Pierwsza Ławka · zestaw działowy · liceum i technikum

Dział 12

Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm

Wyszukiwanie liniowe i binarne (z logarytmem), jednoczesne wyszukiwanie minimum i maksimum (parami i metodą dziel i zwyciężaj), sortowanie przez scalanie i szybkie, porównywanie efektywności algorytmów na gotowych implementacjach. Osiem kart: teoria z przykładami, ćwiczenia, utrwalenie w formule CKE, zadanie przy komputerze z plikami danych (ZIP), kartkówka i sprawdzian w grupach A i B – z kluczem i zasadami oceniania.

Informatyka | klasa 3 LO i technikum | Dział 12 | zakres rozszerzony (ZR)

	Karta	Rodzaj	Poziom	Zadania
1	**Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – najważniejsze wiadomości** Teoria, przykłady krok po kroku, rozgrzewka	Najważniejsze wiadomości	ZR	4 zad. · 8 pkt
2	**Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – ćwiczenia podstawowe** Ćwiczę umiejętności z działu	Ćwiczenia	ZR	6 zad. · 20 pkt
3	**Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – utrwalenie** Zadania w formule CKE: polecenia i punktacja jak w arkuszach maturalnych	Zadania typu maturalnego	ZR	6 zad. · 13 pkt
4	**Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – w praktyce, przy komputerze** Pliki danych w archiwum ZIP · wyniki w plikach wynikiN.txt	Ćwiczenia	ZR	1 zad. · 7 pkt
5	**Kartkówka: wyszukiwanie binarne i scalanie – grupa A** 15 minut · 7 pkt · bez komputera	Kartkówka	ZR	4 zad. · 7 pkt 15 min
6	**Kartkówka: wyszukiwanie binarne i scalanie – grupa B** 15 minut · 7 pkt · bez komputera	Kartkówka	ZR	4 zad. · 7 pkt 15 min

7	**Sprawdzian: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – grupa A** 45 minut · 20 pkt · bez komputera	Sprawdzian	ZR	8 zad. · 20 pkt 45 min
8	**Sprawdzian: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – grupa B** 45 minut · 20 pkt · bez komputera	Sprawdzian	ZR	8 zad. · 20 pkt 45 min

Jak korzystać z zestawu

- Karta 1 przypomina najważniejsze wiadomości i pokazuje przykłady rozwiązane krok po kroku (z kodem w Pythonie albo opisem kroków w aplikacji) – od niej warto zacząć.
- Karta 2 to ćwiczenia podstawowe, karta 3 – utrwalenie w formule zadań maturalnych CKE (ramka „Jak to sprawdza CKE").
- Karta 4 to praca przy komputerze: pliki danych są w archiwum ZIP dołączonym do zestawu, a odpowiedzi kontrolne dla plików *_przyklad (pliki o nazwie z dopiskiem „_przyklad") podano przy zadaniach. Wyniki zapisuje się w plikach wynikiN.txt – jak na maturze.
- Karty 5–6 to kartkówka (15 minut), karty 7–8 – sprawdzian (45 minut), obie w grupach A i B: te same typy zadań i ta sama punktacja, inne dane. Kartkówka i sprawdzian nie wymagają komputera.
- Klucz zawiera odpowiedzi, przykładowe programy, zasady oceniania w stylu CKE i numery wymagań podstawy. Osobny plik „Powtórka" zbiera teorię, typowe błędy i test „Sprawdź się".

Oznaczenia

ZP	zakres podstawowy
ZR	zakres rozszerzony (treści tylko dla ZR)
(0–2 pkt)	liczba punktów możliwych do zdobycia za zadanie
Zadanie 4.1.	wiązka zadań do wspólnego materiału źródłowego
Jak to sprawdza CKE	typ zadania i typowy błąd ze sprawozdań CKE
Na maturze	wymagania z podstawy programowej, których dotyczy temat

Wymagania z podstawy programowej (Dz.U. 2024 poz. 1019)

- **I+II.R1b** zapisuje za pomocą listy kroków lub pseudokodu i implementuje w wybranym języku programowania algorytmy poznane na wcześniejszych etapach oraz algorytmy: b) znajdowania określonego elementu w zbiorze uporządkowanym metodą binarnego wyszukiwania,
- **I+II.R1d** zapisuje za pomocą listy kroków lub pseudokodu i implementuje w wybranym języku programowania algorytmy poznane na wcześniejszych etapach oraz algorytmy: d) jednoczesnego wyszukiwania elementu najmniejszego i największego,
- **I+II.R1e** zapisuje za pomocą listy kroków lub pseudokodu i implementuje w wybranym języku programowania algorytmy poznane na wcześniejszych etapach oraz algorytmy: e) sortowania ciągu liczb przez scalanie,
- **I+II.R3a** objaśnia oraz porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przykłady problemów i algorytmów, w szczególności: a) wyszukiwanie elementów liniowe i przez połowienie (do znajdowania elementów w zbiorze, sortowania przez wstawianie, przybliżonego rozwiązywania równań),
- **I+II.R3c** objaśnia oraz porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przykłady problemów i algorytmów, w szczególności: c) metodę dziel i zwyciężaj (jednoczesne znajdowanie minimum i maksimum, sortowanie przez scalanie i szybkie),
- **I.R2** porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;
- **I.R5** objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;
- **I.R6** ilustruje i wyjaśnia rolę pojęć, obiektów i operacji matematycznych w projektowaniu rozwiązań problemów informatycznych i z innych dziedzin, posługuje się pojęciem logarytmu;

Materiał oryginalny – żadne zadanie nie jest kopią zadania CKE ani podręcznika; dane w plikach są fikcyjne. Podstawa programowa LO i technikum w brzmieniu z 2024 r. (Dz.U. 2024 poz. 1019). Wymagania: I+II.R1b, I+II.R1d, I+II.R1e, I+II.R3a, I+II.R3c, I.R2, I.R5, I.R6. Miejsce w programach nauczania: „Informatyka na czasie" (Nowa Era) ZR – algorytmy wyszukiwania i sortowania (nazwy serii podano tylko informacyjnie). Do użytku własnego i pracy z własnymi uczniami.

PIERWSZA ŁAWKA

Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – najważniejsze wiadomości

Teoria, przykłady krok po kroku, rozgrzewka

ZR </> KARTA 1/8

Imię i nazwisko: ..... Klasa: ..... Data: ..... Wynik: ..... / 8 pkt

Najważniejsze wiadomości · 8 pkt

Przypomnij sobie

- **Wyszukiwanie liniowe** porównuje x z kolejnymi elementami – pesymistycznie n porównań. Działa dla dowolnego ciągu.
- **Wyszukiwanie binarne (przez połowienie)** działa tylko w ciągu **uporządkowanym**: porównujemy x z elementem środkowym przedziału [lo, hi] i odrzucamy tę połowę, w której x być nie może (lo = s + 1 albo hi = s − 1). Przedział za każdym razem zmniejsza się co najmniej o połowę.
- **Logarytm**: $\log_2 n$ to wykładnik, do którego trzeba podnieść 2, aby otrzymać n ($\log_2 1024 = 10$). Pętla wyszukiwania binarnego obraca się pesymistycznie $\lfloor \log_2 n \rfloor + 1$ razy: dla n = 13 – 4 razy, dla miliona – 20, dla 40 milionów – 26. Podwojenie n dodaje tylko jeden obrót.
- W Pythonie moduł **bisect**: bisect_left(A, x) – pierwsza pozycja, na którą można wstawić x bez psucia porządku (przed równymi), bisect_right(A, x) – za równymi. Liczba wystąpień x w ciągu uporządkowanym to bisect_right − bisect_left. Samo wyszukiwanie – O(log n), ale insort (wstawianie do listy) – O(n).
- **Jednoczesne min i max**: dwa osobne przebiegi to 2(n − 1) porównań. Porównując elementy **parami** (mniejszy z pary z minimum, większy z maksimum) wykonujemy 3 porównania na 2 elementy – dla parzystego n: 3n/2 − 2. To samo daje metoda **dziel i zwyciężaj**: min i max połówek, potem 2 porównania łączące (dla n będącego potęgą 2: T(n) = 3n/2 − 2).
- **Sortowanie przez scalanie** (dziel i zwyciężaj): dzielimy ciąg na połowy, sortujemy każdą rekurencyjnie i scalamy. Scalenie ciągów długości a i b wymaga co najwyżej a + b − 1 porównań. Poziomów podziału jest około $\log_2 n$, na każdym scalamy razem n elementów – złożoność O(n log n) zawsze, także pesymistycznie. Wersja z ≤ przy scalaniu jest stabilna; potrzebuje dodatkowej pamięci na ciąg wynikowy.
- **Sortowanie szybkie** (I+II.R3c – objaśnić i porównać): wybieramy element osiowy, *podział* ustawia mniejsze od niego przed nim, a pozostałe za nim; potem sortujemy rekurencyjnie obie części (bez scalania). Średnio O(n log n), ale gdy element osiowy jest zawsze najmniejszy lub największy (np. ostatni element w ciągu już posortowanym), podziały są skrajnie nierówne – $O(n^2)$.

Algorytm	Przypadek pesymistyczny	Uwagi
wyszukiwanie liniowe	O(n)	dowolny ciąg
wyszukiwanie binarne	O(log n)	ciąg uporządkowany
sortowanie przez wstawianie / bąbelkowe	$O(n^2)$	proste, w miejscu
sortowanie przez scalanie	O(n log n)	dodatkowa pamięć, stabilne
sortowanie szybkie	$O(n^2)$, średnio O(n log n)	w miejscu, zależy od elementu osiowego

- **Efektywność (I.R5)** mierzymy liczbą *operacji dominujących* (np. porównań elementów) w zależności od rozmiaru danych n. Gdy algorytm $O(n^2)$ dla n = 10 000 działa 1 s, to dla 4 razy większych danych – około 16 s; algorytm O(n log n) – około 4,6 s ($4 \cdot \log 40\,000 / \log 10\,000 \approx 4{,}6$).

Przykłady z rozwiązaniem

Przykład 1. **W uporządkowanym ciągu A = 4, 9, 13, 18, 21, 27, 30, 36, 42, 47, 55, 61, 68 (indeksy 0–12) wyszukaj binarnie 47, a potem 20. Ile obrotów pętli?**

```
def szukaj(A, x):
    lo, hi = 0, len(A) - 1
    while lo <= hi:
        s = (lo + hi) // 2
        if A[s] == x:
            return s
        if A[s] < x:
            lo = s + 1
        else:
            hi = s - 1
    return -1
```

x = 47: lo = 0, hi = 12, s = 6, A[s] = 30; lo = 7, hi = 12, s = 9, A[s] = 47 → znaleziono na pozycji 9 po 2 obrotach.
x = 20: lo = 0, hi = 12, s = 6, A[s] = 30; lo = 0, hi = 5, s = 2, A[s] = 13; lo = 3, hi = 5, s = 4, A[s] = 21; lo = 3, hi = 3, s = 3, A[s] = 18; potem lo = 4 > hi = 3 → wynik −1 po 4 obrotach ($\lfloor \log_2 13 \rfloor + 1 = 4$ – więcej nie będzie).

Przykład 2. **Znajdź jednocześnie min i max ciągu 6, -2, 11, 4, 0, 9, 15, 3, 8, 1 metodą par. Ile porównań?**

```
def minmax(A):
    n = len(A)
    if n % 2 == 1:
        mn = mx = A[0]
        i = 1
    else:
        if A[0] < A[1]:
            mn, mx = A[0], A[1]
        else:
            mn, mx = A[1], A[0]
        i = 2
    while i < n:
        if A[i] < A[i + 1]:
            male, duze = A[i], A[i + 1]
        else:
            male, duze = A[i + 1], A[i]
        if male < mn:
            mn = male
        if duze > mx:
            mx = duze
        i = i + 2
    return mn, mx
```

Para (6, −2): 1 porównanie → mn = −2, mx = 6. Każda z 4 kolejnych par (11, 4), (0, 9), (15, 3), (8, 1): 1 porównanie w parze + 1 z mn + 1 z mx. Razem $1 + 4 \cdot 3 = 13$ porównań (dwa osobne przebiegi: 18). Wynik: min = -2, max = 15.

Przykład 3. **Scal ciągi uporządkowane 3, 10, 14, 22 oraz 5, 6, 19, 25, 31. Ile porównań?**

```
def scal(A, B):
    C = []
    i = j = 0
    while i < len(A) and j < len(B):
        if A[i] <= B[j]:
            C.append(A[i])
            i = i + 1
        else:
            C.append(B[j])
            j = j + 1
    return C + A[i:] + B[j:]

def sortuj(T):
    if len(T) <= 1:
        return T
    s = len(T) // 2
    return scal(sortuj(T[:s]), sortuj(T[s:]))
```

Porównujemy pierwsze nieprzepisane elementy: 3–5 → 3; 10–5 → 5; 10–6 → 6; 10–19 → 10; 14–19 → 14; 22–19 → 19; 22–25 → 22. Ciąg A się skończył – resztę B (25, 31) przepisujemy bez porównań. Razem 7 porównań (≤ 4 + 5 − 1 = 8). Wynik: 3, 5, 6, 10, 14, 19, 22, 25, 31.

Przykład 4. **Wykonaj podział z sortowania szybkiego dla ciągu 12, 3, 17, 8, 1, 14, 9 (element osiowy – ostatni, 9).**

```
def podziel(A, l, p):       # element osiowy: A[p]
    os = A[p]
    i = l
    for j in range(l, p):
        if A[j] < os:
            A[i], A[j] = A[j], A[i]
            i = i + 1
    A[i], A[p] = A[p], A[i]
    return i

def szybkie(A, l, p):
    if l < p:
        k = podziel(A, l, p)
        szybkie(A, l, k - 1)
        szybkie(A, k + 1, p)
```

j przechodzi po elementach 12, 3, 17, 8, 1, 14; mniejsze od 9 (3, 8, 1) trafiają kolejno na początek, na końcu 9 zamienia się z elementem na pozycji i = 3. Po podziale: 3, 8, 1, 9, 17, 14, 12. Element 9 stoi już na swoim miejscu; dalej sortujemy osobno 3, 8, 1 oraz 17, 14, 12.

Zadania

Zadanie 1. (0–3 pkt) ..... / 3

Uzupełnij.

1. $\log_2 64 =$ _____
2. $\log_2 1024 =$ _____
3. $2^{?} \geq 5000$ – najmniejszy wykładnik: _____

Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – ćwiczenia podstawowe

Ćwiczę umiejętności z działu

ZR </> KARTA 2/8

Imię i nazwisko: ..... Klasa: ..... Data: ..... Wynik: ..... / 20 pkt

Ćwiczenia · 20 pkt

Zadanie 1. (0–3 pkt) ..... / 3

Wykonaj funkcję szukaj(A, x) dla ciągu A = 2, 7, 11, 16, 20, 25, 33, 38, 44, 50, 59 (indeksy 0–10) i x = 12. Wpisz w kolejnych wierszach wartości lo, hi, s i A[s] przy każdym obrocie pętli, a w ostatnim wierszu – wynik funkcji.

```
def szukaj(A, x):
    lo, hi = 0, len(A) - 1
    while lo <= hi:
        s = (lo + hi) // 2
        if A[s] == x:
            return s
        if A[s] < x:
            lo = s + 1
        else:
            hi = s - 1
    return -1
```

lo	hi	s	A[s]
0	10		
wynik			

Miejsce na odpowiedź

Zadanie 2. (0–4 pkt) ..... / 4

Ile obrotów pętli wykona pesymistycznie funkcja szukaj dla ciągu długości n?

1. n = 100: _____ 2. n = 1000: _____ 3. n = 1 000 000: _____ 4. n = 1 000 000 000: _____

Zadanie 3. (0–4 pkt) ..... / 4

Dla A = [3, 5, 5, 5, 8, 12, 12, 20] podaj wyniki funkcji z modułu bisect.

1. bisect_left(A, 5) = _____
2. bisect_right(A, 5) = _____
3. bisect_left(A, 12) = _____
4. bisect_right(A, 9) = _____

Zadanie 4. (0–3 pkt) ..... / 3

Scal ciągi 1, 8, 9 oraz 2, 3, 4, 10. Podaj wynik i liczbę porównań. Ile porównań wymaga scalenie ciągów 5, 6, 7 oraz 1, 2, 3?

Miejsce na odpowiedź

Zadanie 5. (0–2 pkt) ..... / 2

Wykonaj funkcję podziel(A, 0, 5) z sortowania szybkiego dla A = 7, 15, 2, 11, 4, 10. Podaj ciąg po podziale i zwrócony indeks.

```
def podziel(A, l, p):        # element osiowy: A[p]
    os = A[p]
    i = l
    for j in range(l, p):
        if A[j] < os:
            A[i], A[j] = A[j], A[i]
            i = i + 1
    A[i], A[p] = A[p], A[i]
    return i
```

Miejsce na odpowiedź

Zadanie 6. (0–4 pkt) ..... / 4

Dopasuj algorytm do jego pesymistycznej złożoności. Jedna złożoność jest zbędna.

1. wyszukiwanie binarne	a) O(log n)
2. sortowanie przez scalanie	b) O(n)
3. sortowanie szybkie	c) O(n log n)
4. wyszukiwanie liniowe	d) O(n^2)
	e) O(2^n)

1 ____ 2 ____ 3 ____ 4 ____

Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – utrwalenie

Zadania w formule CKE: polecenia i punktacja jak w arkuszach maturalnych

ZR </> KARTA 3/8

Imię i nazwisko: ..... Klasa: ..... Data: ..... Wynik: ..... / 13 pkt

Zadania typu maturalnego · 13 pkt

Jak to sprawdza CKE

Efektywność algorytmu CKE sprawdza na gotowym kodzie: Informator o egzaminie z informatyki – przykładowe zad. 4.1 (najmniejsza i największa liczba operacji sortowania, 1 pkt) i 4.2 („Dokończ zdanie" o pesymistycznej złożoności, 1 pkt); 2024 zad. 2.1 „Cyfry" – tabela z liczbą wykonań instrukcji (2 pkt). Potęgi dwójki i liczenie „ile poziomów" – 2023 zad. 1.2 „Biblioteczka Adama" (tabela, 3 pkt). Zapis algorytmu – zawsze ze *Specyfikacją*; gdy polecenie wymaga liczby kroków rzędu log n, przeszukiwanie liniowe nie dostaje punktów za algorytm.

Zadanie 1. (0–1 pkt) ..... / 1

Oceń prawdziwość podanych zdań. Zaznacz P, jeśli zdanie jest prawdziwe, albo F – jeśli jest fałszywe.

```
def szukaj(A, x):
    lo, hi = 0, len(A) - 1
    while lo <= hi:
        s = (lo + hi) // 2
        if A[s] == x:
            return s
        if A[s] < x:
            lo = s + 1
        else:
            hi = s - 1
    return -1
```

1. Funkcja szukaj dla uporządkowanego ciągu 2000 liczb wykona pesymistycznie 11 obrotów pętli.	P	F
2. Jeśli liczba elementów ciągu wzrośnie dwukrotnie, pesymistyczna liczba obrotów pętli w szukaj wzrośnie dwukrotnie.	P	F
3. Wyszukiwanie binarne w ciągu nieuporządkowanym może nie znaleźć elementu, który w nim jest.	P	F

Zadanie 2. (0–1 pkt) ..... / 1

Dokończ zdanie. Zaznacz właściwą odpowiedź spośród podanych.
Sortowanie szybkie z elementem osiowym zawsze równym ostatniemu elementowi fragmentu wykona liczbę porównań rzędu n^2 dla ciągu

A. o długości będącej potęgą dwójki.
B. już uporządkowanego rosnąco.
C. losowo wymieszanego.
D. w którym element środkowy jest medianą.

Zadanie 6. (0–4 pkt) ..... / 4

Zapisz w pseudokodzie lub w wybranym języku programowania algorytm, który w ciągu A uporządkowanym niemalejąco znajdzie największy indeks i, dla którego A[i] = x, albo zwróci −1, gdy x w ciągu nie ma. Algorytm ma wykonać liczbę porównań rzędu log n.

Przykład: dla A = 2, 4, 4, 4, 7, 9 i x = 4 wynikiem jest 3, a dla x = 5 wynikiem jest −1.

Specyfikacja

Dane: n – liczba całkowita dodatnia; A[0..n−1] – ciąg liczb całkowitych uporządkowany niemalejąco; x – liczba całkowita

Wynik: i – największy indeks, dla którego A[i] = x, albo −1

Miejsce na odpowiedź

UWAGA Rozwiązanie przeglądające ciąg element po elemencie (liczba porównań rzędu n) nie spełnia warunku zadania.

PIERWSZA ŁAWKA

Zadanie 6. (0–4 pkt) ..... / 4

Dla A = 20, 5, 16, 2, 25, 8, 13 wykonaj funkcję podziel(A, 0, 6) z sortowania szybkiego (element osiowy – ostatni). Podaj ciąg po podziale i zwrócony indeks. Następnie wyjaśnij, dlaczego dla ciągu już uporządkowanego rosnąco to sortowanie wykonuje liczbę porównań rzędu n^2.

```
def podziel(A, l, p):          # element osiowy: A[p]
    os = A[p]
    i = l
    for j in range(l, p):
        if A[j] < os:
            A[i], A[j] = A[j], A[i]
            i = i + 1
    A[i], A[p] = A[p], A[i]
    return i
```

Miejsce na odpowiedź

Zadanie 7. (0–3 pkt) ..... / 3

Uporządkowany słownik ma 500 000 haseł. Wyjaśnij, korzystając z pojęcia logarytmu, ile najwyżej porównań z hasłem środkowym wykona wyszukiwanie binarne, i porównaj to z wyszukiwaniem liniowym.

Miejsce na odpowiedź

CO JESZCZE JEST W ŚRODKU

Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – kartkówka, sprawdzian i karty pracy z…

Podgląd pokazał 11 z 34 stron. Pełna wersja: 34 stron A4, PDF do druku i DOCX do edycji.

- Karta 1: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – najważniejsze wiadomości – 4 zadania, 8 pkt
- Karta 2: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – ćwiczenia podstawowe – 6 zadań, 20 pkt
- Karta 3: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – utrwalenie – 6 zadań, 13 pkt
- Karta 4: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – w praktyce, przy komputerze – 1 zadanie, 7 pkt
- Karta 5: Kartkówka: wyszukiwanie binarne i scalanie – grupa A – 4 zadania, 7 pkt
- Karta 6: Kartkówka: wyszukiwanie binarne i scalanie – grupa B – 4 zadania, 7 pkt
- Karta 7: Sprawdzian: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – grupa A – 8 zadań, 20 pkt
- Karta 8: Sprawdzian: Wyszukiwanie i sortowanie: binarne, min-max jednocześnie, przez scalanie, szybkie; efektywność i logarytm – grupa B – 8 zadań, 20 pkt Karta 1 przypomina teorię

Kup pełną wersję – plik od razu na e-mail

pierwszalawka.pl/material/informatyka-liceum-klasa-3-zr-dzial-12-wyszukiwanie-i-sortowanie-binarne-min-max