Zadanie 3 (25%)
Masz dwie posortowane tablice długości a i b. Podaj algorytm o złożoności obliczeniowej $\mathrm{O}(\log(a) + \log(b))$, który oblicza c-ty najmniejszy element listy stworzonej ze złączenia dwóch wejściowych tablic. Odpowiedź uzasadnij.

Rozwiązanie: Celem tego zadania jest skonstruowanie algorytmu wybierającego c-ty najmniejszy element (nazywany także c-tą statystyką pozycyjną) spośród wartości znajdującej się w dwóch posortowanych tablicach (nazwijmy je A i B) zawierających, odpowiednio, a oraz b elementów. Dodatkowo mamy ograniczenie na złożoność obliczeniową konstruowanej procedury rzędu $\mathrm{O}(\log(a) + \log(b))$.

Na początek zauważmy, że proste rozwiązanie polegające na scaleniu posortowanych tablic A i B w czasie proporcjonalnym do łącznej liczby elementów, tj. $\Theta(a + b)$, a następnie wybranie elementu na pozycji c[1] w czasie stałym nie spełnia podanych ograniczeń na czas działania. Również prosta modyfikacja procedury MERGE, polegająca na przerwaniu jej po przetworzeniu c-tego elementu i zwrócenie jego wartości jest niewystarczająca – jej złożoność jest rzędu $\Theta(c)$, gdzie c może być dowolną liczbą od 1 do $a + b$.

W celu skonstruowania algorytmu o złożoności $\mathrm{O}(\log(a) + \log(b))$ dla rozważanego problemu posłużymy się metodologią *Divide and conquer*. Głowna idea zaproponowanego algorytmu będzie nawiązywała do strategii wykorzystanej w rozwiązaniu zadania 4 z listy 4 (por. opublikowane rozwiązania zadań z listy 4 na ćwiczenia).

Nasz algorytm otrzymuje na wejściu dwie posortowane tablice A oraz B o długościach, odpowiednio, a oraz b, a także liczbę $c \in \{1, \ldots, a + b\}$. W każdym kroku będziemy zawężać „zakres przeszukiwań" przez porównywanie odpowiednich elementów tablic A oraz B oraz „odrzucenie" połowy elementów pozostałych w jednej z tych tablic. Oczywiście interesują nas tylko elementy na pozycjach $i \leqslant c$, zatem jeśli c jest dostatecznie małe ($c < a$ lub $c < b$), możemy przez rozpoczęciem algorytmu odrzucić wszystkie elementy tablic o indeksach $i > c$.

Konstrukcję naszego algorytmu zacznijmy od oczywistych obserwacji, która określą nam warunki brzegowe. Zauważmy mianowicie, że jeśli jedna z wejściowych tablic jest pusta, to szukany element jest c-tym najmniejszym elementem drugiej tablicy i może on zostać zwrócony w czasie $\Theta(1)$ (ponieważ tablice są posortowane). Jeśli obie tablice są niepuste, a jedna z nich (powiedzmy, tablica A) ma długość 1, wystarczy porównać ze sobą elementy $B[c-1]$, $B[c]$ oraz $A[1]$ (o ile istnieją) i zwrócić jeden z nich, w zależności od ich względnego porządku. To również może zostać wykonane w czasie stałym.

Załóżmy zatem, że $a, b > 1$ i oznaczmy $m_a = \lceil a/2 \rceil$ oraz $m_b = \lceil b/2 \rceil$ (m_a oraz m_b są indeksami środkowych elementów obu tablic). Rozważymy teraz dwa przypadki, w zależności od tego, czy $c > m_a + m_b$, czy $c \leqslant m_a + m_b$.

- Przypuśćmy najpierw, że $c \leqslant m_a + m_b$. Jeśli $A[m_a] \geqslant B[m_b]$, możemy wówczas odrzucić wszystkie elementy tablicy A o indeksach $i > m_a$ i rekurencyjnie znaleźć c-ty najmniejszy element w tablicach $B[1 : b]$ oraz $A[1 : m_a]$. Istotnie, zauważmy, że w takim przypadku każdy z $m_a + m_b \geqslant c$ elementów w podtablicach $A[1 : m_a]$ oraz $B[1 : m_b]$ jest nie większy niż elementy w podtablicy $A[(m_a + 1) : a]$. Te możemy zatem pominąć, gdyż szukaną wartością jest c-ty najmniejszy spośród elementów w tablicach $B[1 : b]$ oraz $A[1 : m_a]$. Jeśli $A[m_a] < B[m_b]$, w analogiczny sposób wnioskujemy, że możemy pominąć elementy w podtablicy $B[(m_b + 1) : b]$ i zredukować nasz problem do znalezienia c-tego najmniejszego elementu w tablicach $B[1 : m_b]$ oraz $A[1 : a]$.

- Rozważmy teraz przypadek $c > m_a + m_b$. Zauważmy, że jeśli $A[m_a] \geqslant B[m_b]$, to wówczas wszystkie elementy tablicy B o indeksach $i \leqslant m_b$ są na pewno nie większe od szukanego elementu. Wówczas c-ty najmniejszy element tablic A i B będzie $(c - m_b)$-tym

[1] W rozwiązaniu tablicę o długości L indeksować będziemy kolejnymi liczbami naturalnymi od 1 do L

najmniejszym spośród pozostałych elementów w tablicach $A[1:a]$ oraz $B[(m_b+1):b]$. Analogicznie, jeśli $A[m_a] < B[m_b]$, wystarczy rekurencyjnie znaleźć $(c-m_a)$-ty najmniejszy element w tablicach $A[(m_a+1):a]$ oraz $B[1:b]$.

Poniżej przedstawiamy krótkie podsumowanie otrzymanego algorytmu wyznaczania c-tej najmniejszej wartości z dwóch posortowanych tablic A i B rozmiarów a i b.

1. Jeśli jedna z tablic ma długość 0, zwróć c-ty element drugiej tablicy.
2. Jeśli jedna z tablic (powiedzmy, A) ma tylko jeden element ($a = 1$, $b \geqslant 1$), wówczas:
   (a) jeśli $c = 1$, zwróć $\min\{A[1], B[1]\}$;
   (b) jeśli $c = b+1$, zwróć $\max\{A[1], B[b]\}$;
   (c) jeśli $1 < c < b+1$, zwróć „środkowy" z elementów $B[c-1]$, $B[c]$ oraz $A[1]$.
3. Jeśli $a, b > 1$ oraz $c \leqslant m_a + m_b$, to
   (a) jeśli $A[m_a] \geqslant B[m_b]$, znajdź rekurencyjnie c-ty najmniejszy element w tablicach $B[1:b]$ oraz $A[1:m_a]$;
   (b) jeśli $A[m_a] < B[m_b]$, znajdź rekurencyjnie c-ty najmniejszy element w tablicach $B[1:m_b]$ oraz $A[1:a]$.
4. Jeśli $a, b > 1$ oraz $c > m_a + m_b$, to
   (a) jeśli $A[m_a] \geqslant B[m_b]$, znajdź rekurencyjnie $(c-m_b)$-ty najmniejszy element w tablicach $A[1:a]$ oraz $B[(m_b+1):b]$;
   (b) jeśli $A[m_a] < B[m_b]$, znajdź rekurencyjnie $(c-m_a)$-ty najmniejszy element w tablicach $A[(m_a+1):a]$ oraz $B[1:b]$.

Ponieważ w każdym kroku zmniejszamy rozmiar jednej z dwóch przeglądanych tablic o połowę[2], do znalezienia c-tego najmniejszego elementu potrzebne będzie $\mathrm{O}(\log(a)+\log(b))$ rund algorytmu. Ponadto w każdej rundzie wykonujemy stałą ilość operacji o koszcie $\Theta(1)$. Zauważmy tutaj, że nie ma potrzeby przekopiowywania elementów tablic – wszystkie operacje mogą być wykonywane na ich indeksach. Z tego wnioskujemy, że złożoność obliczeniowa zaproponowanego algorytmu wynosi $\mathrm{O}(\log(a)+\log(b)) = \mathrm{O}(\log(ab))$.

[2]Formalnie powinniśmy uwzględnić kwestię podłóg i sufitów, tj. w uzasadnieniu ograniczenia górnego na czas działania przyjąć, że jeśli rozmiar l tablicy jest nieparzysty, to w kolejnym kroku mamy tablicę o $\lceil l/2 \rceil$ elementach, ale w tym przypadku nie ma to wpływu na asymptotyczną złożoność obliczeniową algorytmu.