Algorytmy i Struktury Danych (2023)

Lista zadań 5 - quick sort, selekcja, sortowanie bez porównań

1. Udowodnij, że jeśli dla pewnego ustalonego q, takiego że $\frac{1}{2} < q < 1$, podczas sortowania szybkiego, procedura `partition`, na każdym poziomie rekurencji podzieli elementy tablicy w stosunku $q : (1 - q)$ to algorytm wykona się w czasie $O(n \log n)$. Wskazówka: udowodnij, że głębokość rekurencji nie przekroczy $-\log n / \log q$ i zaniedbaj błędy zaokrągleń do wartości całkowitych.

2. Ile porównań (zapisz wyniki w notacji O) wykona algorytm `quicksort` z procedurą `partition` w wersji Hoare'a, a ile w wersji z procedurę `partition` w wersji Lomuto dla danych: (a) posortowanych rosnąco, (b) posortowanych malejąco, (c) o identycznych kluczach?

3. Napisz wzór na numer kubełka, do którego należy wrzucić liczbę x w sortowaniu kubełkowym, jeśli kubełków jest n, a elementy tablicy mieszczą się przedziale (a, b). Numeracja zaczyna się od 0.

4. Dla jakich danych sortowanie metodą kubełkową ma złożoność $O(n^2)$?

5. Jak obliczyć k-tą od końca cyfrę w liczby x? Jak obliczyć ilość cyfr liczby x? Przyjmujemy układ dziesiętny. Jak wyniki zmienią się w układzie pozycyjnym gdzie różnych cyfr jest m a ich wartości x należą do przedziału $0 \leq x < m$?

6. Posortuj metodą sortowania pozycyjnego liczby: 101, 345, 103, 333, 432, 132, 543, 651, 791, 532, 987, 910, 643, 641, 12, 342, 498, 987, 965, 322, 121, 431, 350. W pisemnym rozwiązaniu pokaż, jak wygląda zawartość kolejek, za każdym razem, gdy tablica wyjściowa jest pusta i wszystkie liczby znajdują się w kolejkach, oraz jak wygląda tablica wyjściowa, za każdym razem, gdy sortowanie ze względu na kolejną cyfrę jest już zakończone.

7. (2pkt) Które z procedur sortujących:
   (a) `insertion_sort` (przez wstawianie),
   (b) `quick_sort` (szybkie),
   (c) `heap_sort` (przez kopcowanie),
   (d) `merge_sort` (przez złączanie),
   (e) `counting_sort` (przez zliczanie)
   (f) `radix_sort` (pozycyjne),
   (g) `bucket_sort` (kubełkowe)
   są stabilne? W każdym przypadku uzasadnij stabilność lub znajdź konkretny przykład danych, dla których algorytm nie zachowa się stabilnie.

8. Napisz funkcję `void counting_sort(node* lista, int m);` sortującą przez zliczanie listę linkowaną liczb całkowitych nieujemnych mniejszych od m. Procedura nie powinna usuwać ani tworzyć nowych węzłów, tylko sprytnie zmieniać pola `next` wykorzystując tylko $O(m)$ dodatkowej pamięci na wskaźniki.

9. (algorytm Hoare'a) Korzystając funkcji `int partition(int t[], int n)` znanej z algorytmu sortowania szybkiego napisz funkcję `int kty(int t[], int n)`, której wynikiem będzie k-ty co do wielkości element początkowo nieposortowanej tablicy `t`. Średnia złożoność Twojego algorytmu powinna wynieść $O(n)$.