1. ПРАКТИЧНІ ЗАНЯТТЯ

1.1. Загальні рекомендації щодо проведення практичних занять

План проведення практичних занять подано в табл. 1.1.

Таблиця 1.1 – План проведення практичних занять

№	Теми практичних занять	Кількість годин
1	Математичні основи теорії алгоритмів	2
2	Алгоритми сортування	4
3	Алгоритмічні стратегії	2
4	Машина Тюрінга	2
5	Генератори псевдовипадкових чисел	2
6	Фундаментальні алгоритми на графах і деревах	2
7	Геометричні алгоритми	2

На основі матеріалів, розглянутих на лекції, студентам пропонується виконати ряд завдань, направлених на закріплення теоретичного матеріалу.

На початку кожного практичного заняття студентам пропонуються питання для самоперевірки. Дані питання дозволяють не лише перевірити рівень засвоєння теоретичного матеріалу, але й поєднати вивчений матеріал з практикою.

Для перевірки отриманих знань, пов’язаних з основними поняттями і визначеннями, класифікацією структур даних та алгоритмів, пропонується тест (див. підрозділ 1.9).

На початку навчального семестру кожен студент отримує комплексне індивідуальне домашнє розрахункове завдання, метою якого є закріплення навичок розв’язання практичних задач. Варіанти даних завдань, а також вимоги до їх виконання та оформлення наводяться в підрозділі 1.9.

1.2. Математичні основи теорії алгоритмів

1.2.1. Короткі теоретичні відомості

Для оцінки складності алгоритмів використовують часову та просторову складності.

Часова складність алгоритму (в гіршому випадку) – це функція

від розміру вхідних даних, яка дорівнює максимальній кількості елементарних операцій, що здійснює алгоритм для розв'язання екземпляра задачі вказаного розміру.

Для оцінки просторової складності алгоритмів розглядають обсяг пам'яті, що використовується.

Аналізуючи алгоритм, можна спробувати знайти точну кількість виконуваних ним дій, але в більшості випадків достатньо оцінити асимптоту зростання часу роботи алгоритму, коли розмір входу прямує до нескінченості.

Для запису асимптотичної складності алгоритмів використовують такі асимптотичні позначення: $O(n)$, $\Theta(n)$, $\Omega(n)$, $o(n)$ і $w(n)$.

Запис $f(n) = \Theta(g(n))$ означає, що знайдуться такі константи $c_1, c_2 > 0$ і таке n_0, що $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ для всіх $n \ge n_0$.

Запис $f(n) = \mathrm{O}(g(n))$ означає, що знайдеться така константа $c > 0$ і таке n_0, що $0 \le f(n) \le cg(n)$ для всіх $n \ge n_0$.

Запис $T(n) = \Omega(n)$ означає, що знайдеться така константа $c > 0$ і таке n_0, що $0 \le cg(n) \le f(n)$ для всіх $n \ge n_0$.

Говорять, що $f(n) = o(g(n))$, якщо для будь-якого додатного ε знайдеться таке n_0, що $0 \le f(n) \le \varepsilon g(n)$ при всіх $n \ge n_0$.

Говорять, що $f(n) = w(g(n))$, якщо для будь-якого додатного ε знайдеться таке n_0, що $0 \le \varepsilon g(n) \le f(n)$ при всіх $n \ge n_0$.

Найважливіші функції, які використовуються для оцінки складності алгоритмів:

1) логарифм: $f(n) = \log_a n$ і полілогарифм: $f(n) = \log_a^b n$;

2) багаточлен: $p(n) = \sum_{i=0}^{d} a_i n_i$;

3) експонента: $f(n) = a^n$;

4) факторіал: $f(n) = n!$.

1.2.2. Приклади розв'язання задач

Задача 1

Визначити складність алгоритму, записаного у вигляді псевдокоду:

```
for k=1 to m
  for i=1 to m
    for j=1 to m
      if a[i,k]-a[k,j]≤a[i,j]
        a[i,j]=a [i,k]-a[k,j].
```

Розв'язання

Псевдокод – мова описання алгоритмів, яка використовує ключові слова мов програмування, але не враховує подробиці та специфічний синтаксис. Детальніше про псевдокод можна прочитати в авторів книги [1].

Як було зазначено, часом роботи алгоритму називається число елементарних кроків, які він виконує. Вважаємо, що один рядок псевдокоду потребує не більше фіксованого числа операцій.

Позначимо біля кожного рядка алгоритму його вартість (число операцій) і кількість разів, яку даний рядок буде виконуватись (табл. 1.2).

Таблиця 1.2 – Вартість операцій і кількість разів виконання рядків алгоритму

№ рядка	Алгоритм	Кількість операцій (вартість)	Скільки разів буде виконаний рядок
1	`for k=1 to m`	c_1	$m+1$
2	`  for i=1 to m`	c_2	$m(m+1)$
3	`    for j=1 to m`	c_3	$m^2(m+1)$
4	`      if a[i,k]-a[k,j]≤a[i,j]`	c_4	m^3
5	`        a[i,j] =a [i,k]-a[k,j]`	c_5	$\leq m^3$

Перший рядок виконуватиметься $(m+1)$ разів. Для кожного k від 1 до m підрахуємо, скільки разів буде виконаний другий рядок псевдокоду, для кожного i від 1 до m підрахуємо, скільки разів буде виконаний третій рядок, для кожного j від 1 до m підрахуємо, скільки разів будуть виконуватись 4-й та 5-й рядки.

Поєднавши внески всіх рядків, отримаємо:

$$T(m) = c_1(m+1) + c_2 m(m+1) + c_3 m^2(m+1) + c_4 m^3 + c_5 m^3 =$$
$$= (c_3 + c_4 + c_5)m^3 + (c_2 + c_3)m^2 + (c_1 + c_2)m + c_1.$$

Перетворимо даний вираз, відкинувши члени менших порядків і коефіцієнт при m. Тоді отримаємо $T(m) = O(m^3)$.

Задача 2

Порівняти швидкості зростання заданих функцій, визначивши, чи можна дану пару визначити, як $f(n) = O(g(n))$:

$$f(n) = \sin^k(n),\ g(n) = \log^k(n),\ n,k > 1.$$

Розв'язання

Якщо k – непарне число, то функція $f(n) = \sin^k(n)$ може бути від'ємною, а за визначенням $O(n)$ функції мають бути невід'ємні, а значить, функції $f(n)$ і $g(n)$ не можна порівнювати при непарному значенні k.

Якщо k – парне число, то $0 \le f(n) \le 1$, а значить, $\sin^k(n) = O(\log^k(n))$.

Задача 3

Користуючись O-позначенням, довести:

$$n^2 + 5n = O(n^2).$$

Розв'язання

Позначимо ліву функцію у виразі, який необхідно довести, як $f(n) = n^2 + 5n$. Праву функцію позначимо $g(n) = n^2$.

Тоді у відповідності до визначення O-позначення необхідно вказати такі константи $c > 0$ і n_0, що $\forall n \ge n_0$ буде правильним вираз $0 \le n^2 + 5n \le cn^2$.

Розділимо ліву і праву частини даного виразу на n. Отримаємо

$$0 \le 1 + \frac{5}{n} \le c.$$

Тепер легко вибрати константи, при яких даний вираз залишається правильним. Наприклад, виберемо значення $c = 2$ і $n_0 = 5$. Для $n \ge 5$ (тобто $\forall n \ge n_0$) вираз $(1 + \frac{5}{n})$ зменшуватиметься, а значить, твердження $n^2 + 5n = O(n^2)$ є правильним.

Запитання для самоконтролю

1. Що таке псевдокод?

2. Що називається часом роботи алгоритму?

3. Як визначається складність алгоритму?

4. Перелічіть найважливіші функції, які використовуються для визначення складності алгоритмів. Як за швидкістю зростання пов'язані дані функції?

Задачі для аудиторних занять

1. Визначити складність алгоритмів, записаних у вигляді псевдокоду.

1)
```
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            a[i][j] = min(a[i][j], a[i][k] + a[k][j])
```

2)
```
is_prime = true
for k = 2 to n – 1
  if n % k == 0
    is_prime = false
  break
```

3)
```
for j = 1 to n
  for i = 1 to j – 1
    if a[i] > a[i + 1]
      //Swap a[i] and a[i + 1]
      a[i], a[i + 1] = a[i + 1], a[i]
```

4)
```
for i = 1 to n - 1
  min_index = i
  for j = i + 1 to n
    if a[min_index] > a[j]
      min_index = j
  if min_index ≠ i
        //Swap a[min_index] and a[i]
        a[min_index], a[i] = a[i], a[min_index]
```

5)
```
merged = []
while !empty(a) && !empty(b)
  if a[1] < b[1]
    merged.append(a[1])
    a.pop_front()
  else:
    merged.append(b[1])
```

```
    b.pop_front()
while !empty(a)
    merged.append(a[1])
    a.pop_front()
while !empty(b)
    merged.append(b[1])
    b.pop_front()
```

6)
```
for i = 1 to m / 2
  //Swap a[i] and a[m − i + 1]
  a[i], a[m - i + 1] = a[m - i + 1], a[i]
```

7)
```
for i = 1 to m
  if a[i] % 3 = 1
    for j = 1 to m
      b[i][j] = a[i]
```

8)
```
result = 1
for i = 1 to k
  result *= k
```

9)
```
a_in_power = a
result = 1
while k > 0
  if k % 2 == 1
    result *= a_in_power
  a_in_power *= a_in_power
  k = k / 2
```

10)
```
i_lower = 1
i_upper = n
while i_lower < i_upper
  i_middle = (i_lower + i_upper) / 2
  if a[i_middle] > goal
    i_upper = i_middle - 1
  elif a[i_med] < goal
    i_lower = i_middle + 1
  else
    break
```

2. Порівняти попарно швидкості зростання заданих функцій, визначивши, чи можна кожну пару записати як $f(n) = O(g(n))$, при цьому $n, k > 1$:

$$f_1(n) = \log_k n;$$
$$f_2(n) = \log_3(n^{3n});$$
$$f_3(n) = n^k;$$

$$f_4(n) = k^n;$$
$$f_5(n) = \sqrt{n};$$
$$f_6(n) = \sin n;$$
$$f_7(n) = \sin^2 n;$$
$$f_8(n) = \sin^2 n + \cos^2 n;$$
$$f_9(n) = \text{n!};$$
$$f_{10}(n) = e^n.$$

3. Довести дані твердження, користуючись визначенням O-позначення:

$$7n^3 + n^2 = O(n^3);$$
$$n^{1024} = O(n!);$$
$$n^{14} = O\left(2^n\right);$$
$$n^3 + n^2 + n + 1 = O(n^4);$$
$$n + 0{,}75^n = O(n);$$
$$e^n = O(\pi^n);$$
$$e^n + n^{512} = O(e^n);$$
$$2\log n = O(n);$$
$$n\log n = O(n^2);$$
$$n^2 + n\log^4 n = O(n^2).$$

1.3. Алгоритми сортування

1.3.1. Сортування вставками. Короткі теоретичні відомості та приклади розв'язання задач

Сортування вставками (англ. *Insertion Sort*) зручне для коротких послідовностей, складність алгоритму $O(n^2)$, де n – кількість елементів для сортування. Даний алгоритм не потребує додаткового обсягу пам'яті, окрім однієї змінної.

Суть алгоритму: послідовність даних поділяється на відсортовану та невідсортовану частини (спочатку як відсортована частина приймається перший елемент); елементи з невідсортованої частини по черзі вибираються та вставляються у відсортовану частину таким чином, щоб не порушувати в ній впорядкованість елементів.

Задача 1

Вихідна послідовність подана такими елементами: $6, 3, 8, 1, 5, 7$. Необ-