Algorytmy i Struktury Danych
Kolokwium uzupełniajace (4.VII.2023)

Format rozwiązań

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 zaawansowanych struktur danych (np. wbudowanej kolejki priorytetowej, słowników czy zbiorów),
2. korzystanie z wbudowanych algorytmów sortujących,
3. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
4. modyfikowanie testów dostarczonych wraz z szablonem,
5. 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`.

Wszystkie inne algorytmy (w tym sortowania) lub struktury danych (w tym kolejka priorytetowa) 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. `.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: `python3 kolu.py`

Szablon rozwiązania:	`kolu.py`
Złożoność akceptowalna (2.0pkt):	$O(n \log n)$
Złożoność wzorcowa (+2.0pkt):	$O(n)$
Gdzie n to liczba kubełków z lodami.	

Danych jest n kubełków z lodami. W każdym kubełku jest pewna objętość lodów (reprezentowana przez dużą liczbę naturalną). Zjedzenie jednego kubełka zajmuje jedną minutę (niezależnie od tego ile zawiera lodów). Z upływem każdej minuty w każdym niepustym kubełku topi się 1 jednostka objętości lodów (i nie można jej później zjeść). Zaproponuj i zaimplementuj algorytm wyliczający maksymalną objętość lodów, którą w sumie można zjeść.

Algorytm należy zaimplementować jako funkcję:

```
ice_cream( T )
```

przyjmującą tablicę liczb naturalnych `T` i zwracającą maksymalną objętość lodów, którą w sumie można zjeść. Element `T[i]` to początkowa objętość lodów w i-tym kubełku. Proszę uzasadnić poprawność zaproponowanego algorytmu i oszacować jego złożoność obliczeniową.

Przykład. Dla wejścia:

```
T = [5, 1, 3, 7, 8]
```

wynikiem jest 17. Lody można jeść zaczynając w pierwszej minucie od czwartego kubełka z 7 jednostkami lodów. W drugiej minucie zjadamy pozostałe 7 jednostki z kubełka nr 5 (początkowo było tam 8 jednostek lodów ale 1 jednostka roztopiła się po pierwszej minucie). W trzeciej minucie zjadamy 3 pozostałe jednostki w pierwszym kubełku. W tym momencie nie możemy zjeść nic więcej ponieważ lody z kubełków 2 i 3 całkowicie się już roztopiły. Tak więc otrzymujemy 7 + 7 + 3 = 17