Imię i nazwisko: ..... **WdI, Kolokwium nr 2**
Indeks: .....

1. Rozważmy podaną poniżej funkcję wylicz

```
def wylicz(n, m):
    if m==0: return n
    if n==0: return m
    if (n%2==0): k=n/2
    else: k=(n+1)/2
    return wylicz(k,m-1)+wylicz(n-1,1)+1
```

```
int  wylicz(int n, int m){
  int k;
  if (m==0) return n;
  if (n==0) return m;
    if (n%2==0) k=n/2;
    else k=(n+1)/2;
    return wylicz(k,m-1)+wylicz(n-1,1)+1;
}
```

a) **[3]** Narysuj drzewo wywołań dla `wylicz(3,1)`.

b) **[3]** Uzupełnij wartości funkcji wylicz w poniższej tabeli.

n	**m**	**wylicz(n, m)**
0	5	5
3	0	
1	1	
3	2	
2	3	

c) **[3]** Wywołaniem elementarnym nazwiemy każde wywołanie wylicz(n,0) lub wylicz(0,m) dla dowolnych argumentów n, m. Na przykład liczba wywołań elementarnych dla wylicz(1,1) wynosi 2, a dla wylicz(2,2) wynosi 4. Uzupełnij poniższą tabelę podając liczby wywołań elementarnych będących wynikiem wywołania wynik(n,m) dla podanych wartości n i m.

n	**m**	**liczba wywołań elementarnych dla wylicz(n, m)**
0	5	1
3	0	
1	1	
3	2	
2	3	

d) **[3]** Przez T(n,m) oznaczmy liczbę wywołań elementarnych będących wynikiem wywołania wynik(n,m). Uzupełnij poniższą zależność rekurencyjną:

T(n, 0) = 1 dla $n \geq 0$,

T(0, m) =..... dla $m \geq 0$,

T(n, m) = ... + ... dla $n > 0$ i $m > 0$

Imię i nazwisko: ..... **WdI, Kolokwium nr 2**
Indeks: .....

Imię i nazwisko: ..... **WdI, Kolokwium nr 2**
Indeks: .....

2. Dana jest następująca deklaracja:

```
struct elem { int val; struct elem *next; };
```

```
class ListItem:
 def __init__(self,value): self.val = value; self.next = None
```

Wersja 1 [15]: Napisz funkcję `usunOstPar`, która z listy podanej jako argument usunie ostatni element o parzystej wartości `val` (o ile parzyste elementy występują w liście) i zwróci wynikową listę jako wartość. Np. dla listy składającej się z elementów `1, 6, 2, 4, 3` wywołanie `usunOstPar(lista)` powinno dać w wyniku listę o wartościach kolejnych elementów `1, 6, 2, 3`. Natomiast dla listy z elementami 1, 3, 5, 7 w wyniku dostaniemy tę samą listę.
Nagłówek funkcji w języku C powinien mieć postać

```
struct elem *usunOstPar(struct elem *lista)
```

a w Pythonie `def usunOstPar(lista).`

Wersja 2 [8]: Napisz funkcję `parzyste`, która dla listy podanej jako argument zwraca sumę wartości elementów parzystych. Np. dla listy z elementami o wartościach 1, 6, 2, 4, 3 wynik powinien wynieść 12, a dla listy 1, 3, 5, 7 wynik to 0.
Nagłówek funkcji w języku C powinien mieć postać

```
int parzyste(struct elem *lista)
```

a w Pythonie `def parzyste(lista).`

Imię i nazwisko: ..... **WdI, Kolokwium nr 2**
Indeks: .....

Imię i nazwisko: ..... **WdI, Kolokwium nr 2**
Indeks: .....

3. Dana jest następująca funkcja:

```
def zagadka(a, n):
    k=1
    while k<n:
        i=0
        while i<n:
            scalaj(a,i,k)
            i=i+2*k
        k=2*k
```

```
void zagadka(int a[], int n){
    int k=1;
    while (k<n){
        i=0;
        while (i<n){
            scalaj(a,i,k);
            i=i+2*k;}
        k=2*k
}
```

Funkcja `scalaj` łączy posortowane ciągi a[i], a[i+1], …, a[i+k – 1] oraz a[i+k], a[i+k+1],…a[i+2k – 1] w posortowany ciąg a[i], a[i+1], …, a[i+2k – 1]. Dokładniej, funkcja scalaj realizuje następującą specyfikację:

Wejście: i, k – liczby naturalne; a – tablica liczb całkowitych takie, że ciągi a[i], a[i+1], …, a[i+k – 1] oraz a[i+k], a[i+k+1], …a[i+2k – 1] są uporządkowane niemalejąco

Wynik działania: ciąg a[i], a[i+1], …, a[i+2k – 1] jest posortowany niemalejąco, składa się z tych samych elementów, które zawierał przed wykonaniem funkcji.

a) **[3]** Podaj zawartość a[0..15] po wykonaniu zagadka(a,8) dla poniższych wartości początkowych:

a[0..15] **przed** wykonaniem zagadka(a, 8)	a[0..15] **po** wykonaniu zagadka(a, 8)
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	
15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0	
2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1	

b) **[3]** Uzupełnij poniższą specyfikację funkcji `zagadka`. Odpowiedź uzasadnij.

Specyfikacja
Wejście: a – tablica liczb naturalnych; n – liczba naturalna postaci $n=2^j$ dla naturalnego *j* (czyli n jest naturalną potęgą dwójki)

Wynik działania funkcji:

c) **[3]** Dla n=512 podaj liczbę wywołań (w trakcie działania `zagadka(a,n)`) funkcji `scalaj` z różnymi wartościami ostatniego parametru, uzupełniając poniższą tabelę:

j	Liczba wywołań `scalaj(a,i,k)` dla `k=j`
2	256
4	
8	
128	
512	

d) **[3]** Przyjmijmy, że czas działania `scalaj(a, i, k)` jest O(k). Uzupełnij poniższe zdania, przyjmując, że n jest naturalną potęgą liczby 2.

Liczba obrotów głównej pętli „`while k<n`” przy uruchomieniu `zagadka(a, n)` wynosi:

.....

Czas działania jednego obrotu głównej pętli „`while k<n`” przy uruchomienieu `zagadka(a, n)` wynosi (w notacji dużego O):

.....

Czas działania całej funkcji zagadka(a, n) wynosi (w notacji dużego O):

.....

Imię i nazwisko: ..... **WdI, Kolokwium nr 2**
Indeks: .....

4. Chcemy znaleźć ścieżkę prowadzącą przez kwadratową planszę rozmiaru n×n, opisaną przez dwuwymiarową tablicę a. Ścieżka powinna spełniać następujące warunki:
   (i) Ścieżkę zaczynamy w lewym górnym polu planszy, czyli a[0][0].
   (ii) Z pola a[i][j] możemy przejść do pól a[i+1][j – 1], a[i+1][j], a[i+1][j+1], z zastrzeżeniem, że nie można wyjść poza planszę. Oznacza to, że obie współrzędne pola, do którego przechodzimy należą do zakresu [0, n – 1].
   (iii) Wartość zapisana w każdym polu ścieżki jest **dodatnia**.

Poniżej prezentujemy funkcję służącą realizacji powyższego zadania:

```
int szukaj(int n, int kol, int wi, int a[n][n])
{ int i;
  if (kol==n) return 1;
  if (wi<0 || wi>=n) return 0;
  if (a[kol][wi]<=0) return 0;
  for(i=-1; i<=1; i++){
    if (szukaj(n,kol+1, wi+i, a)) return 1;
  }
  return 0;
}
```

```
def szukaj(n, kol, wi, a):
    if kol==n: return 1
    if wi<0 or wi>=n: return 0
    if a[kol][wi]<=0: return 0
    c[kol]=wi
    for i in range(-1,2):
        if szukaj(n,kol+1, wi+i, a): return 1
    return 0
```

a) **[2]** Podaj z jakimi parametrami należy wywołać funkcję `szukaj` tak, aby efektem była realizacja zadania o poniższej specyfikacji:

Wejście: n – liczba naturalna, a – tablica dwuwymiarowa

Wyjście: 1 – gdy istnieje ścieżka spełniająca warunki (i) – (iii) na planszy rozmiaru n×n opisanej tablicą a

0 – w przeciwnym przypadku.

Wywołanie:

```
szukaj( ...., ...., ...., .... )
```

b) **[3]** Uzasadnij, że uruchomienie funkcji szukaj z parametrami, które podałaś/podałeś w punkcie a) wymagać będzie sprawdzenia co najwyżej 3^{n-1} różnych ścieżek składających się z n pól.

c) **[3]** Uzasadnij, że uruchomienie funkcji szukaj z parametrami, które podałaś/podałeś w punkcie a) może wymagać w najgorszym przypadku sprawdzenia co najmniej 2^{n-1} różnych ścieżek składających się z n pól. Podaj przykład zawartości planszy, który będzie tego wymagał.

d) **[9]** Napisz funkcję realizującą poniższą specyfikację. Twoje rozwiązanie może (nie musi!) polegać na odpowiedniej modyfikacji funkcji `szukaj`.

Uwaga. Za poprawne rozwiązanie o złożoności czasowej $O(n^2)$ można uzyskać dodatkowy bonus 5 punktów

Wejście: n – liczba naturalna, a – tablica dwuwymiarowa

Wyjście: liczba ścieżek spełniających warunki (i) – (iii) na planszy rozmiaru n×n opisanej tablicą a.