Opracował:

PJATK 19.05.2021 r.

Michał Czerwiak

S21356

**POMIAR CZASU DZIAŁANIA ALGORTYMÓW SORTUJĄCYCH**

SPRAWOZDANIE

1. **Wstęp**

W celu realizacji postawionego zadania, zaimplementowano w języku Python 3 algorytmy sortujące:

* Quicksort:

#quicksort

def partition(arr, p, r):

    pivot = arr[r]

    smaller = p

    for j in range(p, r):

        if arr[j] <= pivot:

            arr[smaller], arr[j] = arr[j], arr[smaller]

            smaller = smaller + 1

    arr[smaller], arr[r] = arr[r], arr[smaller]

    return smaller

def quicksort(arr, p, r):

    if p < r:

        q = partition(arr, p, r)

        quicksort(arr, p, q - 1)

        quicksort(arr, q + 1, r)

* Heapsort:

#heapsort

#tworzenie kopca

def tworzKopiec(arr, n, i):

    largest = i

    l = 2 \* i + 1

    r = 2 \* i + 2

    if l < n and arr[i] < arr[l]:

        largest = l

    if r < n and arr[largest] < arr[r]:

        largest = r

    if largest != i:

        arr[i],arr[largest] = arr[largest],arr[i]

        tworzKopiec(arr, n, largest)

def tworzKopiec1(arr):

    n=len(arr)

    for i in range(n//2 - 1, -1, -1):

        tworzKopiec(arr,n,i)

#sortowanie

def sortujKopiec(arr):

    n = len(arr)

    tworzKopiec1(arr)

    for i in range(n -1, 0, -1):

        arr[i],arr[0] = arr[0],arr[i]

        tworzKopiec(arr, i, 0)

* Bubblesort:

#bubblesort

def bubblesort(arr):

    for i in range(len(arr)):

        j=len(arr)-1

        while j>i:

            if arr[j]<arr[j-1]:

               tmp=arr[j]

                arr[j]=arr[j-1]

                arr[j-1]=tmp

            j-=1

Powyższe implementacje napisano w programie Visual Studio Code.

Pomiary wykonano na komputerze DELL o parametrach:

* Procesor: Intel Core i5-9400H 2,5GHz, 8 Mb cache
* Pamięć RAM: 32GB DDR4 2666MHz

1. **Realizacja pomiarów.**

W celu wykonania pomiarów utworzono listę wypełnioną losowo wygenerowanymi, 180 000 liczb z zakresu (-100, 100).

Następnie wywołano funkcje odpowiedzialne za poszczególne algorytmy sortowania, mierząc jednocześnie czas działania każdej funkcji dla tej samej nieposortowanej listy liczb:

#wypelnianie tablicy losowymi liczbami

list = [random.randint(-100,100) for \_ in range(180000)]

#tworzenie płytkich kopi tablicy vet na potrzeby testów

tabtest1 = copy.copy(list)

tabtest2 = copy.copy(list)

tabtest3 = copy.copy(list)

#pomiar dla quicksort:

start\_time = time.time()

quicksort(tabtest1, 0, len(tabtest1) - 1)

print("Posortowanie quicksort zajęło: %s sec" % (time.time() - start\_time))

#pomiar dla heapsort:

start\_time = time.time()

tworzKopiec(tabtest2, len(tabtest2), 0)

sortujKopiec(tabtest2)

print("Posortowanie heapsort zajęło: %s sec" % (time.time() - start\_time))

#pomiar dla bubblesort:

start\_time = time.time()

bubblesort(tabtest3)

print("Posortowanie bubble zajęło: %s sec" % (time.time() - start\_time))

Kolejnym krokiem było zmierzenie czasu wykonania się powyższych algorytmów dla listy posortowanej i posortowanej malejąco z tym że liczbę elementów w liście zmniejszono do 2500 ponieważ w przypadku list posortowanych zaimplementowany algorytm quicksort zgłaszał błąd przekroczenia liczby wywołań rekurencyjnych dla list powyżej 2500 elementów. Dodatkowo wykonano pomiary czasów działania badanych algorytmów dla listy nieposortowanej zawierającej 2500 elementów.

Wyniki pomiarów zaprezentowano w tabeli nr 1.

Tabela nr 1. Wyniki pomiarów dla algorytmów sortujących.

|  |  |  |  |
| --- | --- | --- | --- |
| Czas [s] | Quicksort | Heapsort | Bubblesort |
| Lista losowa (180 tys el) | 9,433 | 1,009 | 2630,973 |
| Lista losowa (2500 el) | 0,0036 | 0,0095 | 0,5568 |
| Lista posortowana (2500 el) | 0,350 | 0,012 | 0,356 |
| Lista posortowana malejąco (2500 el) | 0,127 | 0,010 | 0,736 |

1. **Wnioski**

* W każdym z badanych przypadków najwolniejszym algorytmem sortowania okazał się algorytm Bubblesort, który w miarę dobrze radzi sobie z niedużymi listami, natomiast jego wydajność spada wraz z wydłużaniem się listy wejściowej.
* W przypadku dużych list losowych najszybszym algorytmem był algorytm Heapsort, który okazał się 9 razy szybszy niż algorytm Quicksort.
* Przy mniejszych listach losowych nie było już niespodzianki i zdecydowanie najszybszym algorytmem był algorytm Quicksort.
* Zaskoczeniem było sortowanie listy już posortowanej. Z tym zadaniem zdecydowanie najlepiej poradził sobie algorytm Heapsort, natomiast algorytm Quicksort wykonał to w czasie zbliżonym do algorytmu Bubblesort.
* Analizując wyniki, można zauważyć, że czas działania algorytmu Heapsort nie zależy od rodzaju listy a jedynie od jej długości, natomiast pozostałe algorytmy, a w szczególności algorytm Quicksort, są wrażliwe na rodzaj listy wejściowej.
* Należy również zauważyć, że nie ma idealnego algorytmu sortowania i jego dobór powinien zależeć od rodzaju listy i liczby elementów, które się w niej znajdują.