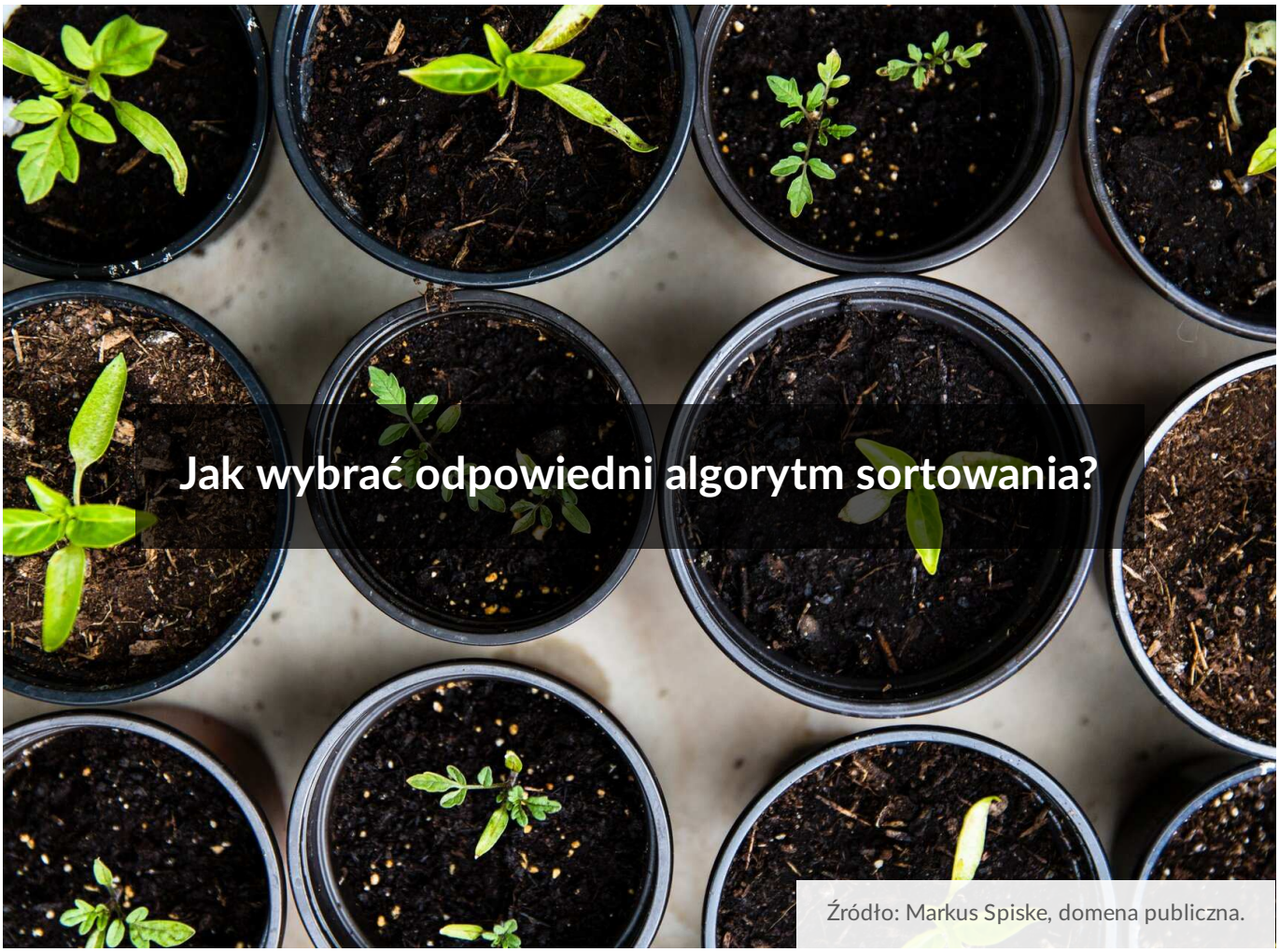
Jak wybrać odpowiedni algorytm sortowania?



Źródło: Markus Spiske, domena publiczna.

Istnieje wiele algorytmów sortowania, różniących się nie tylko implementacją, ale także wydajnością. W pracy programisty kluczową zdolnością jest umiejętność wybrania najlepszego algorytmu dla danej sytuacji.

W tym dobierzemy odpowiednie algorytmy do konkretnych sytuacji.

Informacje o algorytmach sortujących możesz znaleźć w e-materiałach:

- Sortowanie przez wybieranie,
- Sortowanie przez wstawianie,
- Sortowanie bąbelkowe,
- Sortowanie kubełkowe,
- Sortowanie pozycyjne dat,
- Sortowanie pozycyjne liczb,

- Sortowanie pozycyjne słów,
- Sortowanie przez zliczanie,
- Sortowanie przez scalanie,
- Sortowanie szybkie.

> **Twoje cele**
>
> - Wyjaśnisz, co warto wziąć pod uwagę, wybierając najlepszą metodę sortowania.
> - Wykonasz ćwiczenia sprawdzające twoją wiedzę z zakresu doboru właściwego algorytmu do typu problemu.
> - Przeanalizujesz, jak dobierać najlepszy algorytm sortowania dla przykładowej sytuacji.

Przeczytaj

Jak dobrać odpowiedni algorytm sortowania?

Od razu możemy odrzucić algorytmy:

- sortowania bąbelkowego,
- sortowania przez wstawianie,
- sortowania koktajlowego,
- sortowania przez wybieranie.

Algorytmy te stosowane są głównie do poznania zasad działania algorytmów sortowania. Są proste w zrozumieniu, jednak nie znajdują zastosowania w praktyce. Wynika to z faktu, że charakteryzują się one kwadratową złożonością obliczeniową, a przez to są zdecydowanie mniej efektywne od algorytmów sortowania, które posiadają złożoność $O\left(n \log n\right)$.

Przed wyborem algorytmu przyglądamy się danym, z jakimi mamy do czynienia.

- Jeżeli sortujemy słowa, daty lub niewielkie liczby, rozważmy użycie algorytmu sortowania pozycyjnego. Jest on idealny w takich sytuacjach, a dodatkowo charakteryzuje go złożoność obliczeniowa wynosząca $O(n + k)$, gdzie n to liczba elementów do posortowania, a k to liczba znaków w najdłuższym elemencie. Jest to złożoność liniowa, zatem algorytm ten jest jednym z najszybszych algorytmów sortowania.
- Jeśli mamy do czynienia ze zbiorem, w którym występuje niewiele różnych elementów, np. cyfry lub litery powtarzające się wielokrotnie, sortowanie przez zliczanie będzie bardzo dobrym wyborem. Musimy pamiętać, że algorytm ten wymaga użycia dodatkowej pamięci, ponieważ dane zapisywane są tymczasowo w tablicy pomocniczej.

Rozpatrując wybór najlepszej metody, powinniśmy wziąć pod uwagę organizację i strukturę danych, które sortujemy, oraz dostępne miejsce.

- Sortowanie szybkie będzie lepsze, jeżeli sortujemy tablice, a także jeżeli nie mamy do dyspozycji dużej ilości pamięci komputera. Algorytm jest mniej wydajny dla dużej liczby danych.
- Sortowanie przez scalanie działa lepiej dla list, ale wymaga dużej ilości dodatkowej pamięci. Nie zaobserwujemy jednak znacznego spadku wydajności dla dużej liczby danych do sortowania.

Istnieją również algorytmy hybrydowe, często wyspecjalizowane dla jednego konkretnego typu danych. Przykładem takiego algorytmu może być Timsort, używany w języku Python (algorytm Timsort implementuje metodę `sort()` w listach w języku Python). Tego typu algorytmy są bardzo skomplikowane i wieloetapowe, jednak sprowadzają się do omówionych tutaj algorytmów, użytych w niekonwencjonalny sposób.

> **Ciekawostka**
>
> Choć najczęściej zależy nam na jak najszybszym posortowaniu elementów, powstało kilka algorytmów o bardzo dużej złożoności obliczeniowej. Przykładem takiego algorytmu jest algorytm Bogosort. Polega on na nieustannym ustawieniu elementów w losowej kolejności i sprawdzeniu, czy zostały one posortowane.

Szczególny przypadek

Warto pamiętać o tym, że przedstawione tutaj sposoby użycia algorytmów są typowe i najczęściej spotykane. Możemy jednak znaleźć się w sytuacji, w której te zasady trzeba będzie zmienić. Wówczas nie bójmy się użyć innego niż podany tutaj algorytm, czy go zmodyfikować. Jeżeli czujemy się na siłach, możemy nawet spróbować zaprojektować swój własny algorytm sortowania.

Wyobraźmy sobie, że naszym zadaniem jest przygotowanie algorytmu sortowania zbioru liczb całkowitych. Zależy nam na tym, aby najpierw posortować w kolejności

niemalejącej liczby parzyste, a następnie nieparzyste. Wynikiem działania algorytmu dla zbioru A = {5, 1, 2, 6, 4, 3, -3} byłby więc zbiór B = {2, 4, 6, -3, 1, 3, 5}.

Taki algorytm możemy zaprojektować na kilka sposobów. Jeden z najbardziej intuicyjnych polega na dzieleniu zbioru A na dwa podzbiory: liczb parzystych i nieparzystych. Następnie sortujemy każdy ze zbiorów niezależnie, układając elementy w kolejności niemalejące i wreszcie łączymy oba zbiory.

Jaki algorytm wybrać?

Sortowanie przez wybieranie
Użyj go, gdy: • sprawdzasz, czy zbiór jest posortowany; • masz ograniczoną pamięć do wykorzystania.
Sortowanie przez wstawianie
Użyj go, gdy: • zbiór danych jest już w większości posortowany; • sortujesz niewielką liczbę elementów.
Sortowanie bąbelkowe
Użyj go, gdy: • sprawdzasz, czy zbiór jest posortowany; • zbiór danych jest już w większości posortowany; • sortujesz niewielką liczbę elementów.
Sortowanie kubełkowe
Użyj go, gdy: • sortowane dane są równomiernie rozmieszczone
Sortowanie pozycyjne
Użyj go, gdy: • dane mogą być posegregowane leksykograficznie.

Sortowanie przez zliczanie
Użyj go, gdy: • sortowane elementy należą do niewielkiego przedziału i powtarzają się wielokrotnie.
Sortowanie przez scalanie
Użyj go, gdy: • dane dostępne są sekwencyjnie.
Sortowanie szybkie
Użyj go, gdy: • potrzebujesz posortować elementy w sposób wydajny o średniej złożoności rzędu O(n log n).

Słownik

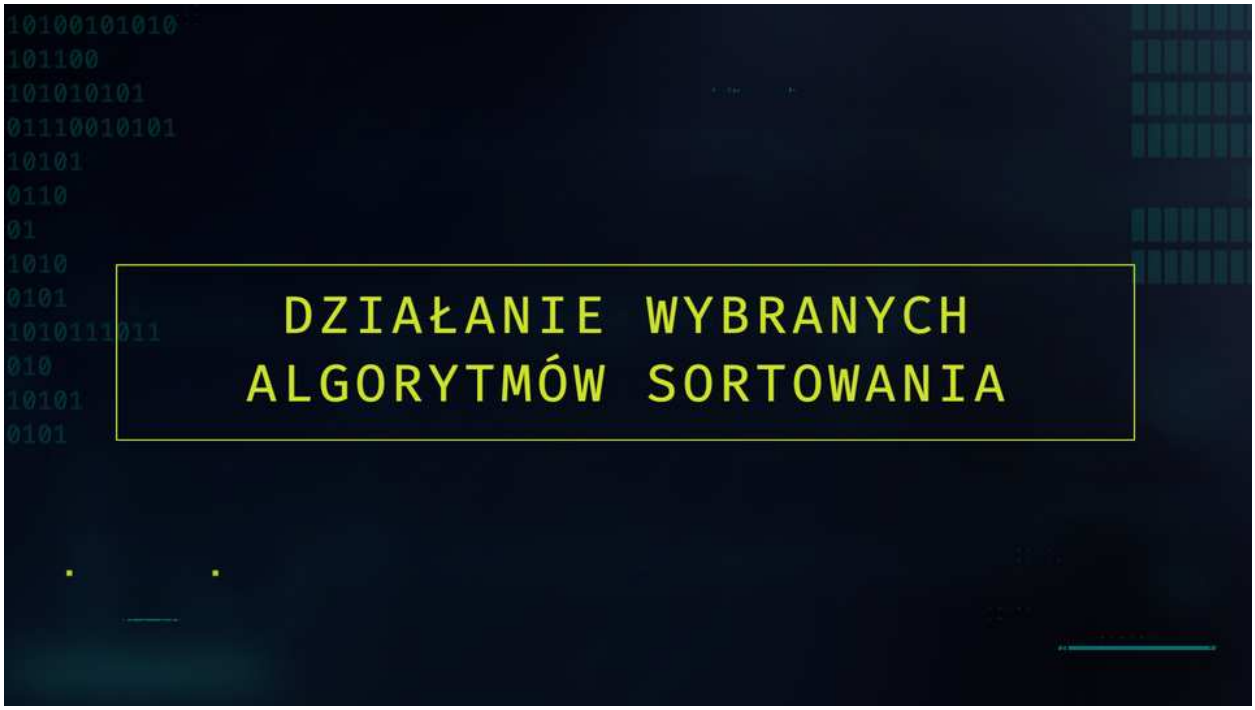
> **Bogosort**
>
> algorytm sortowania, który polega na ciągłym losowym ustawianiu sortowanych elementów i sprawdzaniu, czy po wymieszaniu zostały posortowane

> **Timsort**
>
> algorytm hybrydowy; algorytm, który dla dostatecznie małych zestawów danych przełącza się z algorytmu sortowania przez scalanie na algorytm sortowania przez wstawianie

> **złożoność obliczeniowa**
>
> cecha algorytmu mówiąca o tym, ile zasobów jest potrzebnych do rozwiązania problemu obliczeniowego

Animacja

> **Polecenie 1**
> Przyjrzyj się działaniu różnych algorytmów sortowania w animacji. Następnie spróbuj w podobny sposób sporządzić rysunki kolejnych kroków dla algorytmu sortowania przez scalanie oraz sortowania koktajlowego.



Film dostępny pod adresem /preview/resource/R161FuTdt5oHe

Źródło: reż. Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Film nawiązujący do treści materiału optymalizacji różnych algorytmów sortujących.

Przykłady problemów, w których wybór algorytmu sortowania ma znaczenie

> **Problem 1**
> W zbiorze danych do posortowania znajdują się jedynie liczby jednocyfrowe.

> **Problem 2**
> Mamy do dyspozycji bardzo ograniczoną pamięć komputera.

Polecenie 2

Podaj inne przykłady problemów, w których wybór odpowiedniego algorytmu sortowania ma znaczenie.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż algorytmy o złożoności czasowej $O(n^2)$. Zaznacz wszystkie poprawne odpowiedzi.

- [ ] sortowanie przez wstawianie
- [ ] sortowanie szybkie
- [ ] sortowanie przez scalanie
- [ ] sortowanie bąbelkowe

Ćwiczenie 2

Wybierz poprawne zdanie na temat złożoności obliczeniowej.

- ( ) Do sprawdzenia, czy dane są posortowane potrzebujemy algorytmu o złożoności co najmniej kwadratowej.
- ( ) Im większa jest złożoność obliczeniowa, tym szybszy algorytm.
- ( ) Złożoność obliczeniowa określa ilość zasobów potrzebnych do rozwiązania problemu obliczeniowego.

Ćwiczenie 3

Uzupełnij zdania.

[ ] jest dobrym algorytmem do sortowania słów.

[ ] będzie działało znacznie wolniej dla dużej liczby danych, ale dla małej liczby będzie szybkie.

Sortowanie szybkie | Sortowanie bąbelkowe | Sortowanie kubełkowe | Sortowanie pozycyjne

Ćwiczenie 4

Wyobraź sobie, że twój program ma przechowywać tytuły książek w bibliotece. Jaką strategię powinien on stosować, aby spis był zawsze posortowany alfabetycznie?

Twoja odpowiedź:

Ćwiczenie 5

Wybierz algorytm sortowania, który będzie najszybszy dla każdych możliwych danych.

- ◯ żadna odpowiedź nie jest poprawna
- ◯ swortowanie szybkie
- ◯ sortowanie przez scalanie
- ◯ sortowanie koktajlowe

Ćwiczenie 6

Zaznacz wszystkie poprawne odpowiedzi. Sortowanie pozycyjne sprawdzi się w przypadku sortowania:

- [ ] dat
- [ ] dużych liczb
- [ ] słów

Ćwiczenie 7

Uzupełnij zdania.

Sortowanie przez zliczanie najlepiej sprawdza się w sytuacji, w której do posortowania mamy [ ] różnych danych powtarzających się [ ] razy.

wiele | niewiele

Ćwiczenie 8

Podaj przykład sytuacji, w której dobrym pomysłem byłoby stworzenie własnego algorytmu sortowania. Uzasadnij, dlaczego poznane do tej pory algorytmy nie sprawdzą się.

Twoja odpowiedź:

Dla nauczyciela

Autor: Maurycy Gast

Przedmiot: Informatyka

Temat: Jak wybrać odpowiedni algorytm sortowania?

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

c) porządkowania ciągu liczb: przez wstawianie i metodą bąbelkową,

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

e) sortowania ciągu liczb przez scalanie,

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

a) wyszukiwanie elementów liniowe i przez połowienie (do znajdowania elementów w zbiorze, sortowania przez wstawianie, przybliżonego rozwiązywania równań, sprawdzania przynależności punktu do wielokąta wypukłego),

c) metodę dziel i zwyciężaj (jednoczesne znajdowanie minimum i maksimum, sortowanie przez scalanie i szybkie),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Wyjaśnisz, co warto wziąć pod uwagę, wybierając najlepszą metodę sortowania.
- Wykonasz ćwiczenia sprawdzające twoją wiedzę z zakresu doboru właściwego algorytmu do typu problemu.
- Przeanalizujesz, jak dobierać najlepszy algorytm sortowania dla przykładowej sytuacji.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne;
- burza mózgów.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Jak wybrać odpowiedni algorytm sortowania?". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wprowadza uczniów szczegółowo w temat lekcji i jej cele. Może posłużyć się wyświetloną na tablicy zawartością sekcji „Wprowadzenie".
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. Metodą burzy mózgów uczniowie referują najważniejsze informacje, jakie na temat sortowania znaleźli w sekcji „Przeczytaj", przygotowując się do lekcji.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Animacja". Uczniowie wspólnie zapoznają się z jego treścią. Przyglądają się działaniu różnych algorytmów sortujących w animacji. Następnie próbują w podobny sposób sporządzić rysunki kolejnych kroków dla algorytmu sortowania przez scalanie oraz sortowania koktajlowego.

3. **Ćwiczenie umiejętności.** Uczniowie realizują indywidualnie ćwiczenia nr 1-7, po ich wykonaniu porównują otrzymane wyniki z inną osobą.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 8 z sekcji „Sprawdź się".

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.
- Materiał ma charakter podsumowujący, zatem zaleca się, by uczniowie przed lekcją powtórzyli informacje na temat poznanych metod sortowania.