Lista 6, zadanie 1

Rekurencyjny quicksort wywołuje się na dwóch podrzielonych tablicach, podczas gdy „zwykły" iteracyjny quicksort symuluje stos wywołań pamiętając przedziały, które zostały nam do posortowania. Aby zrobić to bez dodatkowej pamięci musimy znaleźć sposób na pamiętanie tego bez użycia stosu.

Więc mamy procedurę partition (l, r), która dzieli przedział (l, r) na dwie części - większą od pivota i mniejszą lub równą od pivota, gdzie pivotem może być pierwszy element tablicy. Procedura partition zwraca wartość s, która określa miejsce pivota w tablicy po podziale.

W tym momencie rekurencyjny quicksort wywołałby się na przedziałach $(l, s-1)$ i $(s+1, r)$. Iteracyjny natomiast wrzuciłby parę $s+1, r$ na stos i posortował przedział $(l, s+1)$. Nasz algorytm zrobi podobnie do algorytmu iteracyjnego, z tym, że zamiast wrzucać parę $s+1, r$ na stos zrobi to następująco:

→ znajdujemy maksimum w przedziale $s+1, r$ i zamieniamy je z s. Wtedy do odtworzenia porcji r przeszukujemy tablicę od s w prawo aż znajdziemy element większy od s. Najmniejszy element z tego przedziału ustawiamy na porcji s - to był nasz pivot. $s+1$ mamy zdefiniowane przez zakończenie posortowanego już przedziału, który kończy się na $s-1$. Zatem wersja quicksorta z zadania będzie działać tak samo jak

wersja iteracyjna, z tym że zamiast zwracać predykaty do posortowania na stos, zapisze je w tablicy z danymi. Koszt algorytmu zwiększy się o wyszukiwanie minimum i maksimum w tablicy (przy czym szukanie pierwszego elementu większego od s można wykonać wspólnie z szukaniem minimum). Oba te algorytmy mają taką samą złożoność jak partition - $O(n)$, czyli złożoność zwiększy się jedynie o stałą