SPRAWDZIAN · INFORMATYKA

Sprawdzian z algorytmiki **Grupa A**

Informatyka: algorytmy klasyczne, rekurencja, techniki projektowania i złożoność

Imię i nazwisko: _____

Klasa: _____

Data: _____

Punkty: _____ / 28

10 zadań, 28 pkt. Czytaj uważnie polecenia. W zadaniach otwartych zapisz obliczenia.

Zadanie 1. (0–1 pkt)

Algorytm sortowania przez scalanie (mergesort) tablicy n elementów ma złożoność pesymistyczną:

(A) $O(n^2)$
(B) $O(n \log n)$
(C) $O(n)$
(D) $O(\log n)$

Zaznacz jedną poprawną odpowiedź.

Zadanie 2. (0–1 pkt)

Problem plecakowy 0-1 (knapsack) dla n przedmiotów i pojemności W ma złożoność w podejściu programowania dynamicznego:

(A) $O(2^n)$
(B) $O(n!)$
(C) $O(n^2)$
(D) $O(n \cdot W)$

Zaznacz jedną poprawną odpowiedź.

Zadanie 3. (0–3 pkt)

Oceń prawdziwość stwierdzeń dotyczących algorytmów sortowania:

1. Algorytm QuickSort z pivotem zawsze wybieranym jako pierwszy element tablicy dla danych już posortowanych działa w czasie O(n^2).	**P**	**F**
2. Merge Sort jest algorytmem stabilnym o złożoności pesymistycznej O(n log n).	**P**	**F**
3. Algorytm sortowania jest stabilny, jeśli elementy o równych kluczach zachowują swój pierwotny wzajemny porządek po sortowaniu.	**P**	**F**

Przy każdym zdaniu zakreśl P (prawda) albo F (fałsz).

Zadanie 4. (0–3 pkt)

Oceń prawdziwość stwierdzeń o algorytmach sortowania:

1. Sortowanie pozycyjne (radix sort) ma złożoność $O(n \cdot k)$, gdzie k to liczba cyfr.	**P**	**F**
2. Żaden algorytm sortowania przez porównania nie może działać szybciej niż $O(n \log n)$ w pesymistycznym przypadku.	**P**	**F**
3. Sortowanie przez kopcowanie (heapsort) wymaga dodatkowej pamięci $O(n)$.	**P**	**F**

Przy każdym zdaniu zakreśl P (prawda) albo F (fałsz).

Zadanie 5. (0–4 pkt)

Dopasuj technikę algorytmiczną do problemu, który rozwiązuje najefektywniej.

1. Programowanie zachłanne	...	A. Sortowanie przez scalanie
2. Przeszukiwanie z powrotami (backtracking)	...	B. Generowanie wszystkich permutacji
3. Dziel i zwyciężaj (divide & conquer)	...	C. Problem wydawania reszty (minimum monet)
4. Programowanie dynamiczne	...	D. Minimalne drzewo rozpinające (Kruskal)

Przy każdym numerze wpisz literę pasującego elementu.

Zadanie 6. (0–4 pkt)

Dopasuj algorytm do jego pesymistycznej złożoności czasowej:

1. Bubble sort	...	A. $O(n^2)$
2. Merge sort	...	B. $O(\log n)$
3. Wyszukiwanie binarne	...	C. $O(n)$
4. Wyszukiwanie liniowe	...	D. $O(n \log n)$

Przy każdym numerze wpisz literę pasującego elementu.

Zadanie 7. (0–3 pkt)

Wyjaśnij, czym jest notacja Big O. Podaj przykłady algorytmów o złożonościach $O(1)$, $O(\log n)$, $O(n)$ i $O(n^2)$.

Zadanie 8. (0–3 pkt)

Wyjaśnij, czym jest algorytm zachłanny (greedy). Podaj jeden przykład problemu, do którego rozwiązania stosuje się podejście zachłanne, i wyjaśnij, na czym polega podejmowanie decyzji w tym podejściu.

Zadanie 9. (0–3 pkt)

Wyjaśnij zasadę działania algorytmu wyszukiwania binarnego. Podaj jego złożoność czasową w najgorszym przypadku i warunek wstępny, który musi spełniać zbiór danych.

Zadanie 10. (0–3 pkt)

Wyjaśnij, na czym polega technika programowania dynamicznego i czym różni się od zwykłej rekurencji wspomaganej memoizacją pod względem sposobu budowania rozwiązania. Podaj jeden przykład problemu, do którego rozwiązania typowo stosuje się programowanie dynamiczne.

Rozwiąż ten sprawdzian online, z wyjaśnieniem każdej odpowiedzi: matury-online.pl/sprawdzian/informatyka/algorytmika

SPRAWDZIAN · INFORMATYKA

Sprawdzian z algorytmiki **Grupa B**

Informatyka: algorytmy klasyczne, rekurencja, techniki projektowania i złożoność

Imię i nazwisko: _____

Klasa: _____

Data: _____

Punkty: _____ / 28

10 zadań, 28 pkt. Czytaj uważnie polecenia. W zadaniach otwartych zapisz obliczenia.

Zadanie 1. (0–1 pkt)

Przeszukiwanie grafu w głąb (DFS) wykorzystuje strukturę danych:

(A) Kolejkę (FIFO)
(B) Kopiec (heap)
(C) Stos (LIFO)
(D) Tablicę haszującą

Zaznacz jedną poprawną odpowiedź.

Zadanie 2. (0–1 pkt)

Jaka jest złożoność czasowa sortowania przez wstawianie (insertion sort) w najlepszym przypadku, gdy dane wejściowe są już posortowane rosnąco?

(A) $O(n^2)$
(B) $O(n \log n)$
(C) $O(n)$
(D) $O(\log n)$

Zaznacz jedną poprawną odpowiedź.

Zadanie 3. (0–3 pkt)

Oceń prawdziwość stwierdzeń o algorytmie wyszukiwania binarnego:

1. Dla tablicy 1 000 000 elementów potrzeba najwyżej 20 porównań.	**P**	**F**
2. Wyszukiwanie binarne wymaga, aby dane były posortowane.	**P**	**F**
3. Na liście wiązanej jednokierunkowej wyszukiwanie binarne działa w czasie $O(\log n)$, tak samo jak w tablicy.	**P**	**F**

Przy każdym zdaniu zakreśl P (prawda) albo F (fałsz).

Zadanie 4. (0–3 pkt)

Oceń prawdziwość stwierdzeń o algorytmach grafowych:

1. DFS nie zawsze znajduje najkrótszą ścieżkę w grafie nieważonym.	**P**	**F**
2. BFS zawsze znajduje najkrótszą ścieżkę w grafie nieważonym.	**P**	**F**
3. Algorytm Kruskala buduje minimalne drzewo rozpinające.	**P**	**F**

Przy każdym zdaniu zakreśl P (prawda) albo F (fałsz).

Zadanie 5. (0–4 pkt)

Dopasuj problem do techniki algorytmicznej:

1. Wydawanie reszty nominałami (wariant uproszczony)	…	A. Algorytm Dijkstry
2. Sortowanie przez scalanie	…	B. Algorytm zachłanny (greedy)
3. Najkrótsza ścieżka w grafie ważonym	…	C. Programowanie dynamiczne
4. Ciąg Fibonacciego efektywnie	…	D. Dziel i zwyciężaj (divide and conquer)

Przy każdym numerze wpisz literę pasującego elementu.

Zadanie 6. (0–4 pkt)

Dopasuj algorytm do jego typowego zastosowania.

1. Algorytm Euklidesa	…	A. Wyznaczanie najkrótszej ścieżki w grafie ważonym o nieujemnych wagach
2. Algorytm Dijkstry	…	B. Wyznaczanie największego wspólnego dzielnika dwóch liczb
3. Sortowanie topologiczne	…	C. Ustalanie kolejności wykonania zadań powiązanych zależnościami w grafie skierowanym acyklicznym
4. Algorytm zachłanny (greedy)	…	D. Podejmowanie w każdym kroku decyzji lokalnie najlepszej, bez cofania się do wcześniejszych wyborów

Przy każdym numerze wpisz literę pasującego elementu.

Zadanie 7. (0–3 pkt)

Wyjaśnij różnicę między pesymistyczną (worst-case), oczekiwaną (average-case) i optymistyczną (best-case) złożonością czasową algorytmu. Podaj przykład algorytmu, dla którego te trzy złożoności się różnią.

Zadanie 8. (0–3 pkt)

Opisz działanie algorytmu wyszukiwania binarnego (binary search) w posortowanej tablicy. Podaj jego złożoność obliczeniową oraz warunek konieczny do jego zastosowania.

Zadanie 9. (0–3 pkt)

Wyjaśnij, dlaczego pojedyncze dodanie elementu na koniec dynamicznej tablicy (np. `list.append` w Pythonie), która w razie potrzeby podwaja swoją pojemność, ma zamortyzowaną złożoność czasową $O(1)$, mimo że pojedyncze powiększenie tablicy kosztuje $O(n)$.

Zadanie 10. (0–3 pkt)

Wyjaśnij, dlaczego algorytm sortowania przez scalanie (merge sort) ma złożoność czasową $O(n \log n)$.

Rozwiąż ten sprawdzian online, z wyjaśnieniem każdej odpowiedzi: matury-online.pl/sprawdzian/informatyka/algorytmika

Sprawdzian z algorytmiki

Grupa A

1. (1 pkt) **B**

2. (1 pkt) **D**

3. (3 pkt) **1. P; 2. P; 3. P** — po 1 pkt za zdanie

4. (3 pkt) **1. P; 2. P; 3. F** — po 1 pkt za zdanie

5. (4 pkt) **1–D, 2–B, 3–A, 4–C** — po 1 pkt za parę

6. (4 pkt) **1–A, 2–D, 3–B, 4–C** — po 1 pkt za parę

7. (3 pkt) Przykładowa odpowiedź: Notacja Big O opisuje górne ograniczenie złożoności obliczeniowej algorytmu względem rozmiaru danych n, ignorując stałe mnożniki i mniej istotne składniki. Przykłady: O(1) — dostęp do elementu tablicy po indeksie (czas stały); O(log n) — wyszukiwanie binarne (dzielenie n na pół); O(n) — liniowe przeszukiwanie tablicy; $O(n^2)$ — sortowanie bąbelkowe lub przez wstawianie (dwie zagnieżdżone pętle). Algorytmy O(log n) i O(n) są bardzo wydajne w praktyce; $O(n^2)$ staje się problematyczne już dla n > 10 000.
Punktacja: 1. Definicja: Big O opisuje górne ograniczenie czasu/pamięci algorytmu jako funkcję rozmiaru danych, ignorując stałe i niższe rzędy (1 pkt). 2. Przykład O(1) i O(log n) (0,5 pkt). 3. Przykład O(n) i $O(n^2)$ (0,5 pkt). 4. Wyjaśnienie co oznacza każda klasa w praktyce (1 pkt).

8. (3 pkt) Przykładowa odpowiedź: Algorytm zachłanny w każdym kroku podejmuje decyzję, która wydaje się najlepsza (lokalnie optymalna) w danej chwili, bez analizowania wpływu tej decyzji na przyszłe kroki i bez cofania się do wcześniejszych wyborów. Przykładem jest problem wydawania reszty najmniejszą liczbą monet: w każdym kroku algorytm wybiera największy nominał monety, który nie przekracza pozostałej kwoty do wydania, i powtarza tę decyzję, aż reszta zostanie w pełni wydana.
Punktacja: Poprawne wyjaśnienie idei algorytmu zachłannego — podejmowanie w każdym kroku decyzji lokalnie optymalnej, bez cofania się (1 pkt) + poprawny przykład problemu, np. wydawanie reszty najmniejszą liczbą monet, minimalne drzewo rozpinające (1 pkt) + poprawny opis mechanizmu podejmowania decyzji dla podanego przykładu (1 pkt). Maksymalnie 3 pkt.

9. (3 pkt) Przykładowa odpowiedź: Algorytm wyszukiwania binarnego działa na posortowanym zbiorze. Na każdym kroku porównuje szukany element ze środkowym elementem aktualnego zakresu. Jeśli środkowy jest szukany — koniec. Jeśli szukany jest mniejszy — kontynuuje w lewej połowie; jeśli większy — w prawej. Każde porównanie eliminuje połowę pozostałych elementów. Złożoność: O(log n). Warunek konieczny: zbiór musi być posortowany.
Punktacja: 1. Opis: porównanie szukanego elementu ze środkowym, wybór właściwej połowy, powtarzanie (1 pkt). 2. Złożoność O(log n) (1 pkt). 3. Warunek: dane muszą być posortowane (1 pkt). Łącznie 3 pkt.

10. (3 pkt) Przykładowa odpowiedź: Programowanie dynamiczne polega na rozwiązywaniu problemu przez podzielenie go na mniejsze, nakładające się podproblemy i budowanie rozwiązania „od dołu do góry" — najpierw oblicza się rozwiązania najmniejszych podproblemów, zapisuje je w tablicy, a następnie na ich podstawie wylicza się rozwiązania coraz większych podproblemów, aż do uzyskania rozwiązania całego problemu. Różni się to od rekurencji z memoizacją, która działa „od góry do dołu" — zaczyna od pełnego problemu i cofa się do podproblemów, zapamiętując wyniki w miarę ich obliczania, ale nadal korzysta z wywołań rekurencyjnych i związanego z nimi narzutu na stos wywołań. Klasycznym przykładem zastosowania programowania dynamicznego jest efektywne obliczanie n-tego wyrazu ciągu Fibonacciego w czasie liniowym zamiast wykładniczego.

Grupa B

1. (1 pkt) **C**

2. (1 pkt) **C**

3. (3 pkt) **1. P; 2. P; 3. F** — po 1 pkt za zdanie

4. (3 pkt) **1. P; 2. P; 3. P** — po 1 pkt za zdanie

5. (4 pkt) **1–B, 2–D, 3–A, 4–C** — po 1 pkt za parę

6. (4 pkt) **1–B, 2–A, 3–C, 4–D** — po 1 pkt za parę

7. (3 pkt) Przykładowa odpowiedź: Złożoność pesymistyczna (worst-case) to najdłuższy możliwy czas działania algorytmu dla danych wejściowych o danym rozmiarze — opisuje najgorszy możliwy scenariusz. Złożoność oczekiwana (average-case) to średni czas działania dla typowych, losowych danych wejściowych. Złożoność optymistyczna (best-case) to najkrótszy możliwy czas, osiągany dla najbardziej sprzyjających danych. Klasycznym przykładem jest quicksort: w przypadku optymistycznym i przeciętnym ma złożoność O(n log n), ale w przypadku pesymistycznym (np. gdy tablica jest już posortowana, a pivotem jest zawsze pierwszy element) złożoność rośnie do $O(n^2)$.
Punktacja: 1 pkt — poprawne wyjaśnienie trzech pojęć (worst-case: najdłuższy możliwy czas dla danych wejściowych o danym rozmiarze; average-case: oczekiwany czas dla typowych/losowych danych; best-case: najkrótszy możliwy czas). 1 pkt — trafny przykład algorytmu, dla którego złożoności te się różnią (np. quicksort). 1 pkt — poprawne uzasadnienie różnicy na przykładzie (np. quicksort ma O(n log n) w best/average case, ale $O(n^2)$ w worst case przy niekorzystnym wyborze pivota). Łącznie 3 pkt.

8. (3 pkt) Przykładowa odpowiedź: Wyszukiwanie binarne porównuje szukaną wartość z elementem znajdującym się w środku posortowanej tablicy. Jeśli wartości są równe, algorytm kończy działanie. Jeśli szukana wartość jest mniejsza, poszukiwania kontynuowane są w lewej połowie tablicy, jeśli większa — w prawej połowie. Krok ten powtarza się, aż przedział poszukiwań będzie pusty lub element zostanie znaleziony. Ponieważ w każdym kroku przedział zmniejsza się o połowę, złożoność czasowa wynosi O(log n). Warunkiem koniecznym zastosowania algorytmu jest wcześniejsze posortowanie tablicy — dla danych nieuporządkowanych metoda nie działa poprawnie.
Punktacja: Poprawny opis zasady działania — porównanie elementu środkowego z szukaną wartością i zawężanie przedziału poszukiwań o połowę (1 pkt) + poprawnie podana złożoność O(log n) z uzasadnieniem (1 pkt) + wskazanie warunku: tablica musi być posortowana (1 pkt). Razem 3 pkt.

9. (3 pkt) Przykładowa odpowiedź: Gdy dynamiczna tablica jest pełna, dodanie kolejnego elementu wymaga zaalokowania nowego, większego bloku pamięci (zwykle o podwojonej pojemności) i skopiowania do niego wszystkich dotychczasowych n elementów, co kosztuje O(n). Takie kosztowne powiększenia zdarzają się jednak coraz rzadziej — po powiększeniu tablica ma miejsce na kolejne n tanich dodawań O(1) każde, zanim znów będzie trzeba ją powiększyć. Sumując koszt wszystkich operacji append wykonanych od pustej tablicy do rozmiaru n, koszty powiększeń tworzą szereg geometryczny w przybliżeniu $1 + 2 + 4 + \cdots + n \approx 2n$, więc łączny koszt n operacji jest rzędu O(n). Dzieląc ten łączny koszt przez liczbę operacji n, otrzymujemy średni, zamortyzowany koszt pojedynczej operacji równy O(1).
Punktacja: 1 pkt za wyjaśnienie mechanizmu podwajania pojemności przy przepełnieniu. 1 pkt za wskazanie, że kosztowne powiększenia zdarzają się coraz rzadziej. 1 pkt za poprawne uzasadnienie amortyzacji (sumaryczny koszt n operacji podzielony przez n daje wartość stałą). Maks. 3 pkt.

10. (3 pkt) Przykładowa odpowiedź: Tablicę dzieli się na połowy, aż zostaną fragmenty jednoelementowe, co daje około $\log_2 n$ poziomów rekursji. Na każdym poziomie scalanie wszystkich fragmentów kosztuje łącznie rzędu n porównań i przepisań. Razem jest więc $n \cdot \log_2 n$ operacji, czyli złożoność $O(n \log n)$.
Punktacja: Podział tablicy na połowy aż do jednoelementowych, czyli ok. $\log_2 n$ poziomów rekursji (1 pkt). Scalanie na każdym poziomie wymaga łącznie rzędu n operacji (1 pkt). Połączenie obu faktów: liczba poziomów razy koszt poziomu daje $n \log n$ (1 pkt). 3 pkt.

Skala ocen (orientacyjna, 28 pkt w każdej grupie)

bardzo dobry (5)	26–28 pkt
dobry (4)	21–25 pkt
dostateczny (3)	14–20 pkt
dopuszczający (2)	9–13 pkt
niedostateczny (1)	0–8 pkt

Progi: 90%, 75%, 50%, 30%. Progi w Twojej szkole mogą być inne. W zadaniach otwartych uznaj każdą odpowiedź poprawną merytorycznie, także sformułowaną inaczej niż przykład.

Ten sprawdzian online, z wyjaśnieniem każdej odpowiedzi: **matury-online.pl/sprawdzian/informatyka/algorytmika**. Nauczyciele i korepetytorzy ułożą własną kartę z banku ponad 20 tys. zadań: **matury-online.pl/dla-korepetytorow**.