# Sortowanie bez porównań

W przypadku, gdy klucze, które porównujemy, są niewielkimi liczbami naturalnymi, możemy posortować je nie używając porównań:

## Sortowanie przez zliczanie - *counting sort*

- Jeśli klucze należą do zbioru $\{0, \ldots, m-1\}$
- Stwórz tablicę $m$ pustych kolejek `FIFO<T> kolejka[m];`
- Każdy element tablicy wstaw do właściwej kolejki `kolejka[x.key].push(x);`
- Opróżnij kolejno każdą z kolejek zapisując wyjmowane liczby w wyjściowej tablicy

# Sortowanie przez zliczanie - *counting sort*

## Implementacja poglądowa

```
template <class T>
void counting_sort(T t)
{
  FIFO<t> Q[m];
  for(auto x:t)
     Q[x.key].push(x)
  int i=0;
  for(auto &q:Q)
    while(!q.empty())
      t[i++]=q.get();
}
```

# Sortowanie przez zliczanie - *counting sort*

## Implementacja efektywna

```
template <class T, class P>
void counting_sort(T t[], int n, int m, P key)
{
  int count[m]={0};  // tablica liczników
  T *b=new T[n];     // przydzielenie pamięci na bufor

  for(int i=0; i<n; ++i)
      count[key(t[i])]++;  // zliczanie: count[k] = ilość kluczy równych k

  for(int k=1; k<m; ++k)
      count[k] += count[k-1]; // akumulacja: count[k] = ilość kluczy mniejszych bądź równych k

  for(int i=n-1; i>=0; --i)      // wkładanie do kolejek: idziemy po tablicy od końca
    b[--count[key(t[i])]] = t[i]; // bo kolejki wypełniamy też od końca

  for(int i=0; i<n; ++i)
    t[i]= b[i];          // przepisanie z bufora do tablicy wyjściowej

  delete []b;           // zwolnienie bufora z pamięci
}
```

# Sortowanie pozycyjne - *Radix sort*

- posortuj tablicę przez zliczanie kolejno względem każdej cyfry
- począwszy od najmniej znaczącej (cyfra jedności)
- aż do najbardziej znaczącej

```
void radix_sort(int t[], int n)
{
   int max = array_maximum(t,n);
   int m = 10; // może być też np. m=ceil(sqrt(max)), żeby były tylko 2 przebiegi
   for(int r=1; r <= max; r*= m)
     counting_sort(t, n, m, [r,m](int x){return x/r%m;}); // sortowanie względem cyfry r-ek
}
```

# Sortowanie kubełkowe - *Bucket sort*

Dobrze nadaje się do sortowania liczb równomiernie rozłożonych w jakimś przedziale:

- Podziel przedział zawierający liczby z tabeli na $m$ równych kawałków
- Posortuj tablicę przez zliczanie traktując numer kawałka zawierającego liczbę $x$ jako jej klucz
- Posortuj tablicę procedurą `insertion_sort`

```
void bucket_sort(double t[], int n)
{
   double min = array_minimum(t,n);
   double max = array_maximum(t,n);
   max+=(max-min)/100;
   int m = n/3+1; // tak dobieramy ilość kawałków by w każdym znalazły się np 3 liczby
   counting_sort(t, n, m, [min,max,m](double x){return (int) (x-min)/(max-min)*m;});
   insertion_sort(t, n);
}
```