# Instrukcja do laboratorium 1 z ZBP

## 1 Algorytmy sortowania

Porównać wydajność algorytmów sortowania:

- `qsort` — z języka C
- `sort`
- `stable_sort`
- `sort` dla listy

Do sortowania wykorzystać słownik `http:\\sun.aei.polsl.pl\~sdeor\students\zbp\lab1.dic`

## 2 Sortowanie — obiekty funkcyjne

Stworzyć obiekt funkcyjny pozwalający na sortowanie (za pomocą algorytmu `sort`) słów w porządku *a tergo* (słowa czytane od tyłu).

## 3 Algorytmy generowania kombinacji

W bibliotece STL istnieją algorytmy generowania permutacji (`next_permutation`, `prev_permutation`) w porządku leksykograficznym. Brakuje jednak algorytmów generowania kombinacji $k$-elementowych zbioru $n$-elementowego. Należy stworzyć algorytm:

```
template<class BidirectionalIterator>
   bool next_combination(
      BidirectionalIterator first1, BidirectionalIterator last1,
      BidirectionalIterator first2, BidirectionalIterator last2);
```

Zakres [`first1`, `last1`) określa $n$-elementowy zbiór elementów, z którego należy generować (`last2`−`first2`)-elementowe kombinacje w porządku leksykograficznym. Jeśli kombinacja istnieje, to algorytm powinien zwracać `true`, a w przeciwnym przypadku — `false`. Algorytm powinien dać się wykorzystać w poniższym kodzie:

```
char tab[10] = {'A', 'B', 'C', 'E', 'G', 'I', 'M', 'O', 'P', 'Y'};
vector<char> vi(tab, tab+10);
vector<char> vc(tab, tab+7);

do {
   ShowCollection(vc);
} while (next_combination(vi.begin(), vi.end(), vc.begin(), vc.end());
```

Sposób generowania kolejnej (w porządku leksykograficznym) kombinacji jest bardzo prosty i można go zilustrować na przykładzie:

```
123, 124, 125, 126,
134, 135, 136,
145, 146,
156,
234, 235, 236,
245, 246,
256,
345, 346,
356,
456
```

Formalnie można go zapisać następująco:

1. Początkowa kombinacja to $(1, 2, \ldots, k)$.
2. Kolejną kombinację tworzymy z bieżącej w następujący sposób:
   2.1 Przeglądamy bieżącą kombinację od prawej do lewej strony. Znajdujemy element, który jeszcze nie osiągnął maksymalnej wartości.
   2.2 Element ten zwiększamy o 1, a wszystkie elementy po jego prawej stronie otrzymują kolejno najmniejsze z dopuszczalnych wartości.

## 4 Rozwiązywanie łamigłówek*

Wykorzystać algorytmy `next_combination` i `next_permutation` do stworzenia programu rozwiązującego łamigłówki podobne do poniższej.

```
ŻONA
ŻONA
ŻONA
ŻONA
HAREM
```

Rozwiązaniem jest znalezienie takiego przyporządkowania literom cyfr, żeby suma wszystkich liczb z początkowych wierszy (z wyjątkiem ostatniego) była zgodna z liczbą z ostatniego wiersza. Każda cyfra może być przyporządkowana tylko jednej literze.
Algorytm powinien działać według następującego schematu:

1. Wyszukuje wszystkie $k$ różne litery występujące w zadaniu.
2. Generuje wszystkie $k$ elementowe kombinacje zbioru 10 cyfr i dla każdej z nich generuje jej wszystkie możliwe permutacje sprawdzając czy nie stanowią one rozwiązania zadania.

Powinien on też w maksymalnym stopniu wykorzystywać istniejące, bądź stworzone (`next_combination`) algorytmy.