Analiza algorytmów — zadania podstawowe

Zadanie 1 Zliczanie

```
ZLICZAJ(n)
1  r ← 0
2  for i ← 1 to n − 1
3      do for j ← i + 1 to n
4             do for k ← 1 to j
5                    do r ← r + 1
6  return r
```

✎ Jaka wartość zostanie zwrócona przez powyższą funkcję? Wyraź odpowiedź jako funkcję zmiennej n.

Zadanie 2 Mnożenie

Poniższy algorytm wyznacza yz, gdzie $y, z \in \mathbb{N}$.

```
MNÓŻ(y, z)
1  x ← 0
2  while z > 0
3      do if z mod 2 = 1
4            then x ← x + y
5         y ← 2 · y
6         z ← ⌊z/2⌋
7  return x
```

✎ Określ ile razy zostanie wykonane dodawanie (instrukcja w wierszu 4) w przypadku pesymistycznym.

Zadanie 3 Potęgowanie powolne

Poniższy algorytm wyznacza y^z, gdzie $y \in \mathbb{R}$, $z \in \mathbb{N}$.

```
POTĘGA(y, z)
1  x ← 1
2  while z > 0
3      do x ← x · y
4         z ← z − 1
5  return x
```

✎ Określ ile razy zostanie wykonane mnożenie (instrukcja w wierszu 3) w przypadku pesymistycznym.

Zadanie 4 Potęgowanie szybkie

Poniższy algorytm wyznacza y^z, gdzie $y \in \mathbb{R}$, $z \in \mathbb{N}$.

```
POTĘGA(y, z)
1  x ← 1
2  while z > 0
3      do if odd(z)
4             then x ← x · y
5          z ← ⌊z/2⌋
6          y ← y^2
7  return x
```

✎ Określ ile razy zostanie wykonane mnożenie (instrukcja w wierszu 4) w przypadku pesymistycznym.

Zadanie 5 Dzielenie

Poniższy algorytm wyznacza $q, r \in \mathbb{N}$ takie, że $y = qz + r$ oraz $r < z$, gdzie $y, z \in \mathbb{N}$.

```
DZIEL(y, z)
 1  r ← y
 2  q ← 0
 3  w ← z
 4  while w ⩽ y
 5      do w ← 2w
 6  while w > z
 7      do q ← 2q
 8         w ← ⌊w/2⌋
 9         if w ⩽ r
10            then r ← r − w
11                 q ← q + 1
12  return (q, r)
```

✎ Określ ile razy zostanie wykonane odejmowanie (instrukcja w wierszu 10) w przypadku pesymistycznym.

Zadanie 6 Schemat Hornera

Poniższy algorytm wyznacza wartość wielomianu $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ w punkcie x. Znaczy to, że zwracana jest wartość $\sum_{i=1}^{n} A[i] \cdot x^i$, zakładając, że w tablicy $A[0..n]$ przechowywane są współczynniki $a_i = A[i]$ dla wszystkich $0 \leqslant i \leqslant n$. Nazwa algorytmu pochodzi od nazwiska jego autora Williama G. Hornera.

```
HORNER(A, n, x)
1  v ← 0
2  for i ← n downto 0
3      do v ← A[i] + v · x
4  return v
```

✎ Jaka jest jego złożoność obliczeniowa?

Zadanie 7 Silnia

Poniższy algorytm wyznacza $n!$, gdzie $n \in \mathbb{N}$.

```
SILNIA(n)
1  x ← 1
2  while n > 1
3      do x ← x · n
4         n ← n − 1
5  return x
```

✎ Określ ile razy zostanie wykonane mnożenie (instrukcja w wierszu 3).

Zadanie 8 Maksimum

Załóżmy, że w tablicy $A[1..n]$ rozmieszczono n różnych liczb w sposób losowy. Poniższy algorytm znajduje największą z nich.

```
MAX(A, n)
1  m ← A[1]
2  for i ← 2 to n
3      do if A[i] > m
4            then m ← A[i]
5  return m
```

✎ Określ ile razy zostaną wykonane instrukcje przypisania (instrukcja w wierszu 1 i instrukcja w wierszu 4) w przypadku optymistycznym, pesymistycznym i średnim.

Zadanie 9 Para elementów

✎ Dana jest tablica $A[1..n]$ zawierająca n liczb. Zaprojektuj algorytm sprawdzający, czy w tablicy A są dwie liczby dające w sumie wartość x, a następnie określ jego złożoność obliczeniową.

Zadanie 10 Sortowanie bąbelkowe

Poniższy algorytm sortuje elementy tablicy $A[1..n]$.

```
BUBBLE-SORT(A, n)
1  for i ← 1 to n − 1
2      do for j ← 1 to n − i
3             do if A[j] > A[j + 1]
4                    then Zamień A[j] z A[j + 1]
```

✎ Określ ile razy zostaną porównane elementy tablicy (instrukcja warunkowa w wierszu 3).

Zadanie 11 Dopasowanie wzorca

Dane są łańcuch $S[1..n]$ i wzorzec $P[0\,..\,m-1]$, gdzie $1 \leqslant m \leqslant n$. Poniższy algorytm wyznacza pozycję ℓ występowania wzorca P w łańcuchu S, tzn. $\ell = p$ jeśli $S[p\,..\,p+m-1] = P$, a $\ell = n - m + 1$ jeśli wzorzec P nie jest podciągiem S.

```
DOPASUJ(P, S, m, n)
 1  ℓ ← 0
 2  dopasowano ← false
 3  while ℓ ⩽ n − m ∧ ¬dopasowano
 4      do ℓ ← ℓ + 1
 5         r ← 0
 6         dopasowano ← true
 7         while r < m ∧ dopasowano
 8             do dopasowano ← (P[r] = S[ℓ + r])
 9                r ← r + 1
10  return ℓ
```

✎ Ile porównań symboli łańcucha i wzorca (instrukcji w wierszu 8) wykonuje powyższy algorytm w przypadku pesymistycznym?

Zadanie 12 Rekurencja

```
G(n)
1  if n ⩽ 1
2     then return n
3     else return 5 · G(n − 1) − 6 · G(n − 2)
```

✎ Wykaż, że powyższy algorytm zwraca wartość $3^n - 2^n$ dla wszystkich $n \geqslant 0$.

✎ Pokaż, że algorytm ten działa w czasie $O(2^n)$.

Zadanie 13 Mnożenie rekurencyjne

Poniższy algorytm wyznacza yz, gdzie $y, z \in \mathbb{N}$.

```
MNÓŻ(y, z)
1  if z = 0
2     then return 0
3     else  if odd(z)
4              then return MNÓŻ(2 · y, ⌊z/2⌋) + y
5              else  return MNÓŻ(2 · y, ⌊z/2⌋)
```

✎ Jaka jest jego złożoność obliczeniowa?

Zadanie 14 Jeszcze raz rekurencja

```
G(n)
1  if n = 0 ∨ n = 1
2     then return 3 · n
3     else  return G(n − 1) + 2 · G(n − 2)
```

✎ Wykaż, że powyższy algorytm zwraca wartość $2^n - (-1)^n$ dla wszystkich $n \geqslant 0$.

Zadanie 15 Sortowanie przez proste wybieranie

Sortowanie przez proste wybieranie odbywa się w następujący sposób: trzeba wyznaczyć najmniejszy element w tablicy, zamienić go miejscami z pierwszym elementem, wyznaczyć najmniejszy element w $A[2..n]$ i zamienić go z drugim elementem itd., aż cała tablica zostanie posortowana.

```
SELECTION-SORT(A, n)
1  for i ← 1 to n − 1
2     do m ← i
3        for j ← i + 1 to n
4           do if A[j] < A[m]
5                 then m ← j
6              zamień A[m] z A[i]
```

✎ Określ złożoność obliczeniową tej metody sortowania.

Zadanie 16 Optymalny podział

Załóżmy, że pewien algorytm wykonuje m^2 kroków dla m-elementowej tablicy (dla dowolnego $m \geqslant 1$). Algorytm ten ma być użyty do tablic A_1 i A_2. Tablice zawierają łączną liczbę n elementów. A_1 ma k elementów, a A_2 ma $n - k$ elementów ($0 \leqslant k \leqslant n$).

✎ Dla jakiej wartości k obliczenia będą trwały najkrócej? Uzasadnij swoją odpowiedź.

Zadanie 17 Sortowanie przez proste wstawianie

Sortowanie tablicy $A[1..n]$ przez proste wstawianie odbywa się w następujący sposób: niech x będzie elementem drugim, potem trzecim itd., aż do ostatniego (n). Elementy stojące po lewej stronie x są już uporządkowane i należy x wstawić w odpowiednie miejsce w ciągu $A[1..j]$, gdzie $x = A[j]$.

```
INSERTION-SORT(A, n)
1  for j ← 2 to n
2      do x ← A[j]
3         i ← j − 1
4         while i > 0 ∧ A[i] > x
5             do A[i + 1] ← A[i]
6                i ← i − 1
7         A[i + 1] ← x
```

✎ Określ złożoność obliczeniową tej metody sortowania.

Zadanie 18 Wyszukiwanie binarne

Dana jest posortowana, n-elementowa tablica A oraz wartość v. Poniższy algorytm jako wynik działania podaje indeks p taki, że $v = A[p]$ lub NIL jeśli $v \notin A$. Zakładamy, że wywołano go z parametrami SZUKAJ$(A, 1, n)$.

```
SZUKAJ(A, p, r)
1  if p < r
2    then q ← ⌊(p + r − 1)/2⌋
3         if v ⩽ A[q]
4           then SZUKAJ(A, p, q)
5           else SZUKAJ(A, q + 1, r)
6    else if v = A[p]
7           then return p
8           else return NIL
```

✎ Wykaż, że algorytm ten dokonuje logarytmicznej liczby porównań.

Zadanie 19 Powtórzenie słowa

✎ Skonstruuj algorytm sprawdzania, czy dany tekst zaczyna się słowem postaci ww. Następnie określ optymistyczną i pesymistyczną złożoność obliczeniową tego algorytmu.

Zadanie 20 Indeks równy elementowi

✎ Niech $A[1..n]$ będzie posortowaną tablicą parami różnych liczb całkowitych. Zaprojektuj algorytm działający na zasadzie „dziel i zwycię-

żaj", który znajduje indeks i taki, że $A[i] = i$ (jeśli takowy istnieje) i działa w czasie $O(\log n)$.

Zadanie 21 Nieefektywne sortowanie

Rozważmy następujący algorytm sortowania.

```
STOOGE-SORT(A, i, j)
1  if A[i] > A[j]
2     then zamień A[i] z A[j]
3  if i + 1 ⩾ j
4     then return
5  k ← ⌊(j − i + 1)/3⌋
6  STOOGE-SORT(A, i, j − k)   // pierwsze dwie trzecie tablicy
7  STOOGE-SORT(A, i + k, j)   // ostatnie dwie trzecie tablicy
8  STOOGE-SORT(A, i, j − k)   // znowu pierwsze dwie trzecie
```

✎ Jaki jest czas działania tego algorytmu dla tablicy długości n: STOOGE-SORT$(A, 1, n)$?

Zadanie 22 Sortowanie przez scalanie

Scalanie (ang. merge) dwóch części tablicy ($A[p..q]$ i $A[q+1\,..\,r]$) — z których każda jest posortowana — polega na przepisaniu ich do pomocniczej tablicy w odpowiedniej kolejności, a następnie z powrotem do właściwej tablicy, gdzie będą już posortowane. Na przykład tablica $A = [10, 11, 13, 16, 9, 12, 14, 15]$ składa się z dwóch posortowanych części: $A[1..4]$ i $A[5..8]$. Ich scalenie odbywa się następująco (B jest pomocniczą tablicą): porównaj $A[1]$ z $A[5]$ i wpisz mniejszą wartość, czyli 9, do $B[1]$; następnie porównaj $A[1]$ z $A[6]$ i wpisz mniejszą wartość, czyli 10, do $B[2]$; następnie porównaj $A[2]$ z $A[6]$ i wpisz mniejszą wartość, czyli 11, do $B[3]$ itd., aż wszystkie elementy od 1 do 8 zostaną wpisane do tablicy B; na końcu przepisz elementy z tablicy $B = [9, 10, 11, 12, 13, 14, 15, 16]$ z powrotem do A.
Poniżej przedstawiono algorytm sortujący tablicę A od elementu p do elementu r z wykorzystaniem scalania.

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

✎ Jaki jest czas działania tego algorytmu dla tablicy długości n: MERGE-SORT$(A, 1, n)$?