Metody sortowania: sortowanie szybkie

Algorytm sortowania, który zostanie omówiony na bieżących zajęciach, nosi nazwę „*sortowanie szybkie*“, po angielsku „*quicksort*“. Jest on znacznie szybszy od omówionych algorytmów sortowania, jego złożoność obliczeniowa wynosi *O(NlogN)*, w przeciwieństwie do wcześniej poznanego sortowania bąbelkowego oraz sortowania przez wstawianie, których złożoność obliczeniowa wynosi $O(N^2)$. Dla dużych zbiorów danych do posortowania złożoność $O(N^2)$ to zdecydowanie za dużo – czas sortowania będzie rósł proporcjonalnie do kwadratu ilości sortowanych elementów. Algorytm *quicksort* używany jest w sytuacjach, gdy sortuje się duże ilości danych – gdy danych jest niewiele, proste algorytmy sortowania są efektywniejsze! Jako granicę sensowności stosowania sortowania algorytmem *quicksort* przyjmuje się zwykle ilość sortowanych elementów, wynoszącą minimum 10.

Na czym polega algorytm *quicksort*? Wyobraźmy sobie posortowaną rosnąco tablicę z liczbami. Wybierając dowolną komórkę w tablicy możemy być pewni, że jeśli istnieją elementy tablicy o mniejszych indeksach od indeksu wybranej komórki, to zawierają wartości mniejsze lub równe wartości w wybranej komórce. Analogiczne rozumowanie można przeprowadzić dla elementów tablicy o indeksach większych od indeksu elementu wybranego. Te rozważania powinny przybliżyć ideę algorytmu *quicksort*, który przedstawia się następująco:

Jeżeli tablica zawiera co najmniej dwa elementy, wykonaj następujące czynności:

1. Dla tablicy wybierz *element osiowy* (*ang. pivot*).
2. *Podziel tablicę na dwie części*. W jednej z nich umieść elementy tablicy o *wartościach mniejszych lub równych elementowi osiowemu*, a w drugiej *elementy większe* od elementu osiowego. Dla sortowania rosnącego elementy mniejsze lub równe elementowi osiowemu powinny znaleźć się w komórkach tablicy o indeksach mniejszych od indeksów tablicy z elementami większymi od elementu osiowego, dla sortowania malejącego odwrotnie.
3. *Dla każdej z* wydzielonych *dwóch części* tablicy z osobna zrealizuj rekurencyjnie algorytm.

Z powyższego rozumowania jasno wynika, że algorytm *quicksort* jest algorytmem rekurencyjnym. Jak widać z powyższych punktów, jest on dość ogólnie określony – poszczególne implementacje mogą różnić się sposobem realizacji podziału tablicy oraz doboru elementu osiowego. Ponadto, realizując algorytm powinniśmy zadbać o to, aby element osiowy w wyniku podziału tablicy na części znalazł się na ostatniej pozycji (największy indeks) w części tablicy z elementami o wartościach mniejszych lub równych elementowi osiowemu przy sortowaniu rosnącym (odpowiednio przy sortowaniu malejącym będzie to część zawierająca elementy większe lub równe elementowi osiowemu), można i należy wtedy element osiowy pominąć w procesie rekurencyjnego sortowania podtablic – znajduje się on na właściwym miejscu w tablicy.

Przeanalizujmy poniższy listing (plik *q-sort.cpp*):

```
#include<iostream>
using namespace std;

void qsort(float tablica[], unsigned poczatek, unsigned koniec);
void zamiana(float tablica[], unsigned e1, unsigned e2);
void wypisz(float tablica[], unsigned ilosc);

int main()
{
float dane[] = {0, 12.5, 7, 13.33, 25.23, 10, 9.99, 0.25, 0.98, 1.01};

   cout<<"Program posortuje tablice z danymi metoda \"quicksort\""<<endl;
   cout<<"Przed sortowaniem tablica zawiera dane:"<<endl;
   wypisz(dane, 10);
   cout<<endl;
```

```
   cout<<endl<<"Sortujemy tablice. Tablica zawiera obecnie dane:"<<endl;
   qsort(dane, 0, 9);
   wypisz(dane, 10);
   cout<<endl;
}

void qsort(float tablica[], unsigned poczatek, unsigned koniec)
{

   // Jesli tablica ma wiecej niz jeden element
   if( poczatek<koniec )
   {
   unsigned        granica,            // pokazuje pierwszy element "duzy" w tablicy
           indeks;  // sluzy do przegladania tablicy

// Dokonanie przeniesienia elementow "duzych" i "malych" do odpowiednich miejsc

           // Element osiowy umieszczamy w poczatkowej komorce
           // wybor: os przypada w srodku tablicy
           zamiana(tablica, poczatek, (poczatek+koniec)/2);

           // Elementy "duze" nie wczesniej jak w drugim elemencie zakresu tablicy
           granica = poczatek+1;

       // Wyszukujemy pierwszy element "duzy"
       while( (granica<=koniec)&&(tablica[granica]<=tablica[poczatek]) )
                       granica++;

       indeks = granica+1;
       while(indeks<=koniec)
       {
               if( tablica[indeks]<=tablica[poczatek] )
               {
                  // Dopisz znaleziony do elementow "malych"
                  zamiana(tablica, granica, indeks);
                  granica++;
               }

               // Przejdz na nastepny element przegladany
               indeks++;
           }

           // Umiesc zapamietany element osiowy jako ostatni z "malych"
           // Jesli elementow "malych" jest wiecej niz 1, zamien
           // (jesli jest tylko jeden, jest to wlasnie element osiowy)
           if( granica-1>poczatek ) zamiana(tablica, poczatek, granica-1);

// Rekurencyjne wywolanie dla obu czesci tablicy (bez elementu osiowego)

           // Jesli "malych" elementow jest co najmniej 3 (osiowy bedzie pominiety)
           // wywolaj qsort dla elementow "malych" bez osiowego
           if( granica-poczatek >= 3) qsort(tablica, poczatek, granica-2);

           // Jesli "duzych" elementow jest co najmniej dwa
           // wywolaj qsort dla elementow "duzych"
           if( koniec-granica >= 2) qsort(tablica, granica, koniec);
   }
}

void zamiana(float tablica[], unsigned e1, unsigned e2)
{
float tymczasowy;

   if (e1 == e2) return;
```

```
   tymczasowy = tablica[e1];
   tablica[e1] = tablica[e2];
   tablica[e2] = tymczasowy;
}

void wypisz(float tablica[], unsigned ilosc)
{
unsigned indeks;

   for(indeks=0; indeks<ilosc; indeks++) cout<< tablica[indeks] <<" ";
}
```

Powyższy listing realizuje sortowanie rosnące. Stosuje następującą metodę podziału tablicy na dwie podtablice:

- na początku element osiowy (założono, że elementem osiowym jest element „środkowy“) zostaje zamieniony z początkowym elementem tablicy miejscami, od tej pory element osiowy znajduje się w pierwszym elemencie sortowanej tablicy.
- funkcja sortująca używa dwóch zmiennych indeksujących tablicę: *granica* służy do wskazywania pierwszej pozycji, na której znajdują się elementy większe od osiowego („duże”), *indeks* to zmienna, służąca do przeglądania tablicy przy wykonywaniu zamian. Po umieszczeniu elementu osiowego na początkowej pozycji w tablicy, element następny jest pierwszym, który mógłby być „dolną granicą“ dla elementów „dużych“ w tablicy.
- kolejnym krokiem jest znalezienie pierwszego elementu „dużego“ w tablicy, licząc od jej początku. Po wykonaniu tego kroku, elementy tablicy o indeksach mniejszych niż „granica“ są mniejsze lub równe elementowi osiowemu (nazywam je „małe“).
- element w komórce tablicy o indeksie *granica* jest pierwszym elementem, którego wartość przekracza wartość elementu osiowego. Należy teraz dla elementów o indeksach od *granica+1* do końca tablicy dokonać zaszeregowania wartości do odpowiedniej części tablicy. Elementy te wystarczy przejrzeć, idąc w stronę rosnących indeksów tablicy i dla każdego z nich sprawdzić, czy jest większy od elementu osiowego (zapamiętanego obecnie w pierwszym elemencie tablicy). Jeśli jest większy, to nie należy robić nic, bo znajduje się we właściwej części tablicy. Jeśli jest jednak mniejszy lub równy elementowi osiowemu, należy zamienić go miejscami z elementem wskazywanym przez zmienną *granica* oraz zwiększyć wartość zmiennej *granica* o 1 – obszar tablicy zawierający wartości „małe“ zwiększył się o jeden element.
- Ostatnim krokiem jest zamiana miejscami elementów tablicy: pierwszego i poprzedzającego wskazywany przez zmienną *granica* – chcemy, aby element osiowy był ostatnim elementem znajdującym się w części tablicy zawierającej wartości „małe“.

Po tak przeprowadzonej obróbce tablicy, dla obu jej części realizowane są wywołania rekurencyjne funkcji sortującej, o ile są one sensowne tj. jeśli są co najmniej trzy elementy „małe“ (trzy, bo element osiowy zostanie pominięty), oraz co najmniej dwa elementy „duże“.