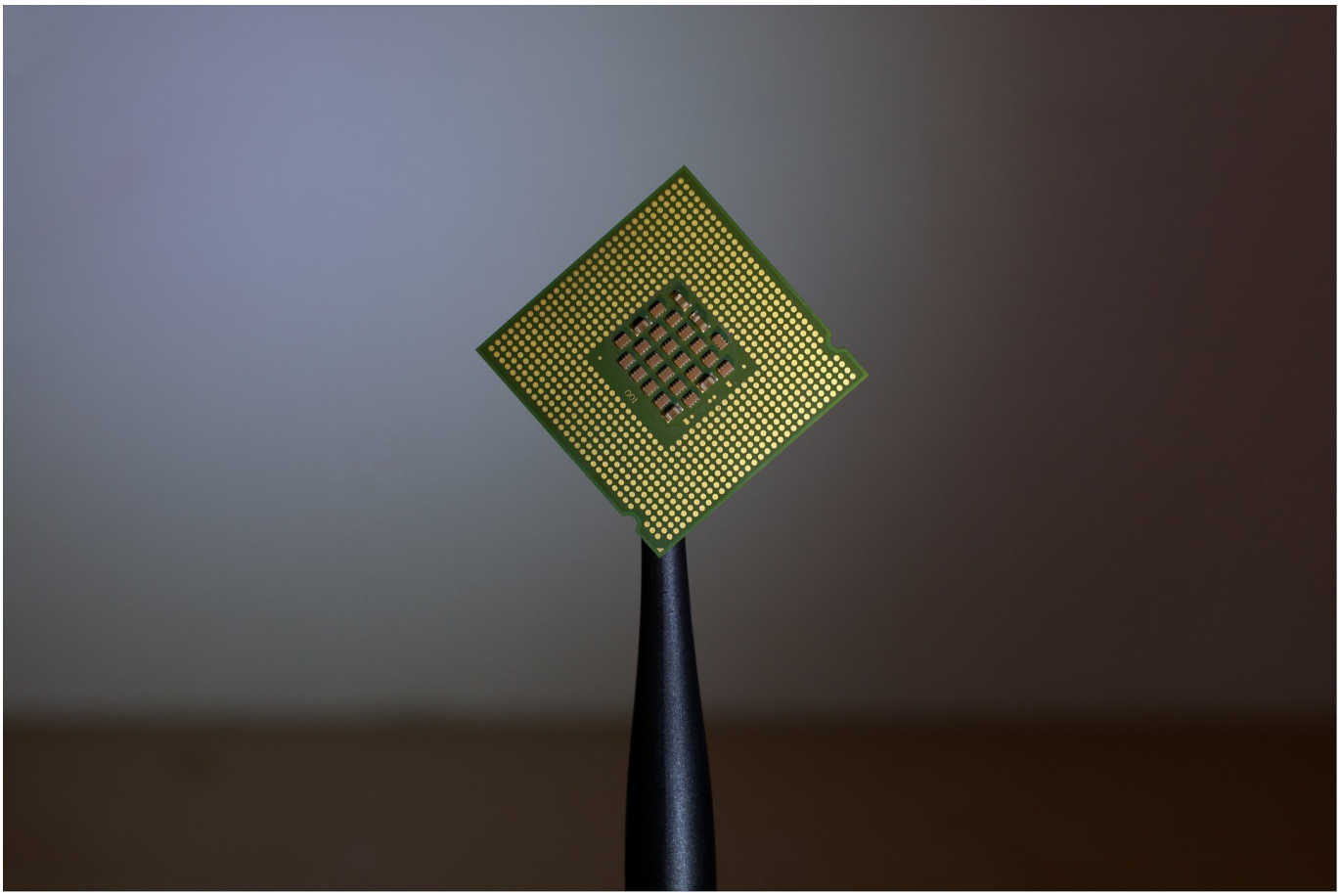


Jak porównywać złożoność obliczeniową?

Źródło: Brian Kostiuk, domena publiczna.

Wiesz już, jak zdefiniować złożoność obliczeniową i jakie jej rodzaje możemy wyróżnić.

Do rozwiązania jednego problemu wykorzystać możemy różne algorytmy – każdy z nich może mieć inną (czasem skrajnie inną!) złożoność obliczeniową. W jaki sposób porównywać złożoność obliczeniową, by wybrać algorytm najodpowiedniejszy do rozwiązania konkretnego problemu?

Więcej informacji o złożoności obliczeniowej algorytmów znajdziesz w e-materiałach:

- Złożoność obliczeniowa algorytmów,
- Algorytmy o złożoności kwadratowej,
- Algorytmy o złożoności logarytmicznej i liniowo-logarytmicznej,
- Algorytmy o złożoności wykładniczej.

> **Twoje cele**
>
> - Zdefiniujesz operacje dominujące.
> - Przeanalizujesz złożoność czasową algorytmów.
> - Prześledzisz koncepcję pamięciowej złożoności obliczeniowej.

Przeczytaj

Złożoność obliczeniowa

Wiesz już, jak możemy zdefiniować złożoność obliczeniową – opisaliśmy to w e-materiale Złożoność obliczeniowa algorytmów. Złożoność obliczeniowa algorytmu pozwala określić ilość zasobów komputerowych niezbędnych do wykonania czynności opisanych w algorytmie w zależności od rozmiaru danych wejściowych.

Analiza złożoności obliczeniowej

Analiza złożoności obliczeniowej jest kluczowym narzędziem w pracy programisty pomagającym w wyborze optymalnych algorytmów dla różnych zastosowań.

W innych e-materiałach przedstawiliśmy różne klasy złożoności obliczeniowej, które można przypisać algorytmom. Porównywanie złożoności obliczeniowej różnych algorytmów pozwala na ocenę ich wydajności. Aby to zrobić, należy zidentyfikować klasę złożoności każdego algorytmu i porównać je względem siebie.

> **Ważne!**
>
> Warto jednak pamiętać, że złożoność obliczeniowa jest jednym z wielu czynników wpływających na wydajność algorytmu w praktyce.

Analiza złożoności obliczeniowej pomaga programistom w wyborze najlepszych algorytmów dla różnych zastosowań, szczególnie w sytuacjach, gdy istotna jest efektywność. Dzięki analizie złożoności obliczeniowej możemy unikać wykorzystywania algorytmów o wysokiej złożoności obliczeniowej w przypadkach, gdy mogą one prowadzić do problemów z wydajnością systemu, zwłaszcza przy dużych zbiorach danych.

Analiza złożoności obliczeniowej pozwala również ocenić, czy proponowane usprawnienia w istniejących algorytmach faktycznie wpłyną na poprawę ich wydajności.

W praktyce analiza złożoności obliczeniowej nie zawsze jest wystarczająca do wyboru najlepszego algorytmu, ponieważ musimy uwzględnić także inne czynniki, np.:

- złożoność pamięciowa,
- ograniczenia sprzętowe.

Analiza złożoności obliczeniowej może być przeprowadzana zarówno teoretycznie, jak i eksperymentalnie. W teoretycznym podejściu badamy algorytm pod kątem liczby operacji, które wykonuje w zależności od wielkości danych wejściowych.

Eksperymentalne podejście polega na przeprowadzaniu testów wydajności, mierząc rzeczywisty czas wykonania algorytmu dla różnych wielkości danych wejściowych.

Dostępne są różne narzędzia i techniki do tego wykorzystywane. W tym e-materiale zaprezentujemy podejście teoretyczne.

Przykładowe problemy

Sortowanie

W przyszłości poznasz różne algorytmy sortowania. W e-materiałach opisujemy następujące:

- sortowanie przez wybieranie,
- sortowanie przez wstawianie,
- sortowanie bąbelkowe,
- sortowanie kubełkowe,
- sortowanie pozycyjne,
- sortowanie przez zliczanie,
- sortowanie przez scalanie.

Sortowanie przez wybieranie

Sortowanie przez wybieranie działa na zasadzie wyszukiwania najmniejszego (lub największego) elementu w tablicy i zamiany jego pozycji z pierwszym (lub ostatnim) elementem. Następnie proces jest powtarzany dla pozostałych elementów.

W przypadku **sortowania przez wybieranie** do czynienia mamy ze złożonością $O(n^2)$.

Sortowanie przez wstawianie

Sortowanie przez wstawianie polega na iteracyjnym wstawianiu elementów do uprzednio posortowanej części tablicy. Algorytm ten przegląda elementy od drugiego do końca tablicy, porównuje je z elementami posortowanymi wcześniej i wstawia je na odpowiednie miejsce.

W przypadku **sortowania przez wstawianie** do czynienia mamy ze złożonością $O(n^2)$.

Sortowanie bąbelkowe

Sortowanie bąbelkowe działa na zasadzie porównywania sąsiednich elementów w tablicy i zamiany ich miejscami, jeśli są w złej kolejności. Proces ten jest powtarzany, aż do uzyskania posortowanej tablicy.

W przypadku **sortowania bąbelkowego** do czynienia mamy ze złożonością $O(n^2)$.

Sortowanie kubełkowe

Sortowanie kubełkowe działa przez podział wartości na kilka „kubełków" (przedziałów) i sortowanie ich niezależnie, zwykle za pomocą innego algorytmu sortowania. Po posortowaniu elementów w każdym kubełku są one łączone z powrotem, tworząc posortowaną tablicę.

W przypadku **sortowania kubełkowego** do czynienia mamy ze złożonością $O(n + k)$ dla przypadków optymistycznego i średniego oraz $O(n^2)$ dla przypadku pesymistycznego.

Nie będziemy analizować kolejnych algorytmów. Na tym etapie możemy zdecydować, że najmniejszą złożoność obliczeniową dla tego przypadku ma algorytm **sortowania przez wstawianie.**

Porównanie złożoności algorytmów sortowania

Sprawdźmy złożoność obliczeniową poszczególnych algorytmów dla danego zbioru danych. W e-materiale Jak wybrać odpowiedni algorytm sortowania? omawiamy to, w jaki algorytm sortowania należy wybrać w danym przypadku – nie będziemy brać tego teraz pod uwagę, natomiast warto mieć to w pamięci w kontekście tego, o czym już zostało powiedziane – na efektywność algorytmu wpływa nie tylko złożoność obliczeniowa, ale również to, czy jest odpowiednim wyborem dla danego problemu.

Załóżmy, że chcemy posortować niemalejąco dany zbiór: {2, 7, 0, 5, 2, 3, 15, 1, 9, 12, 94, 87}. Jak będzie wyglądać złożoność obliczeniowa poszczególnych algorytmów?

Aby określić liczbę operacji dla obu algorytmów, musimy wziąć pod uwagę ich złożoność obliczeniową.

Wiemy, że złożoność **algorytmu sortowania przez wybieranie** wynosi $O(n^2)$, zatem algorytm wykona **maksymalnie** 144 operacje. Natomiast liczba **faktycznych** operacji może być inna. W przypadku implementacji zaprezentowanej w e-materiale dotyczącym sortowania przez wybieranie możemy określić, że dla podanych danych zostanie wykonanych 66 operacji porównania oraz 11 operacji zamiany. Daje to w sumie **77 operacji**.

Złożoność **algorytmu sortowania przez wstawianie** do czynienia mamy ze złożonością $O(n^2)$, zatem algorytm również wykona **maksymalnie** 144 operacje. Natomiast w przypadku tego algorytmu liczba **faktycznych** operacji może być inna. W przypadku implementacji zaprezentowanej w e-materiale dotyczącym sortowania przez wstawianie możemy określić, że dla podanych danych zostanie wykonanych 55 operacji porównania oraz 16 operacji zamiany. Daje to w sumie **71 operacji**.

Algorytm sortowania bąbelkowego charakteryzuje się złożonością $O(n^2)$, zatem algorytm wykona **maksymalnie** 144 operacje. Natomiast liczba **faktycznych** operacji może być inna. W przypadku implementacji zaprezentowanej w e-materiale dotyczącym sortowania bąbelkowego możemy określić, że dla podanych danych zostanie wykonanych 66 operacji porównania oraz 43 operacje zamiany. Daje to w sumie **109 operacji**.

W przypadku **sortowania kubełkowego** do czynienia mamy ze złożonością $O(n+k)$ dla przypadków optymistycznego i średniego oraz $O(n^2)$ dla przypadku pesymistycznego. Wiemy, że n oznacza liczbę elementów w zbiorze. Co jednak z k? Jest to liczba **kubełków**. Liczbę kubełków obliczamy, odejmując od największej wartości w tablicy wartość najmniejszą oraz dodając do tego 1.

W analizowanym przypadku $k = 94 - 0 + 1$. Algorytm wykonałby **maksymalnie** 144 operacje (dla przypadku pesymistycznego) oraz 107 dla przypadków optymistycznego oraz średniego.

Natomiast liczba **faktycznych** operacji może być inna. W przypadku implementacji zaprezentowanej w e-materiale dotyczącym sortowania kubełkowego możemy określić, że dla podanych danych zostanie wykonanych 12 operacji zliczeń oraz 95 operacji inicjalizacji kubełków. Daje to w sumie **107 operacji**.

W analizowanym przypadku najlepszym wyborem byłby zatem **algorytm sortowania przez wybieranie**.

Dlaczego do określania złożoności czasowej używamy notacji „duże O”? Dokładne wyznaczenie czasu realizacji algorytmu jest bardzo trudne. Istnieje wiele czynników, które trzeba by było wziąć pod uwagę, m.in. szybkość wykorzystywanego procesora oraz pamięci czy język programowania. Dlatego zamiast podawania dokładnej liczby sekund wyznaczamy pewien rząd wielkości, który ułatwia przybliżenie czasu wykonywania programu i jest niezależny od środowiska.

Słownik

złożoność czasowa

ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych
notacja duże O

miara określająca, w jaki sposób rośnie wartość funkcji wraz ze zwiększaniem jej argumentów; służy do wyznaczania górnych granic asymptotycznych.
operacje dominujące

operacje charakterystyczne dla danego algorytmu, na ogół zajmujące w nim najwięcej czasu
pamięciowa złożoność obliczeniowa

ilość pamięci potrzebnej do wykonania zadania, wyrażona jako funkcja ilości danych
złożoność obliczeniowa

ilość zasobów komputerowych potrzebnych do wykonania zadania
złożoność optymistyczna

ilość zasobów potrzebna programowi do zrealizowania zadania dla najkorzystniejszego przypadku danych; zapisywana za pomocą notacji „małe O" – o
złożoność oczekiwana

inaczej złożoność średnia; ilość zasobów potrzebna do zrealizowania zadania dla statystycznie oczekiwanych danych; zapisywana za pomocą notacji theta – Θ
złożoność pesymistyczna

ilość zasobów potrzebna do zrealizowania zadania w przypadku najgorszych danych; zapisywana za pomocą notacji „duże O" – O
przypadek optymistyczny

dane wejściowe są już posortowane w wymaganym porządku
przypadek pesymistyczny

dane wejściowe są posortowane odwrotnie od sposobu, w jaki mamy je posortować, np. jeśli chcemy otrzymać tablicę posortowaną rosnąco, a na wejściu otrzymamy tablicę posortowaną malejąco

Symulacja interaktywna

Polecenie 1

Symulacja przedstawia złożoności obliczeniowe różnych rozwiązań tego samego problemu obliczeniowego – znalezienia NWD dwóch liczb. Dane są dwie liczby – a oraz b, takie że $a > b$. Naszym zadaniem jest znalezienie ich największego wspólnego dzielnika. By to zrobić, możemy wykorzystać kilka metod.

- Rozwiązanie siłowe (ang. *brute force*) – to metoda polegająca na sprawdzeniu wszystkich możliwych przypadków w celu znalezienia rozwiązania danego problemu, w tym wypadku sprawdzamy wszystkie liczby całkowite od 1 do b, poszukując tych, które są wspólnymi dzielnikami liczb a oraz b – złożoność tego rozwiązania to $O(min(a, b))$.

- Algorytm Euklidesa z wykorzystaniem iteracji – jego złożoność obliczeniowa wynosi $O(log(min(a, b)))$. Więcej na temat tego algorytmu znajdziesz w e-materiale Algorytm Euklidesa.

- Algorytm Euklidesa z użyciem rekurencji – jego złożoność obliczeniowa wynosi również $O(log(min(a, b)))$. Więcej na temat tego algorytmu znajdziesz w e-materiale Algorytm Euklidesa.

- Rozkład na czynniki pierwsze – jego złożoność obliczeniowa to $O\left(\sqrt{\min a,\ b}\right)$.

- Algorytm Steina – jego złożoność obliczeniowa to $O(log(a) * log(b))$.

> **Definicja: Algorytm Steina**
>
> Algorytm służący do wyliczania największego wspólnego dzielnika dwóch liczb naturalnych. Pozwala znacząco zmniejszyć czas obliczeń. Rozwiązanie opiera się na przyjęciu następujących założeń:
>
> 1. Przyjmując, że $NWD(a, 0) = NWD(0, a) = a$ oraz $NWD(0, 0) = 0$, zakładamy, że obliczenia pozostaną poprawne.
>
> 2. Dla parzystych a, b mamy $NWD(a, b) = 2 \cdot NWD(a : 2, b : 2)$.
>
> 3. Gdy a jest liczbą parzystą, a b jest nieparzystą (pamiętajmy o przemienności NWD), to $NWD(a, b) = NWD(a : 2, b)$.

4. Dla nieparzystych a, b mamy $NWD(a,b) = NWD(|a-b| : 2, min(a,b))$.

Kroki od 2. do 4. wykonujemy w pętli do momentu, w którym $a = b$ albo $a = 0$.

Przeanalizuj wykres interaktywny. Przyjrzyj się temu, jak zmienia się liczba wykonywanych operacji dla poniższych par liczb:

- **przypadek 1:** 27 oraz 5,
- **przypadek 2:** 20 oraz 23,
- **przypadek 3:** 15 oraz 12,
- **przypadek 4:** 94 oraz 87.

Przygotuj własną symulację, w której zapiszesz złożoność obliczeniową dla różnych rozwiązań tego samego problemu, np. sortowania zbioru liczb. Złożoność oblicz dla różnych przypadków.

Polecenie 2

Zapisz wnioski wyciągnięte z zaprezentowanej w tej sekcji symulacji.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż, za pomocą której notacji zapisujemy złożoność obliczeniową

- ◯ małe o
- ◯ duże A
- ◯ małe a
- ◯ duże O

Ćwiczenie 2

Połącz w pary.

n	złożoność stała
1	złożoność liniowo-logarytmiczna
$n\ \log(n)$	złożoność wykładnicza
$\log(n)$	złożoność n-silnia
$n!$	złożoność liniowa
2^n	złożoność kwadratowa
n^2	złożoność logarytmiczna

Ćwiczenie 3

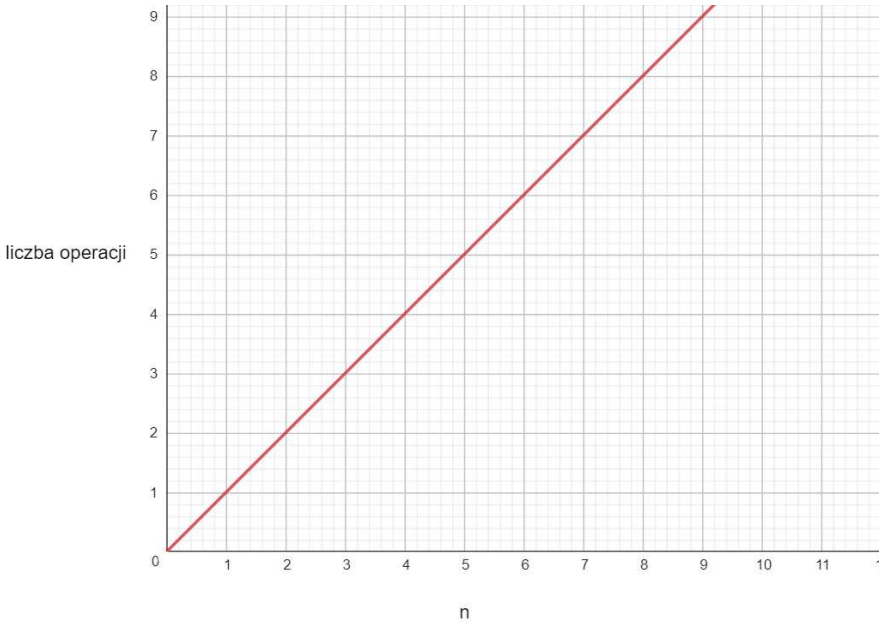
Uzupełnij.

[ ] < [ ] < [ ] < [ ] < [ ] < [ ] < [ ]

$n\ \log(n)$ | 2^n | $n!$ | n | $\log(n)$ | 1 | n^2

Ćwiczenie 4

Wskaż, jaki typ złożoności ilustruje wykres.



- złożoność logarytmiczna
- Nie ma takiego typu złożoności.
- złożoność stała
- złożoność liniowa

Ćwiczenie 5

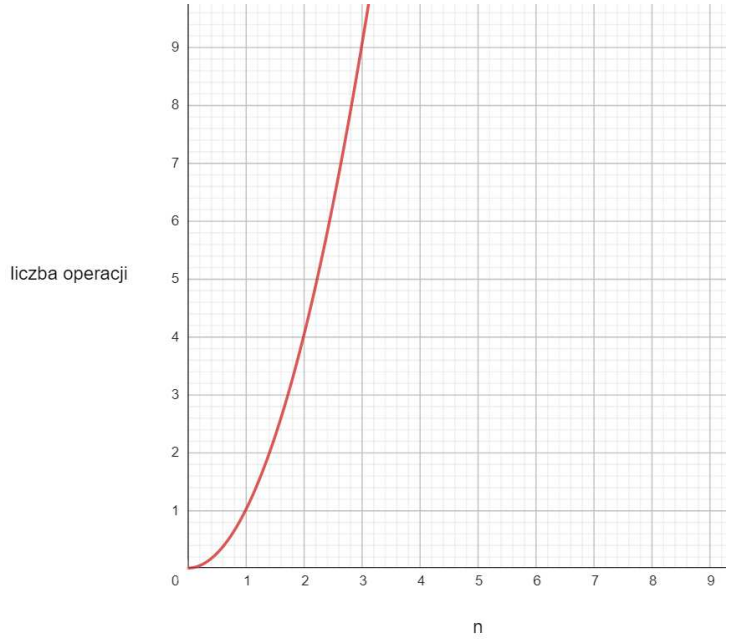
Zdefiniuj złożoność obliczeniową.

Ćwiczenie 6

Zdefiniuj złożoność czasową.

Ćwiczenie 7

Wskaż, jakiego typu złożoności **nie** ilustruje wykres.



- [ ] złożoność stała
- [ ] złożoność logarytmiczna
- [ ] złożoność kwadratowa
- [ ] złożoność liniowa

Ćwiczenie 8

Określ, ile operacji wykona poniższy algorytm dla zbioru {2, 7, 5, 2, 3}, który ma zostać posortowany nierosnąco.

```
dla i = 2, 3, 4 ... n wykonuj
    pom = tab[i]
    j = i-1
    dopóki j >= 1 oraz tab[j] > pom wykonuj
        tab[j + 1] = tab[j]
        j = j - 1
    tab[j + 1] = pom
```

Dla nauczyciela

Autor: Bartosz Zadrożny
Przedmiot: Informatyka

Temat: Jak porównywać złożoność obliczeniową?

Grupa docelowa:

Liceum ogólnokształcące i technikum, 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.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

4) porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;

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;

4) ilustruje i wyjaśnia rolę pojęć, obiektów i operacji matematycznych w projektowaniu rozwiązań problemów informatycznych i z innych dziedzin, posługuje się pojęciem logarytmu;

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):

- Zdefiniujesz operacje dominujące.
- Przeanalizujesz złożoność czasową algorytmów.
- Prześledzisz koncepcję pamięciowej złożoności obliczeniowej.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

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;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji);
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

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

Faza wstępna:

1. Nauczyciel sprawdza przygotowanie uczniów do lekcji.
2. Nauczyciel wyświetla temat oraz cele zajęć, omawiając lub ustalając razem z uczniami kryteria sukcesu.
3. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie analizują treści z sekcji „Przeczytaj" wyświetlone na tablicy.
2. **Praca z multimedium.** Nauczyciel czyta polecenie nr 1 z sekcji „Symulacja interaktywna". Prosi uczniów, aby wykonali je w parach. Następnie wybrana osoba prezentuje propozycję odpowiedzi, a pozostali uczniowie ustosunkowują się do niej. Nauczyciel w razie potrzeby uzupełnia ją, udziela też uczniom informacji zwrotnej.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1 oraz 5–9 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Na koniec zajęć nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…".

Praca domowa:

1. Uczniowie wykonują ćwiczenia 2–4 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie 2 z sekcji „Symulacja interaktywna".

Wskazówki metodyczne:

- Treści w sekcji „Symulacja interaktywna" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.