***Sortowanie***

Sortowanie danych (inaczej: porządkowanie wg ustalonych kryteriów) jest jednym z podstawowych problemów, z którym prędzej czy później spotka się każdy. Poniżej przedstawiamy tylko nieliczne dziedziny, w których występuje potrzeba sortowania danych:

* sport - wyniki uzyskane przez poszczególnych zawodników należy ułożyć w określonej kolejności, aby wyłonić zwycięzcę oraz podać lokatę każdego zawodnika.
* bank - spłaty kredytów należy ułożyć w odpowiedniej kolejności, aby wiadomo było kto i kiedy ma płacić odsetki do banku.
* grafika - wiele algorytmów graficznych wymaga porządkowania elementów, np. ścian obiektów ze względu na odległość od obserwatora. Uporządkowanie takie pozwala później określić, które ze ścian są zakrywane przez inne ściany dając w efekcie obraz trójwymiarowy.
* bazy danych - informacja przechowywana w bazie danych może wymagać różnego rodzaju uporządkowania, np. lista książek może być alfabetycznie porządkowana wg autorów lub tytułów, co znacznie ułatwia znalezienie określonej pozycji.

***Algorytmy sortowania***

Opracowano bardzo wiele algorytmów sortowania i wiele z nich występuje w wielu odmianach. Spis można zobaczyć tutaj: <http://eduinf.waw.pl/inf/alg/003_sort/index.php>

**Sortowanie bąbelkowe** - jest jednym z najstarszych algorytmów sortujących. Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i [zamianie ich kolejności](http://edu.i-lo.tarnow.pl/inf/alg/003_sort/0002.php#Losowe) w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

***Przykład*** Wykonaj rosnące sortowanie bąbelkowe 5-cio elementowej tablicy [7,1,5,3,0].

Poniższa funkcja wykonuje sortowanie bąbelkowe tablicy tab mającej n elementów.

void sortowanie\_babelkowe(int tab[], int n)

{

for(int i=0; i<n-1; i++)

{

for(int j=0; j<n-1; j++)

{

if(tab[j+1]<tab[j])

{

int bufor;

bufor=tab[j+1];

tab[j+1]=tab[j];

tab[j]=bufor;

}

}//koniec pętli wewnętrznej

}//koniec pętli zewnętrznej

}

***Zadanie 1*** Napisz program, który wylosuje i umieści w tablicy n liczb całkowitych losowanych z przedziału <1;100000> oraz posortuje tablicę rosnąco algorytmem sortowania bąbelkowego.

***Zadanie 2*** Istnieje wiele modyfikacji sortowania bąbelkowego. Jedna z nich polega na zmianie pętli wewnętrznej: for(int j=0; j<n-1-i; j++). Co daje taka modyfikacja?.

**Sortowanie szybkie** - algorytm opiera się na strategii "dziel i zwyciężaj", którą możemy krótko scharakteryzować w trzech punktach:

DZIEL - problem główny zostaje podzielony na podproblemy

ZWYCIĘŻAJ - znajdujemy rozwiązanie podproblemów

POŁĄCZ - rozwiązania podproblemów zostają połączone w rozwiązanie problemu głównego

Idea sortowania szybkiego jest następująca:

DZIEL : najpierw sortowany zbiór dzielimy na dwie części w taki sposób, aby wszystkie elementy leżące w pierwszej części (zwanej lewą partycją) były mniejsze lub równe od wszystkich elementów drugiej części zbioru (zwanej prawą partycją).

ZWYCIĘŻAJ : każdą z partycji sortujemy rekurencyjnie tym samym algorytmem.

POŁĄCZ : połączenie tych dwóch partycji w jeden zbiór daje w wyniku zbiór posortowany.

void quicksort(int tablica[], int lewy, int prawy)

{

int v=tablica[(lewy+prawy)/2];

int i,j,x;

i=lewy;

j=prawy;

do

{

while(tablica[i]<v) i++;

while(tablica[j]>v) j--;

if(i<=j)

{

x=tablica[i];

tablica[i]=tablica[j];

tablica[j]=x;

i++;

j--;

}

}

while(i<=j);

if(j>lewy) quicksort(tablica,lewy, j);

if(i<prawy) quicksort(tablica, i, prawy);

}

***Zadanie 3*** Wstaw powyższą funkcję do programu i przetestuj ją. **Uwaga!** Algorytm sortowania szybkiego wywołujemy podając za *lewy* indeks pierwszego elementu tablicy, a za *prawy* indeks elementu ostatniego.

Zasadę działania algorytmu bąbelkowego oraz szybkiego w zabawny sposób przedstawiają filmy na: <http://www.wykop.pl/link/1332207/animacje-algorytmow-sortowania/>

**Porównanie działania sortowania bąbelkowego oraz szybkiego**

***Przykład*** Wykonaj porównanie algorytmów wg filmu:

<http://miroslawzelent.pl/kurs-c++/sortowanie-zlozonosc-algorytmow/>

**Złożoność algorytmów**

Złożoność dzielimy na:

* pamięciową
* Algorytmy sortujące w miejscu (*łac. in situ*) - wymagają stałej liczby np. tablic lub innych zmiennych do wykonania sortowania.
* Algorytmy nie sortujące w miejscu - wymagają zarezerwowania w pamięci dodatkowych zmiennych. Tego typu algorytmy są zwykle bardzo szybkie w działaniu, jednakże okupione to jest dodatkowymi wymaganiami na pamięć.
* czasową - określa ona szacunkowo czas wykonywania algorytmu w zależności od liczby danych wejściowych. Złożoność tę charakteryzujemy przy pomocy tzw. notacji *O*.

|  |  |
| --- | --- |
| *O(n2)* | Algorytm, w którym czas wykonania rośnie z kwadratem liczby przetwarzanych elementów. Dwukrotny wzrost liczby danych powoduje czterokrotny wzrost czasu wykonania. |
| *O(n\*log n)* | Dobre algorytmy sortujące mają taką właśnie złożoność obliczeniową. Czas wykonania przyrasta dużo wolniej od wzrostu kwadratowego. Tego typu złożoność powstaje, gdy zadanie dla n elementów można rozłożyć na dwa zadania zawierające po połowie elementów. |

W lepszym zrozumieniu tego zagadnienia pomoże:

<http://eduinf.waw.pl/inf/alg/003_sort/0001.php#czasowa_zlozonosc>