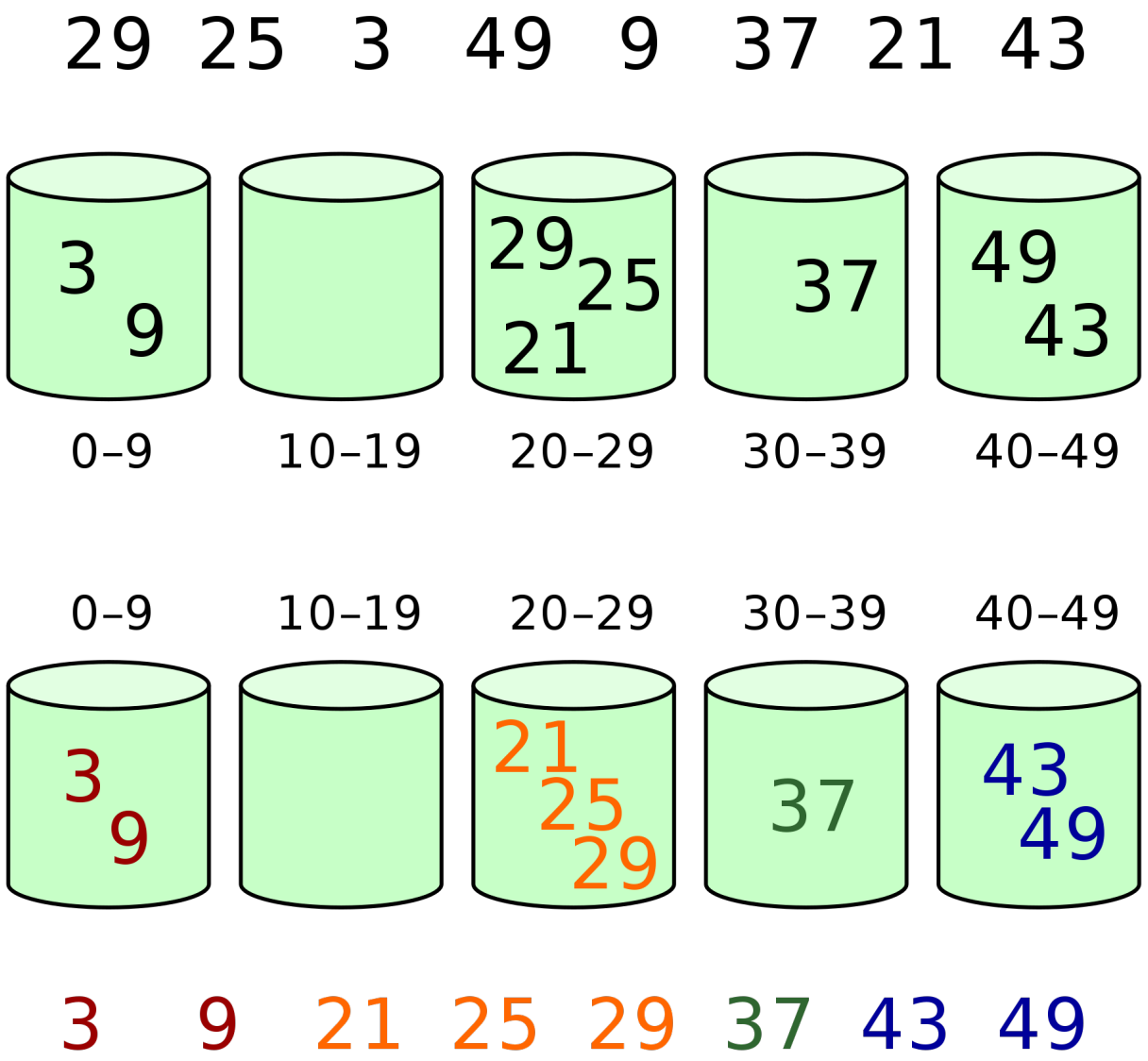
T: Sortowanie kubełkowe

1. Sortowanie kubełkowe (ang. *bucket sort*) – jeden z algorytmów sortowania, najczęściej stosowany, gdy liczby w zadanym przedziale są rozłożone jednostajnie, ma on wówczas złożoność $\Theta(n)$. W przypadku ogólnym pesymistyczna złożoność obliczeniowa tego algorytmu wynosi $O(n^2)$.

Pomysł takiego sortowania podali po raz pierwszy w roku 1956 E. J. Issac i R. C. Singleton.



2. Sposób działania

Idea działania algorytmu sortowania kubełkowego:

1. Podziel zadany przedział liczb na *k* podprzedziałów (*kubełków*) o równej długości.
2. Przypisz liczby z sortowanej tablicy do odpowiednich kubełków.
3. Sortuj liczby w niepustych kubełkach.
4. Wypisz po kolei zawartość niepustych kubełków.

Zazwyczaj przyjmuje się, że sortowane liczby należą do przedziału od 0 do 1 - jeśli tak nie jest, to można podzielić każdą z nich przez największą możliwą (jeśli znany jest przedział) lub wyznaczoną. Należy tu jednak zwrócić uwagę, że wyznaczanie największej możliwej liczby w tablicy m-elementowej ma złożoność obliczeniową O(m).

3. Porządkowanie kubełkowe:

Sortowanie kubełkowe ciągu ***T[0...n-1]*** wymaga wykorzystania dodatkowej tablicy ***P[0...m-1]***, której elementy nazywamy kubełkami.

Do kubełka ***P[i]*** wrzucane są wszystkie elementy sortowanej tablicy ***T[0...n-1]***, które mają wartość ***i***. Po przejrzeniu całego ciągu do tablicy ***T[0...n-1]*** wpisywane są wartości zawarte w kubełkach. Sortowanie wykonywane jest więc przez gromadzenie wyrazów porządkowanego ciągu w odpowiednich kubełkach reprezentujących ich wartości, a nie z wykorzystaniem porównań. Omawiany algorytm realizowany jest z wykorzystaniem programowania dynamicznego. Kubełki ***P[0...m-1]*** reprezentowane są jako listy jednokierunkowe.

Przedstawiona metoda nie wykonuje sortowania w miejscu, jest jednak metodą stabilną.

Specyfikacja:

Dane: Liczba naturalna: ***n > 0*** (liczba elementów tablicy ***T***).

Liczba naturalna: ***m > 0*** (wartość, od której wszystkie elementy tablicy T są mniejsze).

n-elementowa tablica jednowymiarowa zawierająca liczby całkowite ***T[0...n-1]***,

gdzie ***0 < T[i] < m***,dla ***i=0, 1, ..., n-1*** (ciąg do posortowania).

Wynik: Posortowana niemalejąco n-elementowa tablica jednowymiarowa zawierająca liczby całkowite: ***T[0...n-1]***.

Lista kroków:

Krok 0. Wczytaj ***n, T[0...n-1]***.

Krok 1. Dla kolejnych wartości ***i: 0, 1, ..., m-1,*** wykonuj krok 2., a następnie przejdź - do kroku 3.

Krok 2. Utwórz listę pustą ***P[i]***.

Krok 3. Dla kolejnych wartości ***i: n-1, n-2, ..., 0,*** wykonuj krok 4., a następnie przejdź do kroku 5.

Krok 4. Wstaw ***T[i]*** na początek listy ***P[T[i]]***.

Krok 5. Wykonaj scalenie list ***P[0...m-2]*** w jedną listę ***P***, łącząc kolejno ***P[i+1] z P[i]***.

Krok 6. Przepisz kolejne elementy listy **P** do tablicy ***T[0...n-1]***.

Krok 7. Wypisz elementy tablicy ***T[0...n-1]***. Zakończ algorytm.

Przeprowadźmy analizę złożoności czasowej algorytmu. Wiemy, że ***m*** to liczba wartości wyrazów sortowanego ciągu. Operacją dominującą jest tutaj porównanie wykonywane w pętli oraz operacja przypisania realizowana podczas sortowania. W krokach pierwszym i drugim wykonywanych jest ***m*** operacji, w trzecim i czwartym - ***n***, w piątym - ***m-1***, a w szóstym - ***n***. Liczba wykonywanych działań wynosi więc ***m+n+(m-1)+n = 2n+2m-1***. Uzyskujemy złożoność rzędu ***O(n+m)***. Przy założeniu, że ***m*** ma wartość co najwyżej ***n***, możemy stwierdzić, że mamy złożoność liniową ***O(n)***.

Przykładowa implementacja

https://www.programiz.com/dsa/bucket-sort