Algorytmy i Struktury Danych
Kolokwium 1 (30.III.2023)

Format rozwiązań

Wysłać należy tylko jeden plik: `kol1.py`

Plik można wysyłać wielokrotnie, liczy się ostatnia wersja zapisana w systemie.

Rozwiązanie zadania musi się składać z krótkiego opisu algorytmu (wraz z uzasadnieniem poprawności) oraz jego implementacji. Zarówno opis algorytmu jak i implementacja powinny się znajdować w tym samym pliku Pythona (rozszerzenie `.py`). Opis powinien być na początku pliku w formie komentarza (w pierwszej linii w komentarzu powinno być imię i nazwisko studenta). Opis nie musi być długi—wystarczy kilka zdań, jasno opisujących ideę algorytmu. Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. korzystanie z wbudowanych funkcji sortujących,
2. korzystanie z zaawansowanych struktur danych (np. słowników czy zbiorów),
3. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
4. wypisywanie na ekranie jakichkolwiek napisów innych niż wypisywane przez dostarczony kod (ew. napisy dodane na potrzeby diagnozowania błędów należy usunąć przed wysłaniem zadania).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue`),

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania.

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 punktów. Rozwiązania w innych formatach (np. `.ZIP`, `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 punktów, nawet jeśli będą poprawne.

Testowanie rozwiązań

Żeby przetestować rozwiązanie zadania należy wykonać polecenie: `python kol1.py`

Szablon rozwiązania:	`kol1.py`
Złożoność akceptowalna (2.0pkt):	$O(np)$, gdzie n to liczba elementów w tablicy a p to liczba elementów w przedziałach, w których poszukiwane są elementy sumowane.
Złożoność wzorcowa (+2.0pkt):	$O(n \log n)$, gdzie n to liczba elementów w tablicy.

Dana jest n-elementowa tablica liczb naturalnych `T` oraz dodatnie liczby naturalne k i p, gdzie $k \leq p \leq n$. Niech z_i będzie k-tą największą spośród elementów: `T[i]`, `T[i+1]`, ..., `T[i+p-1]`. Innymi słowy, z_i to k-ty największy element w `T` w przedziale indeksów od i do $i + p - 1$ włącznie.

Doprecyzowanie: Rozważmy tablicę `[17,25,25,30]`. W tej tablicy 1-wszy największy element to 30, 2-gi największy element to 25, 3-ci największy element to także 25 (drugie wystąpienie), a 4-ty największy element to 17.

Proszę zaimplementować funkcję `ksum(T, k, p)`, która dla tablicy `T` (o rozmiarze n elementów) i dodatnich liczb naturalnych k i p ($k \leq p \leq n$) wylicza i zwraca wartość sumy:

$$z_0 + z_1 + z_2 + \ldots + z_{n-p}$$

Przykład. Dla wejścia:

```
T = [7,9,1,5,8,6,2,12]
k = 4
p = 5
```

wywołanie `ksum(T, k, p)` powinno zwrócić wartość 17 (odpowiadającą sumie $5 + 5 + 2 + 5$).
Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.