegółowy schemat działania sortowania przez wybieranie z wypisywaniem ilości porównań i zamian po każdym przebiegu listy.
Źródło: opracowanie własne.

Pseudokod algorytmu sortowania przez wybieranie:

```
K01: Otrzymaj = mylist i reverse
K02: length = długośc tablicy mylist
K03: Jeżeli reverse == True wykonuj K04:K06
K04: |     start = 0
K05: |     stop = length - 1
K06: |     step = 1
K07: Inaczej wykonuj K08:K10
K08: |     start = length - 1
K09: |     stop = 0
K10: |     step = -1
K11: Dla i  = start, start + step, start + 2*step, start + 3*step, … , stop: wykonuj K12:K18
K12: |     min_prey = mylist[ i ]
K13: |     p_index = i
K14: |     Dla j = i + step, i + 2*step, i + 3*step, … , stop:  wykonuj K15:K17
K15: |     |       Jeżeli mylist [ i ] < min_prey wykonuj K16:K17
K16: |     |       |       min_prey = mylist[ j ]
K17: |     |       |       p_index = j
K18: |     mylist[ i ] ↔ mylist[ p_index ]
K19: Zwróć mylist
K20: Zakończ
```

Schemat blokowy tego algorytmu podano na schemacie (Schemat blokowy 1).

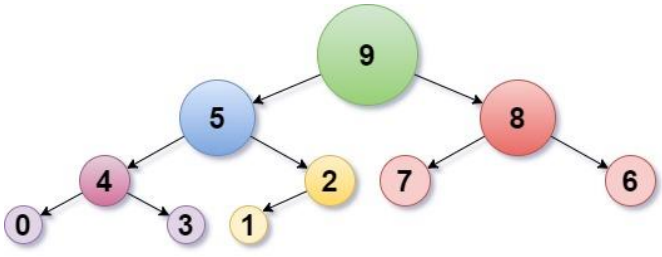


Schemat blokowy 1 Algorytm sortowania przez kopcowanie.
Źródło: opracowanie własne.

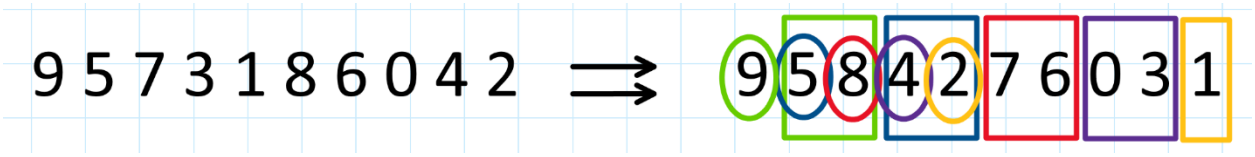
Największą zaletą takiego algorytmu jest minimalna możliwa liczba zamian elementów (w najgorszym przypadku $n - 1$). Mimo to efektywność czasowa sortowania według wyboru jest kwadratowa, więc istnieje wiele algorytmów, które mają mniejszą złożoność czasową.

SORTOWANIE PRZEZ KOPCOWANIE

Jednym z takich algorytmów jest sortowanie przez kopcowanie, które zostało wynalezione przez J. Williamsa w 1964 roku. Podobnie do algorytmu sortowania przez wybieranie, kopcowy algorytm sortuje na miejscu metodą porównań, nie jest stabilnym i analogicznie dzieli dane wejściowe na posortowaną i nieposortowaną partycji. Sortowanie kopcowe wybiera największy element z części niesortowanej i wstawiając go do posortowanej części, iteracyjnie zmniejsza nieposortowaną partycję. W przeciwieństwie do sortowania przez wybieranie, sortowanie kopcowe nie skanuje liniowo cały nieposortowany obszar. Ten algorytm obsługuje nieposortowaną część w postaci kopca (Schemat 1), aby szybko znaleźć największy element, który zawsze będzie znajdować się na

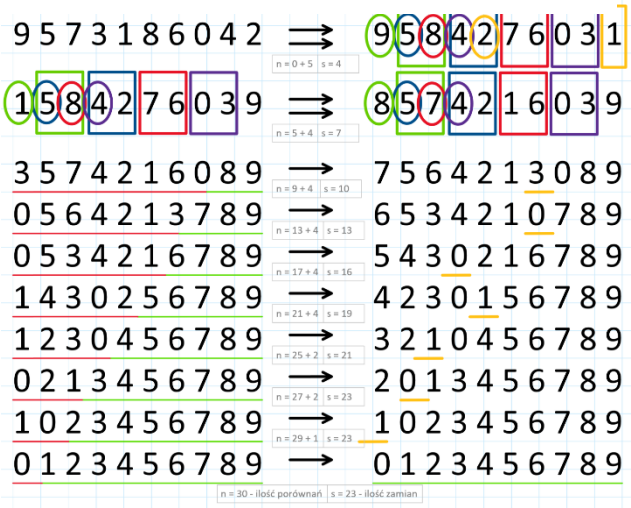


Schemat 1 Wygląd kopca jako struktury danych.
Źródło: opracowanie własne.



Rysunek 2 Kopiec w postaci tablicy.
Źródło: opracowanie własne.

górze. Ale dla roboty takiego algorytmu dane wejściowe muszą być ułożone w postaci kopca. Takie początkowe przygotowanie danych wejściowych wykonuje algorytm Heapify, sprawdzając czy potomki danego elementu są mniejsze od niego. Przykład kopca ze schematu (Schemat 1) w postaci tablicy podano na rysunku (Rysunek 2). Działanie algorytmu krok po kroku pokazano na rysunku (Rysunek 3).



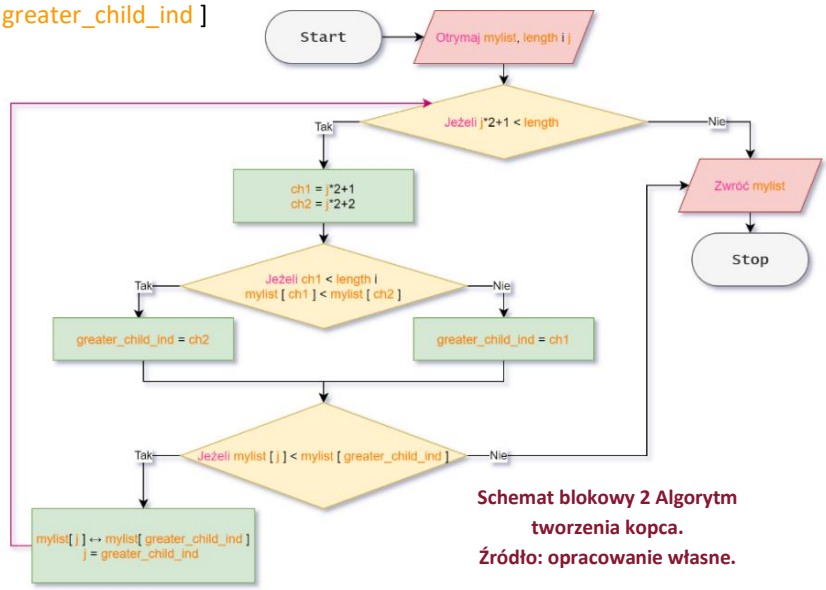
Rysunek 3 Szczegółowy schemat działania sortowania przez kopcowanie z wypisywaniem ilości porównań i zamian po każdym przebiegu listy.
Źródło: opracowanie własne.

Sortowanie kopcowe ma pesymistyczną złożoność czasową $\theta(n \log n)$, co jest nawet lepsze niż szybkie sortowanie ($\theta(n^2)$), chociaż w praktyce działa wolniej (oczekiwana złożoność obu algorytmów jest taka sama i wynosi $\theta(n \log n)$).

Schemat blokowy algorytmu tworzenia kopca podano na schemacie (Schemat blokowy 2).

Pseudokod algorytmu tworzenia kopca:

```
K01: Otrzymaj = mylist, length i  j
K02: Dopóki j*2+1 < length wykonuj K03:K13
K03: |     ch1 = j*2+1
K04: |     ch2 = j*2+2
K05: |     Jeżeli  ch1 < length i mylist [ ch1 ] < mylist [ ch2 ] wykonuj K06
K06: |     |      greater_child_ind = ch2
K07: |     Inaczej wykonuj K08
K08: |     |      greater_child_ind = ch2
K09: |     Jeżeli mylist [ j ] < mylist [ greater_child_ind ] wykonuj K10:K11
K10: |     |      mylist [ j ] ↔ mylist [ greater_child_ind ]
K11: |     |      j = greater_child_ind
K12: |     Inaczej wykonuj K13
K13: |     |      break
K14: Zwróć mylist
K15: Zakończ
```

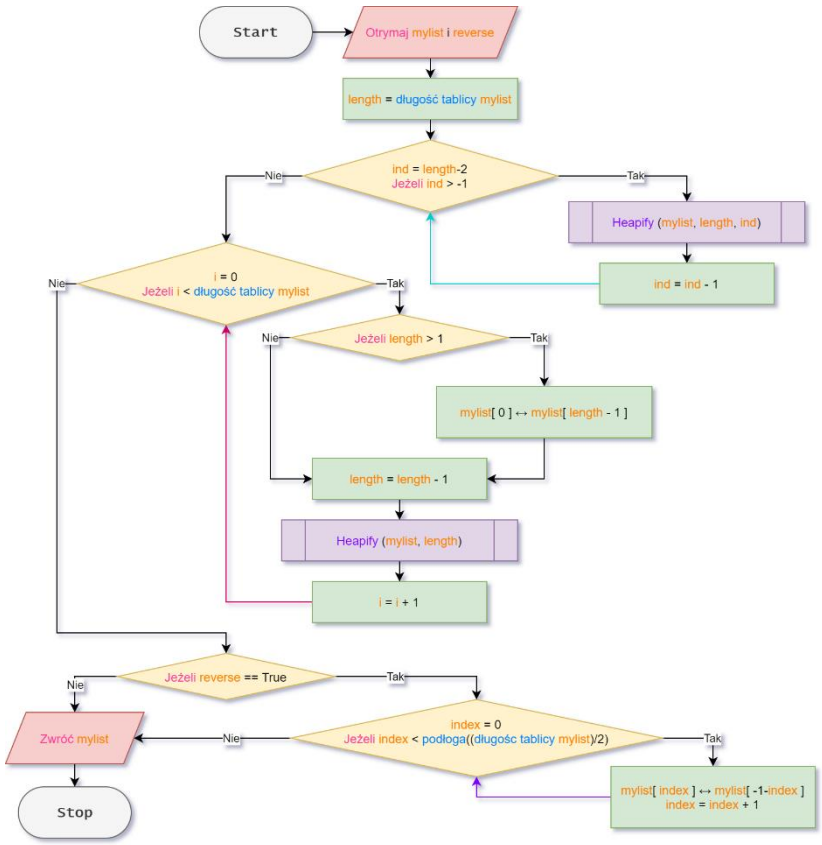


Schemat blokowy 2 Algorytm tworzenia kopca.
Źródło: opracowanie własne.

Schemat blokowy algorytmu sortowania przez kopcowanie podano na schemacie (Schemat blokowy 3).

Pseudokod algorytmu sortowania przez kopcowanie:

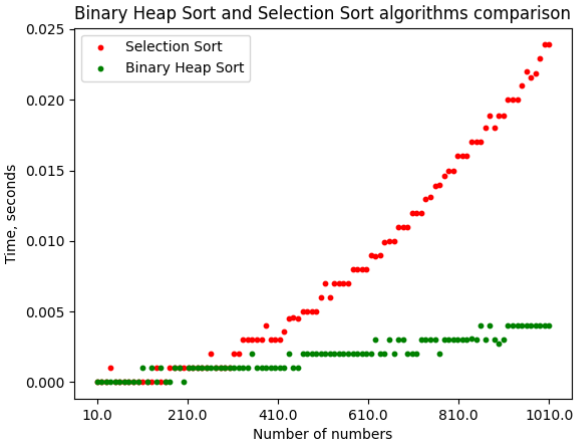
K01: Otrzymaj = mylist i reverse
K02: length = długośc tablicy mylist
K03: Dla ind = length - 2, length - 3, length - 4, … , 0: wykonuj K04
K04: | mylist = tworzenie kopca (mylist, length, ind)
K05: Dla i = 0, 1, 2, … , długośc tablicy mylist - 1: wykonuj K12:K18
K06: | Jeżeli length > 1 wykonuj K06
K07: | | mylist [0] ↔ mylist [length - 1]
K08: | length = length – 1
K09: | mylist = tworzenie kopca (mylist, length, 0)
K10: Jeżeli reverse == True wykonuj K04:K06
K11: | Dla index = 0, 1, 2, … , podłoga((długośc tablicy mylist)/2)- 1: wykonuj K12
K12: | | mylist [index] ↔ mylist [długośc tablicy mylist - index - 1]
K13: Zwróć mylist
K14: Zakończ



Schemat blokowy 3 Algorytm sortowania przez kopcowanie
Źródło: opracowanie własne.

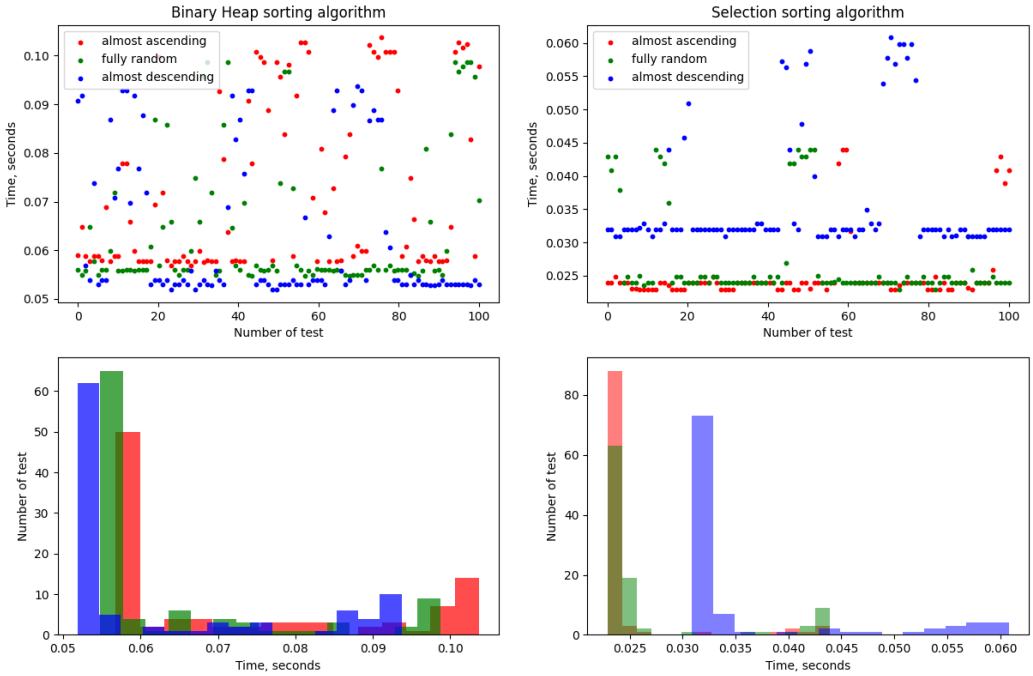
PORÓWNANIE ALGORYTMÓW

Jak już było omówiono w poprzednim rozdziale, złożoność obliczeniowa algorytmu sortowania przez wybór zawsze wynosi $\theta(n^2)$, a sortowania przez kopcowanie - $\theta(n \log n)$. Od razu widać przewagę drugiego algorytmu nad pierwszym, co potwierdzają dane o ilości porównań na rysunkach (Rysunek 1) i (Rysunek 3). Dla pełnego przekonania w poprawności takich danych, wykonano niektóre testy algorytmów i otrzymano wyniki opatrzone na wykresie (Wykres 1). Na wykresie bardzo dobrze widać, że sortowanie przez wybieranie już po czterystu liczbach zaczyna działać znacznie wolniej w porównaniu do sortowania przez kopcowanie.



Wykres 1 Zależność czasu od długości ciągu sortowanego dla obu algorytmów. Na czerwono zaznaczono sortowanie przez wybieranie, na zielono – sortowanie przez kopcowanie.
Źródło: opracowanie własne.

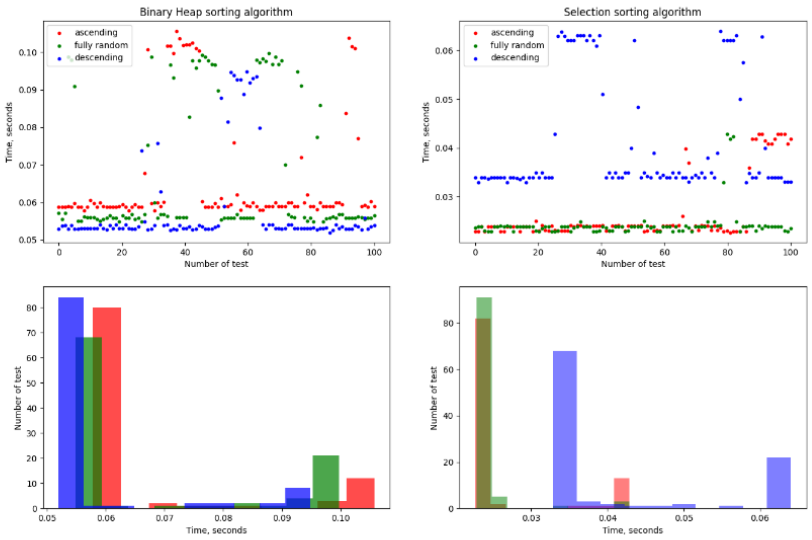
Mimo to, że oba algorytmy sortowania są dość niezależne od rozkładu danych wejściowych (optymistyczna i pesymistyczna złożoność obliczeniowa obu algorytmów nie różni się od oczekiwanej), wciąż dane wejściowe można przygotować w taki sposób, że sortowanie będzie działać wolniej lub szybciej. Przygotowałem zbiór danych wejściowych, które już są prawie posortowane (większość elementów znajdują się na swoich pozycjach) rosnąco lub malejąco, i otrzymałem wyniki pokazane na wykresie (Wykres 2). Widać, że około 60 procent ciągów wejściowych, które były posortowane



Wykres 2 Porównanie zależności czasu sortowania od struktury danych wejściowych (prawie posortowanych malejąco/rosnąco lub losowych) dla algorytmu sortowania przez kopcowanie (po lewej stronie) i algorytmu sortowania przez wstawianie (po prawej stronie). Dolne histogramy – to są podsumowania górnych wykresów.
Źródło: opracowanie własne.

prawie malejąco algorytm sortowania przez kopcowanie posortował najszybciej, gdy algorytm przez wybieranie – najdłużej. Dla sortowania kopcowego taki wzór wynika z tego, że utworzyć kopic jest prościej dla danych, które są

posortowane malejąco. Natomiast dla algorytmu przez wybieranie to wynika z poleceń K16 i K17 w odpowiednym pseudokodzie. Idąc po ciągu liczb, spotykając każdego razu liczbę mniejszą, zapisujemy jej indeks, co zwiększa czas roboty programu. Uzupełnić ten pomysł można patrząc na ciągi liczb posortowanych rosnąco. Oczywistym jest pomysł, że zrobić kopiec z ciągu rosnącego jest dość trudno. Natomiast znaleźć najmniejszy element można prawie z pierwszego razu, dlatego nie jest potrzebnym przepisywanie indeksu każdego razu.



Wykres 3 Porównanie zależności czasu sortowania od struktury danych wejściowych (posortowanych malejąco/rosnąco lub losowych) dla algorytmu sortowania przez kopcowanie (po lewej stronie) i algorytmu sortowania przez wstawianie (po prawej stronie). Dolne histogramy – to są podsumowania górnych wykresów.
Źródło: opracowanie własne.

Jeżeli dane są posortowane idealnie, to wyżej omówione właściwości są jeszcze lepiej widoczne. Takie rezultaty są podane na wykresie (Wykres 3).

DOKUMENTACJA Z DOŚWIADCZEŃ

Algorytm wczytywania plików, który znajduje się w pliku „file_handling.py", przyjmuje wszystkie ciągi liczbowe separowane spacją, przecinkiem lub średnikiem. W przypadku więcej niż jednego ciągu, należy ich oddzielić nową linią. Na rysunku (Rysunek 4) podano przykład danych wejściowych i wyników otrzymanych po uruchomieniu programu.

a)

```
2 4 5 6 7
23 34 23 23 65 72 74 29
23 12 96 03 28 304
32 23 54 76 84
21 12 34 23 188 92
1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 3 2 1 5 8 6 9 7 4 1 2 3
```

b)

```
Sequence #0:
2, 4, 5, 6, 7

Sequence #1:
23, 23, 23, 29, 34, 65, 72, 74

Sequence #2:
3, 12, 23, 28, 96, 304

Sequence #3:
23, 32, 54, 76, 84

Sequence #4:
12, 21, 23, 34, 92, 188

Sequence #5:
1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9
```

Rysunek 4 Przykładowy wygląd pliku wejściowego (a) i wyjściowego (b).
Źródło: opracowanie własne.

Po uruchomieniu każdego sortowania w konsoli są podawane niektóre dane o zadaniu, które oblicza się w tej chwili. Na przykład, dla użytkownika mogą być użytecznymi dane o czasie wykonania sortowania lub który algorytm jest używany. Przykładowe dane z konsoli są podane na rysunku (Rysunek 5).

```
----- Nowe zadanie z pliku: ".\tests\sel_a_asc_test.txt" -----------------------------------------------------------

Przetwarzanie pliku ".\tests\sel_a_asc_test.txt" : [##########] 100% Gotowy...

Rozpoznano 100 wejsciowych ciagow w pliku ".\tests\sel_a_asc_test.txt".
Wszystkie odpowiedzi zostaly zapisane w pliku ".\tests\sel_a_asc_result.txt".
Wykonano sortowanie przez wybieranie.
Czas roboty algorytmu wynosi 2.651 s.
Sredni czas przetwarzania jednego ciagu wynosi 0.02651 s.



----- Nowe zadanie z pliku: ".\tests\sel_f_rand_test.txt" -----------------------------------------------------------

Przetwarzanie pliku ".\tests\sel_f_rand_test.txt" : [##########] 100% Gotowy...

Rozpoznano 100 wejsciowych ciagow w pliku ".\tests\sel_f_rand_test.txt".
Wszystkie odpowiedzi zostaly zapisane w pliku ".\tests\sel_f_rand_result.txt".
Wykonano sortowanie przez wybieranie.
Czas roboty algorytmu wynosi 2.67 s.
Sredni czas przetwarzania jednego ciagu wynosi 0.0267 s.
```

Rysunek 5 Przykładowy wygląd konsoli.
Źródło: opracowanie własne.

WNIOSKI

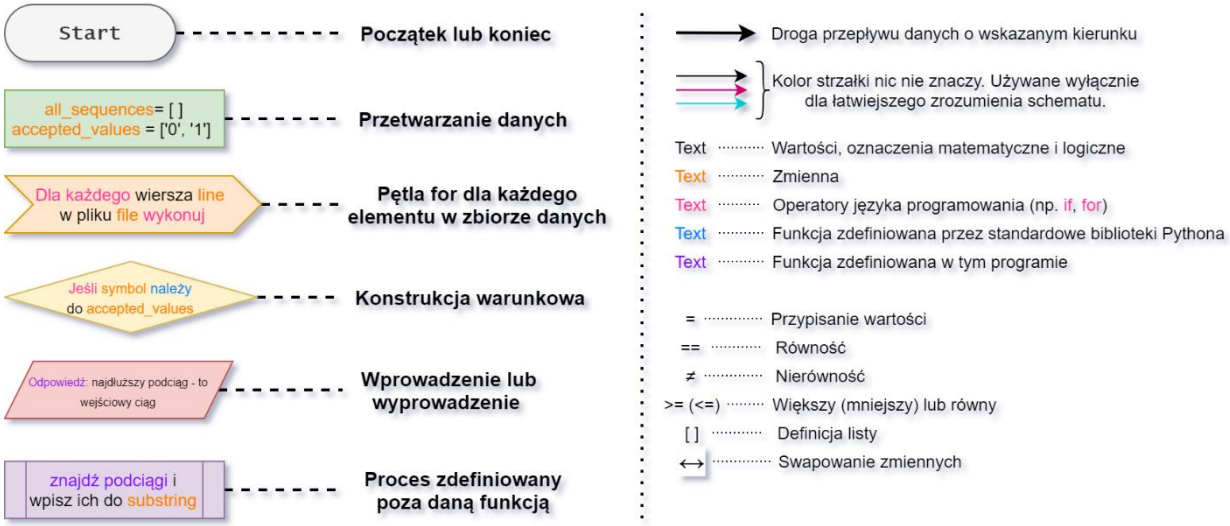
W danym projekcie przeanalizowałem dwa algorytmy sortowania ciągów liczbowych o złożonościach $\theta(n^2)$ i $\theta(n \log n)$, a mianowicie sortowanie przez wybieranie i sortowanie przez kopcowanie.

Wynikiem pracy został program, który ma możliwości:

- wczytywania danych wejściowych z różnych plików tekstowych oraz plików „*.csv”;
- tworzenia danych wejściowych w postaci ciągu losowego lub ciągów o różnym poziomie optymistyczności/pesymistyczności;
- wypisywania posortowanych ciągów do plików;
- demonstracji złożoności algorytmów sortowania;
- demonstracji zależności czasu sortowania od struktury danych wejściowych.

Także były utworzone schematy blokowe, pseudokody i szczegółowe schematy działania obu algorytmów, wykresy porównania złożoności czasowej algorytmów i zależności czasu sortowania od pliku wejściowego.

LEGENDA PSEUDOKODÓW I SCHEMATÓW BLOKOWYCH



Rysunek 6 Legenda pseudokodów i schematów blokowych

SPISY ODWOŁAŃ

RYSUNKI

SCHEMATY

SCHEMATY BLOKOWE

WYKRESY

BIBLIOGRAFIA

Foundation, Python Software. *Python 3.8.6 documentation.* November 27, 2001-2020. https://docs.python.org/3.8/.

Heapsort - Wikipedia. 22 12 2020. https://en.wikipedia.org/wiki/Heapsort (data uzyskania dostępu: 01 14, 2021).

Jerzy Wałaszek. *Sortowanie przez wybór - Selection Sort.* 2012. https://eduinf.waw.pl/inf/alg/003_sort/0009.php (data uzyskania dostępu: 01 14, 2021).

—. *Sortowanie stogowe - Heap Sort.* 2012. https://eduinf.waw.pl/inf/alg/003_sort/0015.php (data uzyskania dostępu: 01 14, 2021).

planetB. *Syntax Highlight Code In Word Documents.* 2018. http://www.planetb.ca/syntax-highlight-word (data uzyskania dostępu: 11 27, 2020).

Selection Sort - Wikipedia. 29 12 2020. https://en.wikipedia.org/wiki/Selection_sort (data uzyskania dostępu: 01 14, 2021).

Stack Exchange Inc. *Stackoverflow.* 2020 . https://stackoverflow.com/questions/3160699/python-progress-bar.

KOD PROGRAMU

Utworzono za pomocą (planetB 2018).

algorithms.py

```
from math import floor      # Importing function floor() from the math module

def SelectionSort(mylist, reverse = False):
    ''''' Sort the list in ascending order and return it.

        The sort is in-
place (i.e. the list itself is modified) and not stable (i.e. the order of two equal
        elements is not maintained).

        The reverse flag can be set to sort in descending order.'''

    length = len(mylist)

    # Defining the direction of the sorting
    if reverse:
        start, stop, step = length-1, -1, -1
    else:
        start, stop, step = 0, length, 1

    # Sorting the list from the smallest value to the biggest one
    for i in range(start, stop - 1, step):
        min_prey, p_index= mylist[i], i         # min_prey - first value of the unsorte
d part of the list
                                                # p_index - index of the min_prey value
 in the list

        # Looking for smaller elements then min_prey in the unsorted part of the list
        for j in range(i+step, stop, step):
            if min_prey > mylist[j]:
                min_prey, p_index = mylist[j], j

        # Swapping the first element of the unsorted part of the list with the smallest
 element in the same part
        mylist[i], mylist[p_index] = mylist[p_index], mylist[i]

    return mylist



def BinaryHeapSort(mylist, reverse = False):
    ''''' Sort the list in ascending order and return it.

        The sort in-
place (i.e. the list itself is modified) and not stable (i.e. the order of two equal
        elements is not maintained).

        The reverse flag can be set to sort in descending order.'''

    length = len(mylist)

    # Creating a heap from the list
    for ind in range(length-2, -1, -1):
        Heapify(mylist, length, ind)

    for i in range(len(mylist)):

        # Replacing the first element with the last one
        if length>1:
            mylist[0], mylist[length-1] = mylist[length-1], mylist[0]

```

```
56.         # Shrink the length of the list
57.         length -= 1
58.
59.         # Restoring the rule of the heap
60.         Heapify(mylist,length)
61.
62.     # Reversing the list if an appropriate flag is set
63.     if reverse:
64.         for index in range(floor(len(mylist)/2)):
65.             mylist[index], mylist[-index-1] = mylist[-index-1], mylist[index]
66.
67.     return mylist
68.
69.
70.
71. def Heapify(mylist, length, j=0):
72.     ''''' Restore the rule of the Heap from the perspective of the given index.
73.     (i.e. checks if children of the index are in the right order) '''
74.
75.     # Looking through the heap
76.     while j*2+1<length:
77.
78.         # Indexes of the children
79.         ch1, ch2 = j*2+1, j*2+2
80.
81.         # Finding the biggest child of the current parent
82.         if ch2<length and mylist[ch1] < mylist[ch2]:
83.             greater_child_ind = ch2
84.         else:
85.             greater_child_ind = ch1
86.
87.         # Repairing the heap if the greatest child is greater than it's parent
88.         if mylist[j]<mylist[greater_child_ind]:
89.             mylist[j], mylist[greater_child_ind] = mylist[greater_child_ind], mylist[j]
90.             j = greater_child_ind
91.         else:
92.             break
93.
94.     return mylist
```

console_handling.py

```
1.  import sys
2.  # The next function is not written by Vitalii Morskyi (just modified)
3.  # Source: https://stackoverflow.com/questions/3160699/python-progress-bar
4.  def update_progress(progress, path_in):
5.      ''''' Displays or updates a console progress bar. WORKS ONLY WITH CONSOLE
6.      Accepts a float between 0 and 1. Any int will be converted to a float.
7.      A value under 0 represents a 'halt'.
8.      A value at 1 or bigger represents 100%.'''
9.
10.     barLength = 10 # Modify this to change the length of the progress bar
11.     status = ""
12.     if isinstance(progress, int):
13.         progress = float(progress)
14.     if not isinstance(progress, float):
15.         progress = 0
16.         status = "error: progress var must be float\r\n"
17.     if progress < 0:
18.         progress = 0
19.         status = "Halt...\r\n"
20.     if progress >= 1:
21.         progress = 1
22.         status = "Gotowy...\r\n"
23.     block = int(round(barLength*progress))
24.     text = "\rPrzetwarzanie pliku \"{3}\" : [{0}] {1}% {2}".format( "#"*block +
```

```
25.         "-"*(barLength-block), round(progress*100,1), status, path_in)
26.     sys.stdout.write(text)
27.     sys.stdout.flush()
28.
29.
30.
31. def draw_separator(path):
32.     ''''' This function draws a "pretty" separator between tasks'''
33.     heading = " Nowe zadanie z pliku: \"{}\" ".format(path)
34.     separator = '-'*5 + heading + '-'*(115 - len(heading))
35.     print('\n\n\n\n' + separator + '\n')
36.
37.
38. def task_info(number_of_sequences, time_results, path_in, path_out, algotithm_type):
39.     ''''' Print some useful info about current task to the console '''
40.
41.     if str(number_of_sequences)[-2:] in ['11', '12', '13', '14']:       # excluding -
    teen numbers
42.         teened = True
43.     else:
44.         teened = False
45.     last_digit=str(number_of_sequences)[-1]
46.
47.     if last_digit=='1' and not teened:
48.         insert_text = 'wejsciowy ciag'
49.     elif last_digit in ['2', '3', '4'] and not teened:
50.         insert_text = 'wejsciowy ciagi'
51.     else:
52.         insert_text = 'wejsciowych ciagow'
53.
54.     algotithm = ''
55.     if algotithm_type == 's':
56.         algotithm = 'sortowanie przez wybieranie'
57.     elif algotithm_type == 'h':
58.         algotithm = 'sortowanie kopcowe'
59.
60.     time_used = round(sum(time_results),3)
61.     avarage_time_used = round(time_used/number_of_sequences, 5)
62.     print("Rozpoznano {0} {1} w pliku \"{2}\". ".format(number_of_sequences, insert_tex
    t, path_in)+
63.         "\nWszystkie odpowiedzi zostaly zapisane w pliku \"{}\". ".format(path_out) +
64.         "\nWykonano {}.".format(algotithm)+
65.         "\nCzas roboty algorytmu wynosi {} s.".format(time_used) +
66.         "\nSredni czas przetwarzania jednego ciagu wynosi {} s.\n".format(avarage_time_
    used), flush = True)
```

file_handling.py

```
1.  def ReadInputFile(path, separator = None):
2.      ''''' Return the list of the sequences of the integers from the file in the given p
    ath and number of misunderstood
3.          words. All float numbers are floored (i.e. rounded to the closest smaller integ
    er).
4.
5.          Return -
    1 and number of misunderstood words if the file was empty or there was not a single num
    ber in the file.
6.          Return -2 if file was not found.
7.          Return -3 if any other error appeared.
8.
9.          The separator can be changed, if numbers in the input file are separated with a
     special symbol
10.         (for ex. separator = "_"; default: separator = " ").'''
11.     #".\\tests\\initial_test.txt"
12.     try:
13.         lists, err_count = [], 0        # Defining the output list and the variable to
    count misunderstood numbers
```

```
        with open(path,mode='r') as file:   # Opens the input file and reads it line by line
            for line in file:
                line = line.replace(", ", " ").replace(",", " ")
                line = line.replace("; ", " ").replace(";", " ")
                this_sequence = []
                for word in line.strip().split(separator):  # Cutting the sequence with the respect to the separator
                    try:
                        this_sequence.append(int(word))
                    except ValueError:
                        err_count += 1
                if this_sequence != []:
                    lists.append(this_sequence)
        if lists != []:
            return lists, err_count
        else:
            return -1, err_count    # File is empty, number of missed numbers/words
    except FileNotFoundError:
        return -2, 0                # File is not found
    except:
        return -3, 0                # Something else went wrong


def PrepareFile(path):
    ''''' Cleaning (or creating) the file in the given path '''

    with open(path,mode='w', encoding='utf-8') as file:
        pass


def AppendFile(path, out_list, label=None, end='\n\n'):
    ''''' Adds the list out_list to the end of the file in the given path.

        If label is given then the output will be labeled.

        The ending of each output can be modified by "end" parameter.'''

    with open(path,mode='a', encoding='utf-8') as file:
        if type(label)==int:
            file.write('Sequence #'+str(label)+':\n')
        elif type(label)==str:
            file.write('Sequence \''+str(label)+'\':\n')
        file.write(str(out_list)[1:-1]+end)
```

main.py

```
import test, algorithms as alg, file_handling as f_handl, console_handling as cl_handl

from time import time

def Sort(path_in, path_out, algorithm = 's',
    captions = True, clear_out_f = True, print_info = True, progress_bar = True):
    ''''' Sort all sequences in the file with path "path_in" and writes it to the file
     with the path "path_out".

     The sorting algorithm can be specified by the "algorithm" parameter:
        's', 'S' - selection sort
        'h', 'H' - binary heap sort

     If captions flag is set then the output sequences will be labeled.

     "clear_out_f" flag can be turned off not to clear an output file before
     writing down the results.

```

```
18.      If print_info flag is set then some info about the task will be printed
19.      to the console.
20.     '''
21.
22.     # Reading an input file
23.     sequences = f_handl.ReadInputFile(path_in)
24.     if type(sequences[0]) != list:
25.         if sequences[0] == -1:
26.             print("File '{}' exist, but is empty or unreadable.\n Number of misundersto
    od words: {}".format(
27.                 path_in, sequences[1]))
28.         elif sequences[0] == -2:
29.             print('File \'{}\' not found'.format(path_in))
30.         elif sequences[0] == -3:
31.             print('Something went wrong while reading file \'{}\''.format(path_in))
32.         return -1
33.     else:
34.         if sequences[1]>0:
35.             print('Number of misunderstood words:',sequences[1])
36.         sequences = sequences[0]
37.
38.     counter = 1
39.     times = []
40.     if clear_out_f:
41.         f_handl.PrepareFile(path_out)   # Clearing the output file
42.
43.     cl_handl.draw_separator(path_in)
44.
45.     counter = 0
46.     for sequence in sequences:
47.
48.         if algorithm.lower() == 's':
49.             start_time = time()
50.             array = alg.SelectionSort(sequence)
51.             end_time = time()
52.         elif algorithm.lower() == 'h':
53.             start_time = time()
54.             array = alg.BinaryHeapSort(sequence)
55.             end_time = time()
56.
57.         times += [round(end_time-start_time, 4)]
58.
59.         if captions:
60.             f_handl.AppendFile(path=path_out, out_list=array, label=counter)
61.         else:
62.             f_handl.AppendFile(path=path_out, out_list=array, end='\n')
63.
64.         counter += 1
65.         if progress_bar:
66.             cl_handl.update_progress(counter/len(sequences), path_in)
67.
68.     if print_info:
69.         print('')
70.         cl_handl.task_info(len(sequences), times, path_in, path_out, algorithm.lower())
71.
72.     return times
73.
74.
75.
76. def demonstrate(generate_data = False):
77.     ''''' Implementation of all functions and showing the graph of the results. In othe
    r words - examples.'''
78.     # path=".\\tests\\test.txt"
79.
80.     # Creating an input file for the test
81.     test.CreateInput(path=".\\tests\\main_test.txt", Nmin = 0, Nmax = 1000, sequences =
     100, len_start = 10,
```

```
82.         len_incr = 10, len_mult = 1, complexity = 0)
83.
84.     # Creating an numpy array of the times of algorithm working time
85.     sel_times = np.array(Sort('.\\tests\\main_test.txt', '.\\tests\\sel_result.txt', al
    gorithm = 'S', captions = False))
86.     heap_times = np.array(Sort('.\\tests\\main_test.txt','.\\tests\\heap_result.txt',al
    gorithm = 'H', captions = False))
87.
88.     # Values for x axes in numpy array format. Similar to np.array(range(100))
89.     x = np.linspace(0, 100, 100)
90.
91.     plt.figure('Algorithms comparison')
92.     plt.scatter(x, sel_times, s=10, c = 'red',  label = 'Selection Sort')
93.     plt.scatter(x, heap_times, s=10, c = 'green', label = 'Binary Heap Sort')
94.     plt.xlabel('Number of numbers')
95.     plt.ylabel('Time, seconds')
96.     plt.title('Binary Heap Sort and Selection Sort algorithms comparison')
97.     plt.legend(loc=2)
98.     ticks = np.linspace(0, 100, 6)
99.     tick_labels = np.linspace(10, 1010, 6)
100.      plt.xticks(ticks=ticks, labels=tick_labels)
101.      plt.show()
102.      plt.close()
103.
104.
105.      n = 100
106.      s_len = 10000
107.      s_len_2 = int(s_len/10)
108.      x = np.linspace(0, n, n)
109.
110.      print('\nCreating a new set of data...', flush = True)
111.
112.      test.CreateInput(path=".\\tests\\a_ascending_test.txt", Nmin = 0, Nmax = s_len*5,
     sequences = n,
113.          len_start = s_len, len_incr = 0, len_mult = 1, complexity = -
    1, distrib = 15, dist_incr = 0, dist_mult = 1)
114.      test.CreateInput(path=".\\tests\\f_random_test.txt",    Nmin = 0, Nmax = s_len*5,
     sequences = n,
115.          len_start = s_len, len_incr = 0, len_mult = 1, complexity =  0, distrib = 15,
     dist_incr = 0, dist_mult = 1)
116.      test.CreateInput(path=".\\tests\\a_descending_test.txt",Nmin = 0, Nmax = s_len*5,
     sequences = n,
117.          len_start = s_len, len_incr = 0, len_mult = 1, complexity =  1, distrib = 15,
     dist_incr = 0, dist_mult = 1)
118.
119.      test.CreateInput(path=".\\tests\\sel_a_asc_test.txt",  Nmin = 0, Nmax = s_len_2*5
    , sequences = n,
120.          len_start = s_len_2, len_incr = 0, len_mult = 1, complexity = -
    1, distrib = 15, dist_incr = 0, dist_mult = 1)
121.      test.CreateInput(path=".\\tests\\sel_f_rand_test.txt", Nmin = 0, Nmax = s_len_2*5
    , sequences = n,
122.          len_start = s_len_2, len_incr = 0, len_mult = 1, complexity =  0, distrib = 1
    5, dist_incr = 0, dist_mult = 1)
123.      test.CreateInput(path=".\\tests\\sel_a_desc_test.txt", Nmin = 0, Nmax = s_len_2*5
    , sequences = n,
124.          len_start = s_len_2, len_incr = 0, len_mult = 1, complexity =  1, distrib = 1
    5, dist_incr = 0, dist_mult = 1)
125.
126.
127.      heap_times_m1 = np.array(Sort('.\\tests\\a_ascending_test.txt','.\\tests\\a_ascen
    ding_result.txt',
128.          algorithm = 'H', captions = False))
129.      heap_times_0 = np.array(Sort('.\\tests\\f_random_test.txt','.\\tests\\f_random_re
    sult.txt',
130.          algorithm = 'H', captions = False))
131.      heap_times_1 = np.array(Sort('.\\tests\\a_descending_test.txt','.\\tests\\a_desce
    nding_result.txt',
132.          algorithm = 'H', captions = False))
```

```
133.
134.      sel_times_m1 = np.array(Sort('.\\tests\\sel_a_asc_test.txt','.\\tests\\sel_a_asc_
    result.txt',
135.          algorithm = 'S', captions = False))
136.      sel_times_0= np.array(Sort('.\\tests\\sel_f_rand_test.txt','.\\tests\\sel_f_rand_
    result.txt',
137.          algorithm = 'S', captions = False))
138.      sel_times_1 = np.array(Sort('.\\tests\\sel_a_desc_test.txt','.\\tests\\sel_a_desc
    _result.txt',
139.          algorithm = 'S', captions = False))
140.
141.      plt.figure('Inputs comparison')
142.
143.      plt.subplot(2,2,1)
144.      plt.scatter(x, heap_times_m1,s=10, c = 'red',  label = 'almost ascending')
145.      plt.scatter(x, heap_times_0, s=10, c = 'green',label = 'fully random')
146.      plt.scatter(x, heap_times_1, s=10, c = 'blue', label = 'almost descending')
147.      plt.xlabel('Number of test')
148.      plt.ylabel('Time, seconds')
149.      plt.title('Binary Heap sorting algorithm')
150.      plt.legend(loc=2)
151.
152.      plt.subplot(2,2,2)
153.      plt.scatter(x, sel_times_m1,s=10, c = 'red',  label = 'almost ascending')
154.      plt.scatter(x, sel_times_0, s=10, c = 'green',label = 'fully random')
155.      plt.scatter(x, sel_times_1, s=10, c = 'blue', label = 'almost descending')
156.      plt.xlabel('Number of test')
157.      plt.ylabel('Time, seconds')
158.      plt.title('Selection sorting algorithm')
159.      plt.legend(loc=2)
160.
161.      plt.subplot(2,2,3)
162.      plt.hist(heap_times_m1, bins=15, color = 'red',  label = 'almost ascending',    a
    lpha = 0.7)
163.      plt.hist(heap_times_0,  bins=15, color = 'green',label = 'fully random',        a
    lpha = 0.7)
164.      plt.hist(heap_times_1,  bins=15, color = 'blue', label = 'almost descending',   a
    lpha = 0.7)
165.      plt.xlabel('Time, seconds')
166.      plt.ylabel('Number of test')
167.
168.      plt.subplot(2,2,4)
169.      plt.hist(sel_times_m1,  bins=15, color = 'red',  label = 'almost ascending',    a
    lpha = 0.5)
170.      plt.hist(sel_times_0,   bins=15, color = 'green',label = 'fully random',        a
    lpha = 0.5)
171.      plt.hist(sel_times_1,   bins=15, color = 'blue', label = 'almost descending',   a
    lpha = 0.5)
172.      plt.xlabel('Time, seconds')
173.      plt.ylabel('Number of test')
174.
175.      plt.show()
176.
177.
178.
179.  def main():
180.      ''''' Main function '''
181.
182.      # Initial test
183.      Sort('.\\tests\\initial_test.txt', '.\\tests\\initial_result.txt', algorithm = 'H
    ',
184.          captions = True, progress_bar = False)
185.
186.      # The most important tests are stored in the demonstrate() function
187.      try:
188.          import matplotlib.pyplot as plt, numpy as np
189.          global plt, np
190.      except ModuleNotFoundError as exc:
```

```
191.          print('\n\n{}\n\nPlease install the module mentioned above\n'.format(exc)+
192.              'to be able to appreciate some beautiful plots.')
193.      else:
194.          demonstrate()
195.
196.      # If you need the console not to close immediately after
197.      # finishing your task, then please uncomment the following two rows.
198.      # import os
199.      # os.system("pause")
200.
201.
202.  if __name__ == "__main__":
203.      main()
```

test.py

```
1.  import random, file_handling as f_handl
2.
3.
4.  def CreateInput(path=".\\tests\\test.txt", complexity = 0, Nmin = 0, Nmax = 1000, seque
    nces = 10,
5.      len_start = 10, len_incr = 1, len_mult = 1, distrib = 0, dist_incr = 0, dist_mult =
     1):
6.      ''''' Create a file with some input sequences depending on given parameters.
7.
8.      path        - path of the output file;
9.      complexity  - the type of input data:
10.         0  - random sequence of values (default)
11.         -
    1 - almost ascending sequence of values, but some elements may not be on their position
    s
12.         1  - almost descending sequence of values, but some elements may not be on thei
    r positions
13.     Nmin        - minimal possible value
14.     Nmax + 1    - maximum possible value
15.     sequences   - number of sequences in the resulting file
16.     len_start   - starting length of the line
17.     len_incr    - line length increment after each line
18.     len_mult    - line length multiplication after each line
19.
20.     distrib     - (only for complexity values -
    1 or 1)determines the degree of the entropy of the
21.         sequence (the higher - the more random)
22.     dist_incr   - distribution (distrib) value increment after each line
23.     dist_mult   - distribution (distrib) value multiplication after each line
24.     '''
25.
26.     f_handl.PrepareFile(path)
27.     for i in range(sequences):
28.         if complexity == 0:
29.             f_handl.AppendFile(path = path, out_list = random.choices(range(Nmin, Nmax)
    , k=len_start), end='\n')
30.
31.         elif complexity == -1:
32.
33.             sequence = CreateSpecialSequence(start=Nmin, stop=Nmax, length=len_start, d
    istrib=distrib)
34.             f_handl.AppendFile(path = path, out_list = sequence, end='\n')
35.
36.         elif complexity == 1:
37.             sequence = CreateSpecialSequence(start=Nmax, stop=Nmin, length=len_start, d
    istrib=distrib)
38.             f_handl.AppendFile(path = path, out_list = sequence, end='\n')
39.
40.         len_start += len_incr
41.         len_start *= len_mult
42.         distrib += dist_incr
```

```
        distrib *= dist_mult


def CreateSpecialSequence(start=0, stop=None, length=None, distrib=None):
    ''''' Return sequence of almost descending or ascending (depending on the start and stop parameters) elements
    (i.e. the sequence is not sorted perfectly, most of the elements are sorted).

    The distribution parameter determine the degree of the entropy of the sequence (the higher - the more random).'''

    sequence = []
    min_point = min(start, stop)
    if distrib == None:
        distrib = 0
    # represents correlation between the length of the line and the distance from start to stop point
    coef = (abs(stop-start)-distrib)/length

    # determining if we are moving forward or backward
    direction = int(abs(stop-start)/(stop-start))
    if direction==1:
        begin, end = 0, length
    else:
        begin, end = length, 0

    # adding a random element from the range 'distribution'
    for i in range(begin, end, direction):
        sequence.append(random.randint(min_point+int(i*coef), min_point+int(i*coef)+int(distrib)))
    return sequence
```