Zliczam, więc sortuję

#czas_liniowy
#count_sort

W jaki sposób najczęściej sortujemy tablicę czy wektor? Korzystamy z gotowej procedury `sort()` z biblioteki STL, a co najwyżej wyposażamy ją we własnoręcznie napisany komparator (pisaliśmy o tym w podrozdziałach *Drużyna, w szeregu zbiórka!* oraz *Kto lepszy?*). Złożoność takiego sortowania zwykle kształtuje się na poziomie $n \log_2 n$ operacji*, gdzie n oznacza rozmiar danych. To dobra złożoność i wystarcza ona w znakomitej większości zastosowań. Czasem jednak pojawia się pytanie: czy nie można posortować danych szybciej, w *czasie liniowym*, czyli proporcjonalnym do rozmiaru danych (bez tego logarytmu). Otóż zdarzają się sytuacje, kiedy jest to możliwe.

Wyobraźmy sobie, że mamy dane wzrosty (w centymetrach) w pewnej grupie *człowieków*. Wzrost każdego człowieka niech będzie liczbą całkowitą. Możemy chyba spokojnie założyć, że dla w miarę normalnych człowieków ich wzrost nie przekracza 250 cm.†

Nasz program powinien przeczytać liczbę n (ilość człowieków), a następnie wczytać kolejno wzrosty kolejnych osób umieszczając je w wektorze `height`. *On y va!*:

```
int main()
{
  int n;
  cin >> n;
  vector<int> height(n);
  for(int i = 0; i < n; i++)
    cin >> height[i];
  . . .
```

Ponieważ przy deklaracji wektora podaliśmy jego rozmiar (n), zatem możemy wygodnie i bezpiecznie wczytywać jego kolejne elementy według ich numerów/indeksów.

Teraz zrobimy następujący trick: zliczymy, ile jest wzrostów o danej wartości. Posłuży nam do tego pomocniczy wektor liczników o nazwie `count`, który przy deklaracji zostanie (automatycznie) wypełniony zerami:

```
const int M = 250;
vector<int> count(M + 1);
```

*Być może należy przemnożyć ten wzór przez pewną stałą i ewentualnie dodać jakieś mniej znaczące wyrażenie.

†Na'vi mogą być sporo wyżsi, ale na razie na Pandorę się nie wybieramy.

Pamiętamy, że wzrosty (teoretycznie) mogą sięgnąć wartości 250 cm (reprezentowanej przez stałą M), stąd taki rozmiar wektora.

No to zliczamy (jeśli napotkamy wzrost o wartości w, powiększamy element wektora $count[w]$):

```
for(int i = 0; i < n; i++)
  count[ height[i] ]++;
```

Teraz, po zakończeniu tej pętli, wektor `count` zawiera wszelkie informacje o strukturze wzrostów całej grupy człowieków. Możemy więc, przeglądając ten wektor, odtworzyć wszystkie wzrosty – i to w kolejności niemalejącej.

Spróbujmy najpierw wypisać je na ekranie. Umówimy się tak: zmienna w będzie oznaczać wzrost człowieka, a zmienna k będzie potrzebna, gdy będziemy mieć więcej osób o tym samym wzroście. No to próbujemy:

```
for(int w = 0; w <= M; w++)
  if(count[w] > 0)
    for(int k = 0; k < count[w]; k++)
      cout << w << " ";
cout << endl;
```

To, że trzeba wypisać wartość zmiennej w (wzrost), to chyba jasne (nieprawdaż?). Ale ta instrukcja warunkowa jest potrzebna? Nie, bo jeśli $count[w]$ jest równe zeru, wtedy po prostu wewnętrzna pętla `for` nie obejdzie ani razu – można więc `if`-a spokojnie opuścić.

Jednak jest jeszcze jeden drobiazg. Powyższy kod działa poprawnie i wyświetla na ekranie dobry wynik. Jeśli dodamy do niego stosowny nagłówek i obudujemy go `main()`-em, wtedy możemy się spodziewać, że po wpisaniu przykładowych danych:

```
10
167 178 161 178 198 177 168 170 178 167
```

otrzymamy listę posortowanych wzrostów:

```
161 167 167 168 170 177 178 178 178 198
```

Cóż jest więc nie tak? Proszę zauważyć, oryginalny wektor `height` nie uległ posortowaniu, tylko na ekranie widzimy rezultaty tej operacji. Posortowane wzrosty trzeba na powrót wpisać do wektora, aby dopełnić formalności. Wystarczy nieco zmodyfikować pętlę wypisującą, dodając zmienną i oznaczającą numer człowieka, który ma dany wzrost:

```
for(int i = 0, w = 0; w <= M; w++)
  for(int k = 0; k < count[w]; k++)
    height[i++] = w;
```

Użycie zmiennej z operatorem inkrementacji już kiedyś wyjaśniliśmy, prawda?

Oczywiście, żeby nacieszyć się posortowaną listą wzrostów, trzeba dodać specjalną pętlę, co zrobimy prezentując pełną wersję programu z tego przykładu:

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

int main()
{
  int n;
  cin >> n;
  vector<int> height(n);
  for(int i = 0; i < n; i++)
    cin >> height[i];
  const int M = 250;
  vector<int> count(M + 1);
  for(int i = 0; i < n; i++)
    count[ height[i] ]++;
  for(int i = 0, w = 0; w <= M; w++)
    for(int k = 0; k < count[w]; k++)
      height[i++] = w;
  for(int i = 0; i < n; i++)
    cout << height[i] << " ";
  cout << endl;
}
```

Dwie uwagi, na koniec. Po pierwsze, zliczanie wzrostów można wykonywać podczas wczytywania danych – na konkursach informatycznych tak postępuje się najczęściej (co się da, robi się przy wczytywaniu danych). Po drugie, dlaczego ten sposób sortowania jest wyjątkowy i kiedy można go zastosować? Tak, ten sposób jest wyjątkowy, bo mamy tu pętle pracujące w czasie liniowym: wykonują mniej więcej n lub M operacji. A metodę tę można zastosować, gdy dane pochodzą ze stosunkowo niezbyt okazałego zbioru (tutaj o rozmiarze $M + 1$) i do tego są *dyskretne*, to znaczy na przykład, że są to liczby całkowite. Z danymi typu `double` nie poszłoby nam tak łatwo.