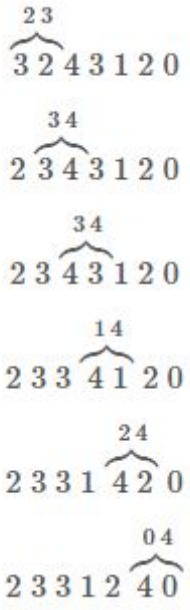
Algorytmy sortowania

1. **Sortowanie bąbelkowe (bubble sort)**

   Jeden z prostszych w implementacji algorytmów sortujących dane. Złożoność czasowa algorytmu jest rzędu **O(n^2)**, a pamięciowa **O(1)** Nie radzi sobie z większymi zbiorami.

   Działanie algorytmu:
   W każdym przejściu pętli wewnętrznej porównywane są ze sobą dwie kolejne wartości. W momencie kiedy porównywana wartość z prawej strony jest mniejsza od tej z lewej, następuje zamiana elementów. Przejście od lewej do prawej po wszystkich elementach to jedna pętla algorytmu. Pętla jest powtarzana do momentu uzyskania w pełni posortowanego zbioru, przy czym z każdą następną ilość elementów do sortowania zmniejsza się o 1.

   Jedna pętla algorytmu

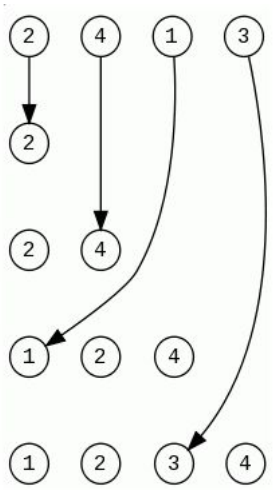
   

2. **Sortowanie przez wstawianie (insertion sort)**

   Zasada działania tego algorytmu przypomina porządkowanie kart w wachlarzu. Złożoność czasowa algorytmu jest rzędu **O(n^2),** a pamięciowa **O(1)**

Działanie algorytmu:

1. Weź dowolny element ze zbioru nieposortowanego.
2. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego, póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
3. Wyciągnięty element wstaw w miejsce, gdzie skończyłeś porównywać.
4. Jeśli zbiór elementów nieuporządkowanych jest niepusty, wróć do punktu 2.

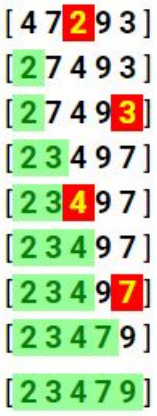


3. Sortowanie przez wymianę/wybór (selection sort)

Algorytm polega na wyszukiwaniu elementu mającego znaleźć się na żądanej pozycji i zamianie miejscami z tymi, który jest tam obecnie. Złożoność czasowa algorytmu jest rzędu **O(n^2).**

Działanie algorytmu:

1. Szukamy najmniejszego elementu
2. Wymieniamy go z pierwszym elementem w zbiorze
3. Kroki powtarzamy do pełnego uporządkowania zbioru



4. **Sortowanie szybkie (quick sort)**

   Algorytm ten opiera się na strategii “dziel i zwyciężaj”, którą można scharakteryzować w trzech punktach:

   1. Dziel - problem główny zostaje podzielony na podproblemy
   2. Zwyciężaj - znajdujemy rozwiązanie podproblemów
   3. Połącz - rozwiązania podproblemów zostają połączone w rozwiązanie problemu głównego

   Złożoność czasowa tego algorytmu to: $O(n \log n)$.

   Idea sortowania szybkiego jest następująca:
   - Dziel: najpierw sortowany zbiór dzielimy na dwie części w taki sposób, aby wszystkie elementy leżące w pierwszej części (zwanej lewą partycją) były mniejsze lub równe od wszystkich elementów drugiej części zbioru (zwanej prawą partycją).
   - Zwyciężaj: każdą z partycji sortujemy rekurencyjnie tym samym algorytmem.
   - Połącz: połączenie tych dwóch partycji w jeden zbiór daje w wyniku zbiór posortowany

5. **Sortowanie stogowe (heap sort)**

   Algorytm ten polega na budowaniu **Kopca** z elementów tablicy które chcemy posortować. Kopiec jest drzewem binarnym, w którym wszystkie węzły spełniają następujący warunek: **Węzeł nadrzędny jest większy lub równy węzłom potomnym.**

   Złożoność czasowa tego algorytmu to: **O(n log n),** a pamięciowa **O(n).**

   Schemat działania przedstawiony graficznie w bardzo przejrzysty sposób można znaleźć tutaj:
   https://eduinf.waw.pl/inf/alg/003_sort/m0015.php

6. Sortowanie przez zliczanie (counting sort)

Algorytm opiera się na zliczaniu, ile razy dana liczba występuje w ciągu, który mamy posortować. Następnie na podstawie danych zebranych wcześniej, utworzyć nowy ciąg z posortowanymi elementami. Może być używany tylko do tablic z liczbami całkowitymi.

Złożoność czasowa tego algorytmu jest rzędu: **O(m + n)**, gdzie m(ilość wartości kluczy), n(ilość sortowanych elementów).

{ 6 3 6 1 4 9 0 1 8 2 6 4 9 3 7 5 9 2 7 3 2 4 1 8 7 0 8 5 8 3 6 2 5 3 }
Schemat działania algorytmu:

- Do każdej wartości w zbiorze przygotowujemy licznik i ustawiamy go na 0.
  [0:0] [1:0] [2:0] [3:0] [4:0] [5:0] [6:0] [7:0] [8:0] [9:0]

- Przeglądamy kolejne elementy zbioru i zliczamy ich wystąpienia w odpowiednich licznikach.
  [0:2] [1:3] [2:4] [3:5] [4:3] [5:3] [6:4] [7:3] [8:4] [9:3]

- Poczynając od drugiego licznika sumujemy zawartość licznika oraz jego poprzednika
  [0:2] [1:5] [2:9] [3:14] [4:17] [5:20] [6:24] [7:27] [8:31] [9:34]

- W wyniku tej operacji w każdym liczniku otrzymaliśmy ilość wartości mniejszych lub równych numerowi licznika, które występują w zbiorze wejściowym.
  [0:2] - w zbiorze wejściowym są dwie wartości 0
  [1:5] - w zbiorze wejściowym jest pięć wartości mniejszych lub równych 1
  [2:9] - w zbiorze wejściowym jest dziewięć wartość mniejszych lub równych 2, itd.

Wartość:	0	0	1	1	1	2	2	2	2	3	3	3	3	3	4	4	4	5	5	5	6	6	6	6	7	7	7	8	8	8	8	9	9	9
Pozycja:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34

Stan licznika określa teraz ostatnią pozycję w zbiorze uporządkowanym, na której należy umieścić wartość równą numerowi licznika.

7. Sortowanie przez scalanie (merge sort)

Podstawową operacją tego algorytmu jest scalanie dwóch zbiorów uporządkowanych w jeden zbiór również uporządkowany. Operację scalenia realizuje się wykorzystując pomocniczy zbiór, w którym będą tymczasowo odkładane scalane elementy dwóch zbiorów.
Złożoność czasowa tego algorytmu to **O(n * log(n)),** a pamięciowa **O(n).**

Schemat działania przedstawiony graficznie w bardzo przejrzysty sposób można znaleźć tutaj:
https://eduinf.waw.pl/inf/alg/003_sort/m0013.php

Źródła:
https://eduinf.waw.pl/inf/alg/003_sort/m0009.php
http://www.algorytm.org/algorytmy-sortowania/
https://eduinf.waw.pl/inf/alg/003_sort/m0018.php
https://eduinf.waw.pl/inf/alg/003_sort/m0015.php
https://eduinf.waw.pl/inf/alg/003_sort/m0023.php
https://eduinf.waw.pl/inf/alg/003_sort/m0013.php