Sortowanie przez wstawianie

Sortowanie przez wstawianie (funkcjonuje też nazwa umieszczanie) jest **metodą porządkowania zbioru o złożoności kwadratowej**. Żeby zrozumieć istotę jego działania wystarczy uzmysłowić sobie sposób, w jaki szeregujemy karty w dłoni (każdą kolejną kartę wstawiamy w odpowiednie miejsce w wachlarzu posortowanych kart). Porównując ten algorytm z bąbelkowym można go nazwać bardziej inteligentnym, gdyż ilość operacji jest zależna od ułożenia wartości w ciągu na wejściu (jest znacznie bardziej wydajny dla danych wstępnie posortowanych).

Zasada działania

W algorytmie **sortowania przez wstawianie** ciąg danych jest niejako sztucznie podzielony na dwie części: uporządkowaną (przed rozpoczęciem działania algorytmu ta część zawiera jeden element), nieuporządkowaną (zawiera wszystkie pozostałe elementy).

Sposób porządkowania można w skrócie opisać następująco:
- pobierz pierwszą z brzegu wartość z części nieuporządkowanej (jeśli ta część jest pusta to zakończ działanie algorytmu),
- wstaw ją we właściwe miejsce pomiędzy elementy z części posortowanej (po takiej operacji część nieuporządkowana jest jeden element krótsza, a część posortowana jeden element zyskuje).

Pozostaje jedynie do rozstrzygnięcia, **w jaki sposób technicznie wyznaczyć właściwe miejsce** nowej wartości w części uporządkowanej.
Najprościej zrobić to porównując wartość elementu wstawianego z kolejnymi elementami w części wcześniej uporządkowanej (począwszy od ostatniego). Jeżeli element na pozycji *i* jest większy bądź równy wstawianemu, to ten nowy element należy umieścić tuż za nim.

Implementacja

```
    for (int i = 1; i < n; i++)   //a[0] to jednoelementowy ciąg posortowany
    {
        taken= a[i]; //pierwszy z brzegu zostaje pobrany do umieszczenia
        int k=i-1;
        while( k>=0 && a[k] > take)
          {
            a[k + 1] = a[k]; //przesuwamy wszystkie o 1 w prawo,tu będzie nowy
            k--;
          }
        a[k+1] = taken; //umieszczenie wartości we właściwym miejscu
    }
```

Sortowanie przez scalanie

Merge sort, czyli sortowanie przez scalanie to algorytm sortowania danych, stosujący metodę dziel i zwyciężaj.

Opis działania algorytmu zamyka się w trzech krokach:

- Podziel dane na dwie równe części,
- Zastosuj sortowanie przez scalanie dla każdej z nich oddzielnie, chyba że pozostał już tylko jeden element,
- Scal posortowane podciągi w jeden.

Sama procedura scalania dwóch ciągów w jeden wygląda następująco:

Dane wejściowe:

d[lewy..sr-1,sr..prawy] - tablica zawierająca dwa ciągi przygotowane do scalenia (jeden ciąg zajmuje indeksy od lewy do sr-1, a drugi ciąg zajmuje indeksy od sr do prawy)

Oczekiwany wynik:

d[lewy..prawy] - docelowa tablica zawierająca posortowany ciąg (scalony)

Dla ułatwienia, wartości scalimy najpierw do tablicy pomocniczej t[lewy..prawy], a po zakończeniu scalania przepiszemy wszystko do tablicy d.

Rozwiązanie:

- Utwórz zmienne wskazujące na początki scalanych ciągów i=lewy, j=sr
- Jeżeli ciąg lewy wyczerpany (i>=sr), dołącz pozostałe elementy ciągu lewego do wynikowego oraz zakończ pracę.
- Jeżeli ciąg prawy wyczerpany (j>prawy), dołącz pozostałe elementy ciągu prawego do wynikowego i zakończ pracę.
- Jeżeli d[i] ≤ d[j] dołącz d[i] do t i zwiększ i o jeden, w przeciwnym przypadku dołącz d[j] do t i zwiększ j o jeden
- Powtarzaj od kroku 2 aż wszystkie wartości z d trafią do t

Teraz można przepisać scalony ciąg z tablicy t do tablicy d.

Implementacja

```
int d[100],ile; //tablica dana w specyfikacji
void sortuj(int lewy, int prawy)
{
  int t[100];
  int sr,i1,i2,i;
  //rekurencyjny podział na dwie częśći
  sr = (lewy + prawy + 1) / 2;
  if(sr - lewy > 1) sortuj(lewy, sr - 1);
  if(prawy - sr > 0) sortuj(sr, prawy);
  i1=lewy;i2=sr;i=lewy;
  //scalanie
  i1 = lewy; i2 = sr;
  for(i = lewy; i <= prawy; i++)
    if (i1==sr || (i2<=prawy && d[i2]<d[i1]))
        t[i]=d[i2++];
    else
        t[i]=d[i1++];                ]

  for(int i = lewy; i <= prawy; i++) d[i] = t[i];
}
```

Sortowanie szybkie

Z tablicy wybieramy element rozdzielający (tzw pivot), za pomocą tego elementu tablica zostaje dzielona na dwa fragmenty: do lewej przenoszone są wszystkie wartości nie większe od pivota, do prawego wszystkie większe. Następnie sortujemy osobno obie części (prawą i lewą) tą samą metodą. Rekurencja kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa tablica nie wymaga sortowania.

Złożoność algorytmu zależy od sposobu wyboru pivota. Jeśli uda się tak wybrać pivota, że podziały są zrównoważone, algorytm jest tak szybki jak sortowanie przez scalanie czyli O(n·logn); w przeciwnym przypadku może działać tak wolno jak sortowanie przez wstawianie (O(n^2)). Średni czas działania przy losowym wyborze elementu rozdzielającego, dorównuje przypadkowi optymistycznemu.

Poniżej pseudokod algorytmu:

```
Partition(T,l,p)
      x←T[l]
      i←l-1
      j←p+1
      while True do
            repeat j←j-1  until T[j]<=x
            repeat i←i+1 until T[i]>=x
            if i<j then T[i]↔T[j]
            else return j

QuickSort(T,l,p)
      If l<p then
            q←Partition(T,l,p)
            QuickSort(T,l,q)
            QuickSort(T,q+1,p)
```

Kliknij by zobaczyć interpretację taneczną algorytmu ;)