Algorytmy i struktury danych – ćwiczenia 2
(2022/20230

Zadanie 2.1
Rozważmy następujący algorytm sortowania n różnych liczb $x_1, x_2, \ldots, x_n$ w porządku malejącym.
Algorytm Szymka R.
begin
  X := {$x_1, x_2, \ldots, x_n$};
  Zainicjuj stos S jako pusty;
  while Not Empty(X) **do**
  begin
    Weź dowolny element x ze zbioru X;
    X := X \ {x};
    Usuń ze stosu S wszystkie elementy większe od x i wstaw je z powrotem do zbioru X;
    Umieść x na wierzchołku stosu S
  end;
  wypisz kolejno elementy ze stosu S
end;

Przyjmijmy, że operacjami dominującymi są operacje stosowe Top, Push, Pop.

a) Jaka jest pesymistyczna złożoność sortowania algorytmem Szymka R.?

   Załóżmy teraz, że element *x* wybieramy losowo ze zbioru *X* z rozkładem jednostajnym – każdy element może zostać wylosowany z prawdopodobieństwem $1/|X|$.

b) Udowodnij, że oczekiwana liczba losowań elementu *x* w algorytmie wynosi $O(n^2)$.
c) Dokonaj analizy oczekiwanej liczby operacji dominujących w algorytmie Szymka R. w opisanym wyżej modelu probabilistycznym.

Zadanie 2.2
Niech n będzie dodatnią liczbą całkowitą. Dla dodatniej liczby całkowitej k powiemy, że ciąg liczb $a[1],\ldots,a[n]$ jest *k-dobry*, jeżeli każda inwersja (i,j), $1 \le i < j \le n$, spełnia $j \le i + k$.

a) Zaproponuj asymptotycznie optymalny ze względu na porównania algorytm sortujący ciągi *k-dobre*. Uzasadnij asymptotyczną optymalność swojego algorytmu. Uwaga: w tym zadaniu argumentami funkcji złożoności są k i n.
b) Zaproponuj wydajny czasowo i pamięciowo algorytm, który sprawdza, czy dany ciąg liczb $a[1],\ldots,a[n]$, dla zadanej dodatniej liczby całkowitej k, jest k-dobry. Uzasadnij poprawność swojego algorytmu i dokonaj analizy jego złożoności czasowej i pamięciowej.

Zadanie 2.3 (opcjonalnie)
Dana jest tablica $a[1..n]$ parami różnych elementów pochodzących ze zbioru z liniowym porządkiem. Należy posortować tablicę a rosnąco. Jedyną operacją służącą do porównywania elementów między sobą jest funkcja *ile*(x,y), której wynikiem jest liczba całkowita k zdefiniowana tak, że

$$|k| = |\{1 \le i \le n\colon \min(x,y) \le a[i] \le \max(x,y)\}|.$$

Wartość k jest ujemna tylko wtedy, gdy x jest większe od y.

a) Udowodnij, że każdy algorytm sortujący *a* wywoła funkcję *ile* w pesymistycznym przypadku co najmniej *n*-1 razy.
b) Zaproponuj algorytm sortowania *a* w miejscu za pomocą O(*n*) wywołań funkcji *ile* i O(*n*) zamian.

Zadanie 2.4 (opcjonalnie)
W tablicy *a*[1..*n*] dany jest ciąg jednocześnie 7- i 11-uporządkowany. Udowodnij, że algorytm Insertion Sort sortuje *a* w czasie liniowym.
Przypomnienie: powiemy, że ciąg *a* jest *k*-uporządkowany, $k > 0$, gdy dla każdego $i = 1, 2, \ldots, n - k$, $a[i] \leq a[i+k]$.

Zadanie 2.5 (Przesunięcie cykliczne w miejscu)
Dana jest *n*-elementowa tablica *a*[1..*n*] oraz liczba całkowita $k \in [1..n]$.
Zaproponuj liniowy algorytmy przesunięcia cyklicznego elementów tablicy *a* o *k* pozycji w lewo.
Przykład: ciąg [1,2,3,4,5] przesunięty cyklicznie o 2 w lewo ma postać [3,4,5,1,2].

Zadanie 2.6
Dana jest *n*-elementowa tablica *a*[1..*n*] zawierająca tylko 0 i 1.
a) Zaprojektuj wydajny algorytm sortowania *a* stabilnie i w miejscu.
b) Załóżmy, że $n = 2k$ i w *a* znajduje się dokładnie *k* zer i *k* jedynek. Chcemy tablicę *a* posortować tak, żeby zera i jedynki były ułożone na przemian, począwszy od zera, tj. 010101... Zaproponuj wydajny algorytm, który wykona to w miejscu i stabilnie.

Zadanie 2.7 (opcjonalnie)
W tym zadaniu rozważamy rekurencyjny algorytm sortowania przez scalanie, w którym scalanie dwóch posortowanych ciągów odbywa się w sposób klasyczny: na swoją docelową pozycję trafia mniejszy z początkowych elementów scalanych ciągów.

Przykład
Podczas scalania ciągów [2, 4, 5, 8] oraz [1, 3, 6, 7] porównywane są kolejno 2 z 1, 2 z 3, 4 z 3, 4 z 6, 5 z 6, 8 z 6 oraz 8 z 7.

Zaprojektuj liniowy algorytm, który sprawdzi, czy w wyniku wykonania algorytmu sortowania przez scalanie na danej permutacji *p*[1..*n*] liczb naturalnych 1, …, *n,* porównane zostaną ze sobą zadane z góry, dwie różne liczby *a* i *b* ze zbioru {1, …, *n*}.

Zadanie 2.8 (Proste scalanie w miejscu bardzo krótkiego ciągu z długim)
Dane są dodatnie liczby całkowite *k*, n, $k \leq n$, oraz tablica liczb całkowitych *a*[1..*n*] taka, że podtablice *a*[1..*k*] i *a*[*k*+1..*n*] są uporządkowane niemalejąco. Przy założeniu, że $k=O(\sqrt{n})$ zaproponuj algorytm sortowania tablicy (scalania dwóch ciągów uporządkowanych) w miejscu i w czasie O(*n*).
Wskazówka: Zastosuj sortowanie przez wstawianie i skorzystaj z rozwiązania zadania 2.5.

Zadanie 2.9 (Sortowanie blokowe)
Niech *a*[1..*n*] będzie tablicą liczb całkowitych. Przyjmijmy, że $n=r^2$ dla pewnego naturalnego *r*.
Zawartość tablic *a* traktujemy jako zapis *r* rekordów (bloków). Każdy rekord-blok zajmuje spójny fragment tablicy od pozycji $(i-1)*r+1$ do pozycji $i*r$, dla pewnego $i \in [1..r]$. Kluczem w rekordzie jest ostatni element bloku. Zaproponuj sortowanie rekordów względem ich kluczy. Twój algorytm

powinien działać w miejscu i w czasie liniowym. Kolejność elementów w rekordzie nie może ulec zmianie.

Wskazówka: Zastosuj sortowanie przez wybieranie (Selection Sort).

Zadanie 2.10 (Scalanie w miejscu)
Dane są dodatnie liczby całkowite $k \le n$ oraz tablica liczb całkowitych *a*[1..*n*] taka, że podtablice *a*[1..*k*] i a[*k*+1..*n*] są uporządkowane niemalejąco. Zaproponuj algorytm sortowania tablicy (scalania dwóch ciągów uporządkowanych) *a* w miejscu i w czasie O(*n*).

Wskazówka: Rozważmy przypadek, gdy oba scalane podciągi mają długości co najmniej $\sqrt{n}$. Idea algorytmu: podziel sortowane podciągi na bloki o długościach $\sqrt{n}$; posortuj bloki względem ich ostatnich (największych) elementów; potraktuj pierwszy blok (jego elementy) jako bufor; scalaj w pętli dwa kolejne bloki zapisując wynik w buforze i przesuwając bufor w prawo; posortuj bufor i scal go z posortowaną resztą.