**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**Лабораторна робота №3**

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

Виконав студент 1 курсу

спеціальності “Інженерія

програмного забезпечення”

Дєлов В’ячеслав Едуардович

**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.

**Завдання 1**

**Варіант 18**Напишіть алгоритм, який обчислює довжину найбільшої зростаючої підпослідовності у заданому масиві.

**Хід роботи:**Динамічне програмування (DP) – це спосіб вирішення завдань, у якому розбиваємо велике завдання на більш дрібні підзадачі, вирішуємо їх і використовуємо результати для побудови відповіді.

1. *Ідея алгоритму (словесний опис)*

**Ініціалізація:**

* Створюємо масив D[], де D[i] — довжина LIS, що закінчується на arr[i].
* Створюємо масив prev[], де prev[i] містить індекс попереднього елемента в LIS.
* Всі D[i] ініціалізуємо як 1 (кожен елемент сам по собі є LIS).

**Обчислення довжин LIS:**

* Для кожного arr[i] перевіряємо всі попередні arr[j] (j < i).
* Якщо arr[j] < arr[i] і D[j] + 1 > D[i], оновлюємо D[i] і prev[i].

**Знаходження індексу найбільшої LIS:**

* Шукаємо max(D), який дасть довжину LIS.
* Запам'ятовуємо відповідний maxIndex.

**Відновлення LIS:**

* Використовуємо prev[], щоб пройти у зворотному порядку і зібрати LIS.
* Реверсуємо отриманий список, щоб отримати LIS у правильному порядку.

1. *Псевдокод:*

ФУНКЦІЯ LIS(arr[], n)

СТВОРИТИ масиви D[n] і prev[n]

ІНІЦІАЛІЗУВАТИ D[] значенням 1, prev[] значенням -1

ДЛЯ i від 1 ДО n-1:

ДЛЯ j від 0 ДО i-1:

ЯКЩО arr[j] < arr[i] і D[j] + 1 > D[i]:

D[i] ← D[j] + 1

prev[i] ← j

maxLength ← 1

maxIndex ← 0

ДЛЯ i від 0 ДО n-1:

ЯКЩО D[i] > maxLength:

maxLength ← D[i]

maxIndex ← i

LIS ← []

ПОКИ maxIndex ≠ -1:

ДОДАТИ arr[maxIndex] у LIS

maxIndex ← prev[maxIndex]

ПОВЕРНУТИ перевернутий LIS

1. *Код програми:*

function longestIncreasingSubsequenceDPLength(arr) {

    let n = arr.length;

    if (n === 0) return 0;

    let D = new Array(n).fill(1);

    for (let i = 1; i < n; i++) {

        for (let j = 0; j < i; j++) {

            if (arr[j] < arr[i]) {

                D[i] = Math.max(D[i], D[j] + 1);

            }

        }

    }

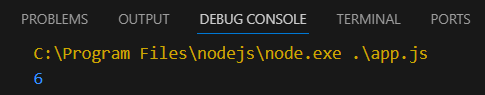
    return Math.max(...D);

}

Тест роботи:

let arr = [10, 22, 9, 43, 99, 33, 21, 50, 41, 60, 80];

console.log(longestIncreasingSubsequenceDPLength(arr))

Результат повертає довжину НЗП:  


// Очікуємий результат: [10, 22, 33, 50, 60, 80]

Зробимо можливість відновлення Найбільшої Зростаючої Підпослідовності (НЗП):

function longestIncreasingSubsequence(arr) {

    let n = arr.length;

    if (n === 0) return [];

    let D = new Array(n).fill(1);

    let prev = new Array(n).fill(-1);

    let maxLength = 1, maxIndex = 0;

    for (let i = 1; i < n; i++) {

        for (let j = 0; j < i; j++) {

            if (arr[j] < arr[i] && D[j] + 1 > D[i]) {

                D[i] = D[j] + 1;

                prev[i] = j;

            }

        }

        if (D[i] > maxLength) {

            maxLength = D[i];

            maxIndex = i;

        }

    }

    let lis = [];

    while (maxIndex !== -1) {

        lis.push(arr[maxIndex]);

        maxIndex = prev[maxIndex];

    }

    return lis.reverse();

}

Приклад ітерацій:

arr = [10, 22, 9, 43, 99, 33, 21, 50, 41, 60, 80]  
prev = [-1, -1,-1,-1,-1,-1,-1,-1,-1,-1,-1];

D = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

* Перша ітерація:

**i = 1 (arr[1] = 22):**

* j = 0: arr[0] = 10, 10 < 22
* D[1] = D[0] + 1 = 2
* prev[1] = 0

D = [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1];

prev = [-1, 0,-1,-1,-1,-1,-1,-1,-1,-1,-1];

* Остання ітерація:

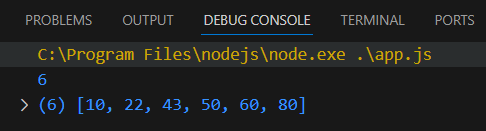
D = [1, 2, 1, 3, 4, 3, 2, 4, 3, 5, 6];

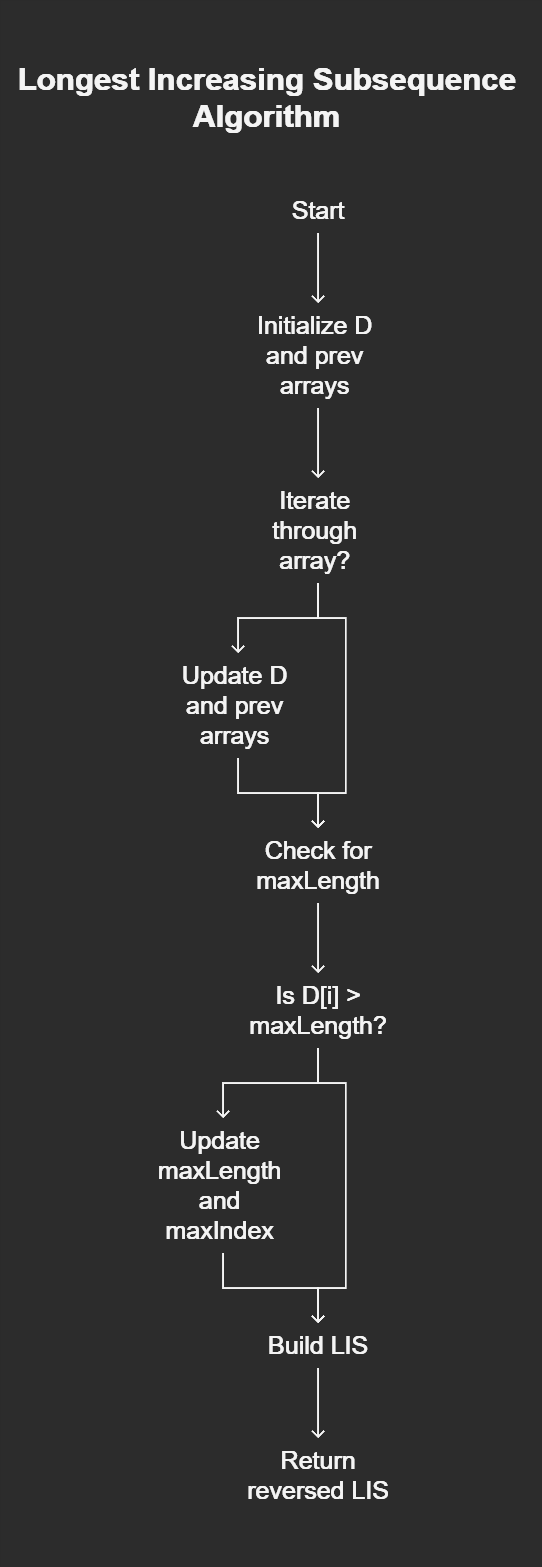
prev = [-1, 0,-1, 1, 3, 1, 2, 5, 5, 7, 9];

Результат:

console.log(longestIncreasingSubsequence(arr));

// Очікуємий результат: [10, 22, 33, 50, 60, 80]



1. Графічне зображення
2. 

Код програм запушено на гіт-хаб в репозиторій https://github.com/DelovSlava/Algoritmi\_-\_Strutures

* **Зовнішній цикл** виконується **n разів** → **O(n)**
* **Внутрішній цикл** в середньому проходить **i разів**, тобто від 1 до n
* Сума операцій у внутрішньому циклі: 1+2+3+...+(n−1)=n(n−1)2​= O(n^2)

🔹 **Підсумкова складність:** O(n^2)

**Висновок:**

У ході виконання лабораторної роботи було розглянуто основні поняття, властивості та способи подання алгоритмів. Було набуте практичне вміння представляти алгоритми у вигляді словесного опису, блок-схем та псевдокоду, що дозволило краще зрозуміти їхню структуру та логіку роботи.

Окрім цього, було досліджено питання складності алгоритмів.