TRIEDENIA

Dorobte projekt **Triedenia**, ktorý je zameraný na utriedenie jednorozmerného poľa.

ÚLOHA 02

Definujte `Kresli(tab, f)`, ktorá nakreslí pole `tab` zdanou farbou `f`.

ÚLOHA 03

Definujte `vymen(tab, i, j)`, ktorá v `tab` vymení dva prvky (aj vizuálne).

ÚLOHA 04

Naprogramujte `BubbleSort` (bublinkové triedenie).

Porovnávanie dvoch prvkov vedľa seba - do prvku poľa s menším indexom ide menší z nich. Po jednom prechode poľom sa určite maximálny prvok dostane na koniec poľa. Potom ostáva utriediť N-1 prvkov poľa (posledný je už na svojom mieste).

ÚLOHA 05

Naprogramujte `MinSort`.

Nájde v úseku poľa minimálny prvok, vymení ho s prvým prvkom úseku a posunie dolnú hranicu triedeného úseku o 1 (t.j. triedený úsek poľa sa skráti o 1), atď.

ÚLOHA 06

Naprogramujte `MinMaxSort`.

Nájde v úseku poľa minimálny aj maximálny prvok. Minimálny vymení s prvým prvkom úseku a posunie dolnú hranicu triedeného úseku o 1, maximálny vymení s posledným prvkom úseku a posunie hornú hranicu triedeného úseku o 1 (t.j. triedený úsek poľa sa skráti o 2), atď.

ÚLOHA 07

Naprogramujte `InsertSort` (triedenie vsúvaním).

Istá časť poľa je už utriedená a do nej zaradí nový prvok na správne miesto a predĺži utriedený úsek o 1.

ÚLOHA 08

Naprogramujte `MergeSort` (metóda rozdeľuj a panuj).

Pole rozdelíme na dve približne rovnako veľké časti. Potom sa zaoberáme s každou z týchto dvoch častí zvlášť a to takým istým spôsobom, t.j. rozdelíme ju na dve časti, atď., ... až kým nedostaneme jednoprvkové úseky. Je zrejmé, že jedna položka je utriedená a teda máme dve utriedené časti. Teraz použijeme druhú časť algoritmu - zlúčenie dvoch utriedených častí tak, aby aj novovzniknutá časť bola utriedená, t.j. dostávame časť s dvoma položkami. Podobne sa rozdelí a zlúči aj druhá časť z rozdelenia a dostávame utriedenú druhú dvojprvkovú časť. ... Algoritmus sa bude opakovať, až kým nebude utriedené celé pole.

ÚLOHA 09

Naprogramujte `QuickSort`.

Vytvára dve časti poľa tak, aby ich nebolo treba zlučovať, ale aby ich stačilo iba položiť za seba. V prvom kroku sa jeden prvok vyberie za pivota (napr. prvý prvok z danej postupnosti). Podľa tohto pivota rozdeľme vstupnú postupnosť prvkov na dve časti, pričom v prvej z nich budú čísla menšie ako pivot a v druhej čísla väčšie a rovné ako pivot (v najlepšom z prípadov sa postupnosť rozdelí na dve časti s približne rovnakým počtom prvkov, v najhoršom prípade však zrejme na nula prvkov, ktoré bude tvoriť prvú časť a na všetky ostatné prvky). Po takomto rozdelení postupnosti dostávame tri časti - časť s jediným prvkom, ktorým je pivot, časť, v ktorej sú prvky menšie ako pivot a časť s prvkami väčšími alebo rovnými ako je pivot. Potom rovnakým spôsobom utriedime druhú časť (prvky menšie ako pivot) a tretiu časť (prvky väčšie ako pivot). Po ich utriedení druhú a tretiu časť "položíme" vedľa seba, pričom pivot dáme na správne miesto.

ÚLOHA 10

Naprogramujte `HeapSort` (triedenie haldou).

Halda je dátová štruktúra, ktorá má tvar "skoro" úplného binárneho stromu, len na poslednej úrovni binárneho stromu môžu chýbať synovia (vrcholov predposlednej úrovne) a to tak, že ak chýba nejaký syn tak budú chýbať aj všetci synovia vpravo na najnižšej úrovni. Pre všetky vrcholy stromu platí, že otec má väčšiu (alebo rovnú) hodnotu ako jeho synovia - ak existujú. Zrejme v koreni haldy je vždy maximálny prvok. Haldu budeme reprezentovať v jednorozmernom poli indexovanom od 0 tak, že koreň stromu je na indexe 0 a i-ty vrchol má synov na indexoch 2*i+1 a 2*i+2.

Algoritmus má 2 fázy. Vytvorenie haldy, v halde je koreň (t.j. prvý prvok poľa) najväčší prvok zo všetkých, jeho výmena s posledným prvkom (ešte neutriedeného) poľa a nové "uhaldovanie", t.j. oprava haldy. Zakaždým pracujeme s o 1 kratším poľom – haldou, teda na jeho konci sa postupne sústreďujú najväčšie prvky.