**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**Практична робота № 3**

**Тема:** Поняття алгоритму. Властивості алгоритмів. Способи подання алгоритмів. Складність алгоритмів.

Виконав студент 1 курсу

спеціальності “Інженерія

програмного забезпечення”

Боднарчук Роман Романович

**Ужгород-2025**

**Мета:** набути практичних навичок подання алгоритмів різними способами та визначення їх складності.

**Завдання до роботи:**

1. Написати код для виконання алгоритму відповідно до варіанту завдання. Аргументуйте використання обраного алгоритму. Визначте складність алгоритму.  
   Примітка. Не використовувати готові методи та функції Javascript.
2. Оформити звіт, де подати алгоритм чотирма способами: словесним, псевдокодом, графічно (використовуючи доступні програмні продукти) та безпосередньо кодом програми.
3. Завантажити звіт в системі електронного навчання ДВНЗ «УжНУ» в установлений термін.
4. Підготувати відповіді на контрольні питання.

# Хід роботи

**Варіант 2**

1. Напишіть алгоритм, який знаходить найбільше число в масиві

const list = []

for (let i = 0; i < 10; i++)

    list.push(Math.round(Math.random() \* 20));

const findMax = (list) => {

    let max = list[0];

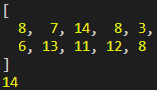
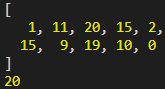
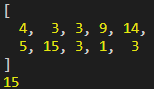
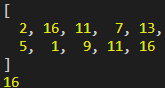
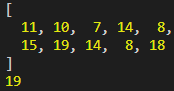
    list.forEach(el => el > max ? max = el : false);

    return max

}

console.log(list)

console.log(findMax(list))



**Словесний опис алгоритма:**

Цей код створює масив list з 10 випадкових цілих чисел від 0 до 20 (включно), а потім шукає максимальне число у цьому масиві.

* 1. Створення масиву: спочатку створюється порожній масив list.
  2. Заповнення масиву: в циклі for змінна i збільшується від 0 до 9. На кожній ітерації до масиву list додається випадкове ціле число від 0 до 20, округлене за допомогою Math.round() та Math.random().
  3. Пошук максимуму: функція findMax отримує масив list як аргумент і шукає найбільший елемент у цьому масиві. Спочатку змінна max встановлюється рівною першому елементу масиву. Потім використовується метод forEach для ітерації через кожен елемент масиву. Якщо поточний елемент більший за поточне максимальне значення max, то max оновлюється.
  4. Вивід результату: виводиться масив list та результат виклику функції findMax(list), яка повертає максимальне число у масиві.

Код створює масив із 10 випадкових чисел та знаходить найбільше число серед них.

**Псевдокод для опису алгоритма:**

Створити пустий список list

//Для кожного i від 0 до 9

    Додати до списку list ціле випадкове число в діапазоні від 0 до 20

ЗнайтиМаксимум(list)

Функція ЗнайтиМаксимум(list):

    ПершийЕлемент = list[0]

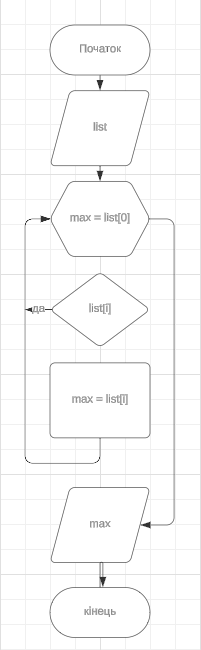
    Для кожного елементу el в списку list

        Якщо el > ПершийЕлемент

            ПершийЕлемент = el

    Вернути ПершийЕлемент

**Графічний опис алгоритму:**



**Питання для самоконтролю**

1. Дайте визначення алгоритму. Наведіть приклади алгоритмів. Чи є алгоритмом програма, подана певною мовою програмування або машинними кодами?

**Відповідь:** Алгоритм - це послідовність конкретних інструкцій, які виконуються для вирішення конкретної задачі. Приклади алгоритмів: сортування бульбашкою, пошук максимального елемента у масиві, обхід дерева у глибину тощо. Програма або машинні коди можуть реалізовувати алгоритм, але самі по собі не є алгоритмами.

1. Вкажіть і схарактеризуйте основні властивості алгоритмів.

**Відповідь:** Основні властивості алгоритмів:

- **Коректність:** алгоритм повинен вирішувати поставлену задачу правильно.

- **Визначеність:** кожна крок алгоритму має бути однозначно визначений і зрозумілий.

- **Ефективність:** алгоритм повинен вирішувати задачу за обмежений час, використовуючи обмежені ресурси.

- **Вхід та вихід:** алгоритм повинен мати чітко визначені вхідні та вихідні дані.

- **Дискретність:** алгоритм повинен складатися з дискретних кроків, які можна виконати по окремому завершенню попереднього кроку.

- **Універсальність:** алгоритм повинен бути загальноприйнятим для вирішення певного класу задач.

1. У чому полягає аналіз ефективності алгоритму? Що таке часова скла- дність алгоритму? Що таке ємнісна складність алгоритму?

**Відповідь:** Аналіз ефективності алгоритму полягає в оцінці його продуктивності та визначенні, наскільки швидко або ефективно алгоритм працює при обробці великої кількості даних:

- **Часова складність** алгоритму визначає, як швидко зростає час виконання алгоритму при збільшенні розміру вхідних даних. Вимірюється в часових одиницях, таких як кількість операцій або кількість кроків, потрібних для завершення алгоритму.

- **Ємнісна складність** алгоритму (іноді називається просторовою складністю) визначає, як зростає кількість пам'яті, потрібна для виконання алгоритму, при збільшенні розміру вхідних даних. Вимірюється у величині пам'яті, необхідної для виконання алгоритму.

1. Нехай необхідно побудувати такі алгоритми:
   * Обчислення суми перших n елементів геометричної прогресії.
   * Сортування масиву вставками.
   * Пошук максимального числа в тривимірному масиві.
   * Знаходження коренів нелінійного рівняння 5-го степеня.

Визначте, які з цих алгоритмів є чисельними, а які – логічними.

**Відповідь:**

- Обчислення суми перших n елементів геометричної прогресії - це чисельний алгоритм, оскільки вимагає математичних обчислень для знаходження суми.

- Сортування масиву вставками - це чисельний алгоритм, оскільки вимагає порівнянь та переміщень елементів у масиві для досягнення впорядкування.

- Пошук максимального числа в тривимірному масиві - це логічний алгоритм, оскільки вимагає перевірки умови (чи є поточний елемент більшим за попередній максимум) для знаходження результату.

- Знаходження коренів нелінійного рівняння 5-го степеня - це чисельний алгоритм, оскільки вимагає чисельного розв'язання рівняння для знаходження коренів.

1. Чи можна побудувати паралельний алгоритм додавання N чисел?

**Відповідь:** Так, можна побудувати паралельний алгоритм додавання N чисел. Для цього можна розділити множину чисел на декілька груп і обчислити суму кожної групи паралельно. Після цього можна обчислити суму цих проміжних сум також паралельно, що дозволить ефективно виконати додавання N чисел у паралельному режимі.

1. Схарактеризуйте константну, лінійну, квадратичну, логарифмічну, лінійно-логарифмічну і експоненційну складності алгоритму.

**Відповідь:**

- **Константна складність** означає, що час виконання алгоритму не залежить від обсягу вхідних даних. Приклад: доступ до елементу масиву за індексом.

- **Лінійна складність** означає, що час виконання алгоритму зростає пропорційно обсягу вхідних даних. Приклад: ітерація по елементам масиву.

- **Квадратична складність** означає, що час виконання алгоритму зростає квадратично відносно обсягу вхідних даних. Приклад: вкладений цикл для сортування методом бульбашки.

- **Логарифмічна складність** означає, що час виконання алгоритму зростає логарифмічно відносно обсягу вхідних даних. Приклад: бінарний пошук у відсортованому масиві.

- **Лінійно-логарифмічна складність** означає, що час виконання алгоритму зростає лінійно помножено на логарифм від обсягу вхідних даних. Приклад: швидке сортування (QuickSort).

- **Експоненційна складність** означає, що час виконання алгоритму зростає експоненціально від обсягу вхідних даних. Приклад: рекурсивний алгоритм обчислення чисел Фібоначчі без оптимізації.

1. Які задачі належать до класу задач поліноміальної складності? Чи завжди велика асимптотична складність свідчить про те, що алгоритм буде виконуватися довше для будь-яких обсягів даних?

**Відповідь:** Задачі поліноміальної складності - це задачі, для яких існують алгоритми з часовою складністю, яка зростає не більше, ніж поліноміально від розміру вхідних даних. Велика асимптотична складність не завжди свідчить про те, що алгоритм буде виконуватися довше для будь-яких обсягів даних. Наприклад, алгоритм з експоненційною складністю може бути швидшим за алгоритм з квадратичною складністю для невеликих обсягів даних, але стає неприйнятно повільним для великих обсягів даних через експоненційне зростання часу виконання.

1. Вкажіть способи подання алгоритмів. Схарактеризуйте словесне подання алгоритму. Які блоки використовують для графічного подання алгоритму? Які відмінності алгоритмічної мови від машинного коду?

**Відповідь:** Способи подання алгоритмів:

- **Словесне:** алгоритм може бути описаний у вигляді послідовності кроків за допомогою природної мови, наприклад, англійської.

- **Псевдокод:** це спосіб опису алгоритму, що нагадує синтаксис конкретної мови програмування, але є більш зрозумілим для людини.

- **Блок-схеми:** графічний спосіб подання алгоритму за допомогою блоків, які представляють операції, рішення та зв'язки між ними.

- **Мова програмування:** алгоритми можна також представляти у вигляді програм на конкретних мовах програмування, таких як JavaScript, C#, Python тощо.

Словесне подання алгоритму - це опис кроків, які потрібно виконати, використовуючи звичайну мову. Воно може бути зрозумілим для людини, але може бути менш точним та не таким структурованим, як псевдокод або код на мові програмування.

Для графічного подання алгоритмів використовують блоки, такі як прямокутники для операцій, ромби для рішень, паралелограми для введення/виведення даних та стрілки для зв'язків між блоками.

Алгоритмічна мова - це спеціальна мова програмування, яка використовується для написання алгоритмів та структурних діаграм. Вона відрізняється від машинного коду тим, що призначена для спрощеного та зрозумілого опису алгоритмів, тоді як машинний код призначений для безпосереднього виконання комп'ютером.

**Висновки:** У ході виконання Практичної роботи №3, я створив Алгоритм який знаходить найбільше число в масиві. Тим самим набув практичних навичок із виконанням алгоритмів та зробив висновок, що складність мого алгоритму має O(n), де n – це кількість елементів у списку. Оскільки він перебирає всі елементи у списку один раз, то його часова складність зростає лінійно з розміром списку.