**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**ЛАБОРАТОРНА РОБОТА № 8**

**Тема**: “Алгоритми у js.”

Виконав студент І курсу

спеціальності «Основи веб-розробки »

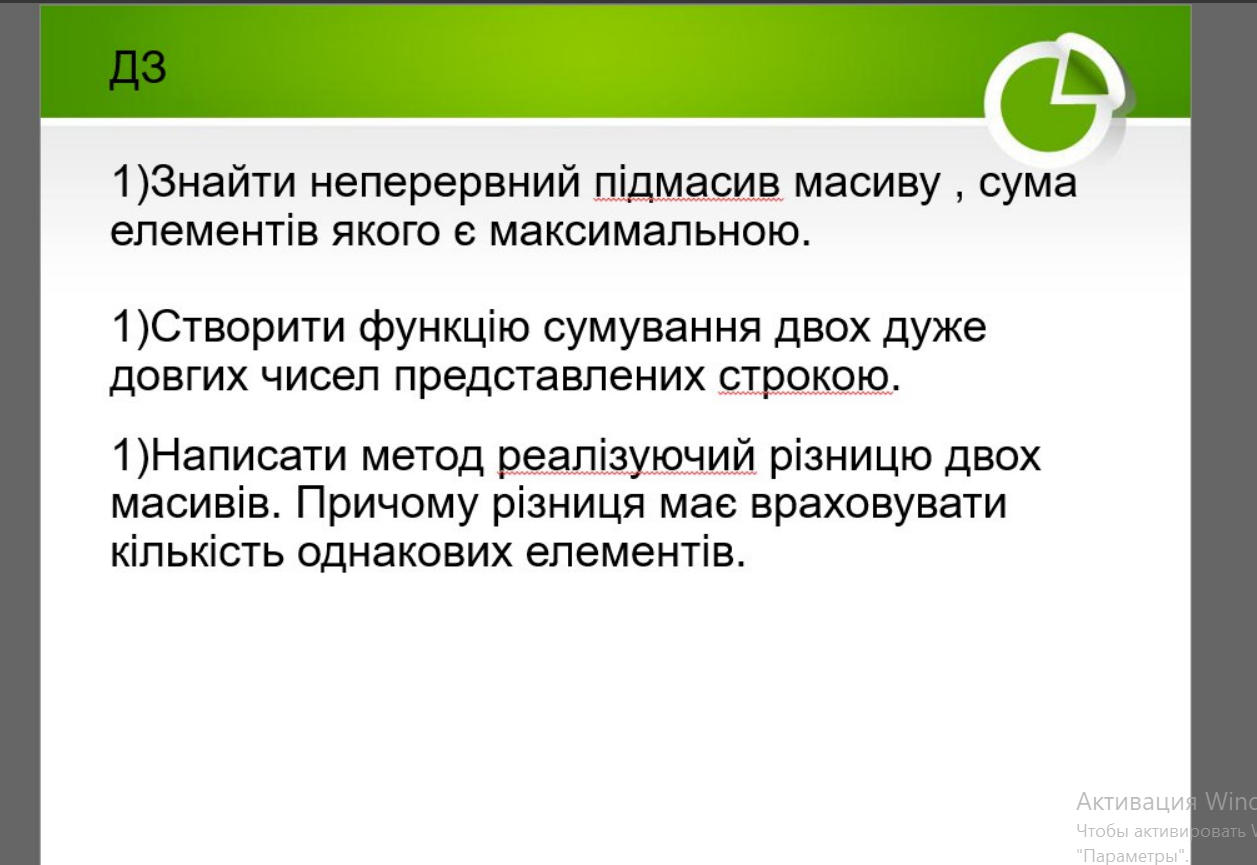
Вялков Владислав Костянтинович

**Ужгород-2024**

**Мета:**Метою цієї лабораторної роботи є дослідження та вивчення алгоритмів, що дозволяють визначити максимальну суму неперервного підмасиву в масиві чисел. Зокрема, лабораторна робота має на меті:

1. **Ознайомлення з алгоритмом Кадане**: Розглянути ефективний алгоритм для знаходження підмасиву з максимальною сумою та зрозуміти його принципи роботи.
2. **Розвиток навичок програмування**: Вивчити реалізацію алгоритму на мові JavaScript, що сприятиме підвищенню практичних навичок програмування та роботи з масивами.
3. **Аналіз продуктивності**: Дослідити складність алгоритму та його ефективність у порівнянні з іншими методами вирішення аналогічних задач.
4. **Практичне застосування**: Розглянути можливості застосування отриманих знань у реальних задачах, таких як фінансовий аналіз, обробка даних та оптимізація ресурсів.

Завдання до работи:



Хід роботи

Користувався компілятором

https://www.programiz.com/javascript/online-compiler/

1. Знайти неперервний підмасив масиву, сума елементів якого є максимальною

function maxSubArray(nums) {

let maxSum = nums[0];

let currentSum = nums[0];

for (let i = 1; i < nums.length; i++) {

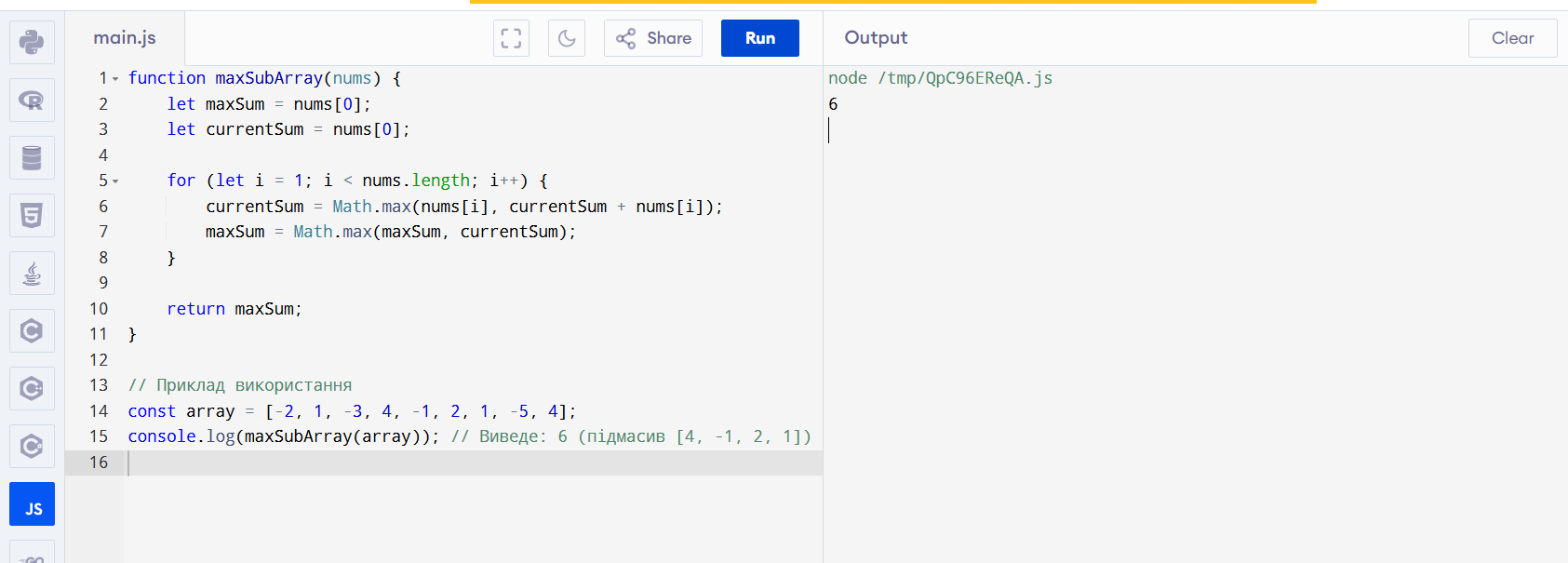
currentSum = Math.max(nums[i], currentSum + nums[i]);

maxSum = Math.max(maxSum, currentSum);

}

return maxSum;

}



Для розв'язання цієї задачі використовують **алгоритм Кадане**. Він працює за наступними кроками:

* Ініціалізуйте дві змінні: maxSum для збереження максимальної суми підмасиву і currentSum для обчислення суми поточного підмасиву.
* Пройдіться по всіх елементах масиву. На кожному кроці порівняйте поточний елемент з сумою currentSum, щоб визначити, чи вигідніше почати новий підмасив або продовжити поточний.
* Поновлюйте maxSum, якщо currentSum перевищує його.
* В кінці поверніть maxSum як результат.

Цей алгоритм має складність O(n), що робить його дуже ефективним для великих масивів.

1. Створити функцію сумування двох дуже довгих чисел, представлених строкою

function sumLargeNumbers(num1, num2) {

let result = '';

let carry = 0;

let maxLength = Math.max(num1.length, num2.length);

// Додайте нулі попереду, щоб обидва числа мали однакову довжину

num1 = num1.padStart(maxLength, '0');

num2 = num2.padStart(maxLength, '0');

for (let i = maxLength - 1; i >= 0; i--) {

const sum = parseInt(num1[i]) + parseInt(num2[i]) + carry;

result = (sum % 10) + result; // Додаємо останню цифру

carry = Math.floor(sum / 10); // Обчислюємо перенос

}

// Додаємо залишок, якщо він є

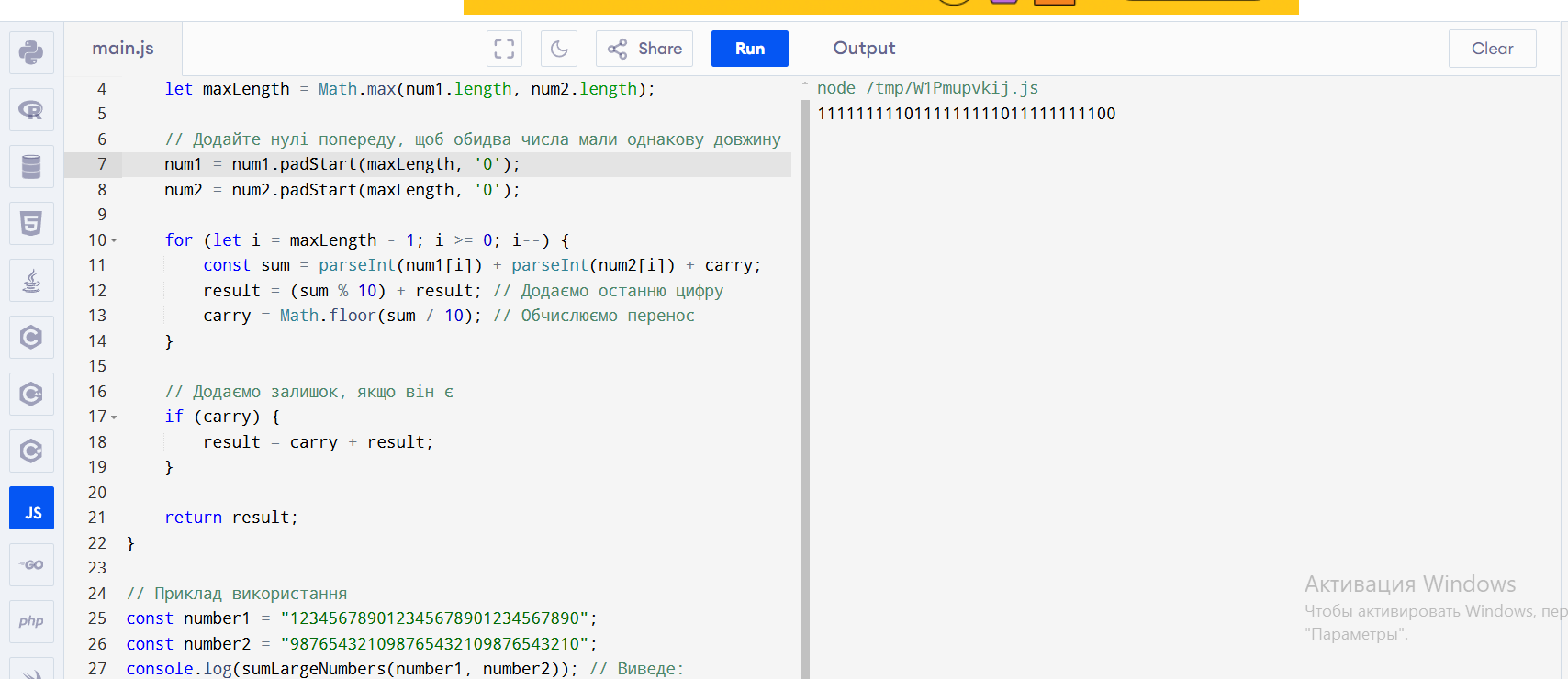
if (carry) {

result = carry + result;

}

return result;

}



Для цієї задачі потрібно врахувати, що числа можуть бути представлені у вигляді рядків, і стандартні арифметичні операції не можуть бути застосовані до таких великих чисел. Розв'язання може бути таким:

* Розверніть обидва рядки (числа) для спрощення сумування з одиниць.
* Використовуйте цикл для проходження по обох рядках, додаючи відповідні цифри, враховуючи перенесення (як у звичайному додаванні).
* Додайте результати в новий рядок, зберігаючи порядок.
* В кінці оберніть результат назад, щоб отримати його в правильному порядку.

Ця задача демонструє важливість маніпуляцій з рядками та основних арифметичних операцій.

3. Написати метод, що реалізує різницю двох масивів, враховуючи кількість однакових елементів

function arrayDifference(arr1, arr2) {

const count = {};

// Підрахунок елементів у другому масиві

arr2.forEach(num => {

count[num] = (count[num] || 0) + 1;

});

// Створюємо новий масив, зменшуючи кількість однакових елементів

const result = [];

arr1.forEach(num => {

if (count[num]) {

count[num]--;

} else {

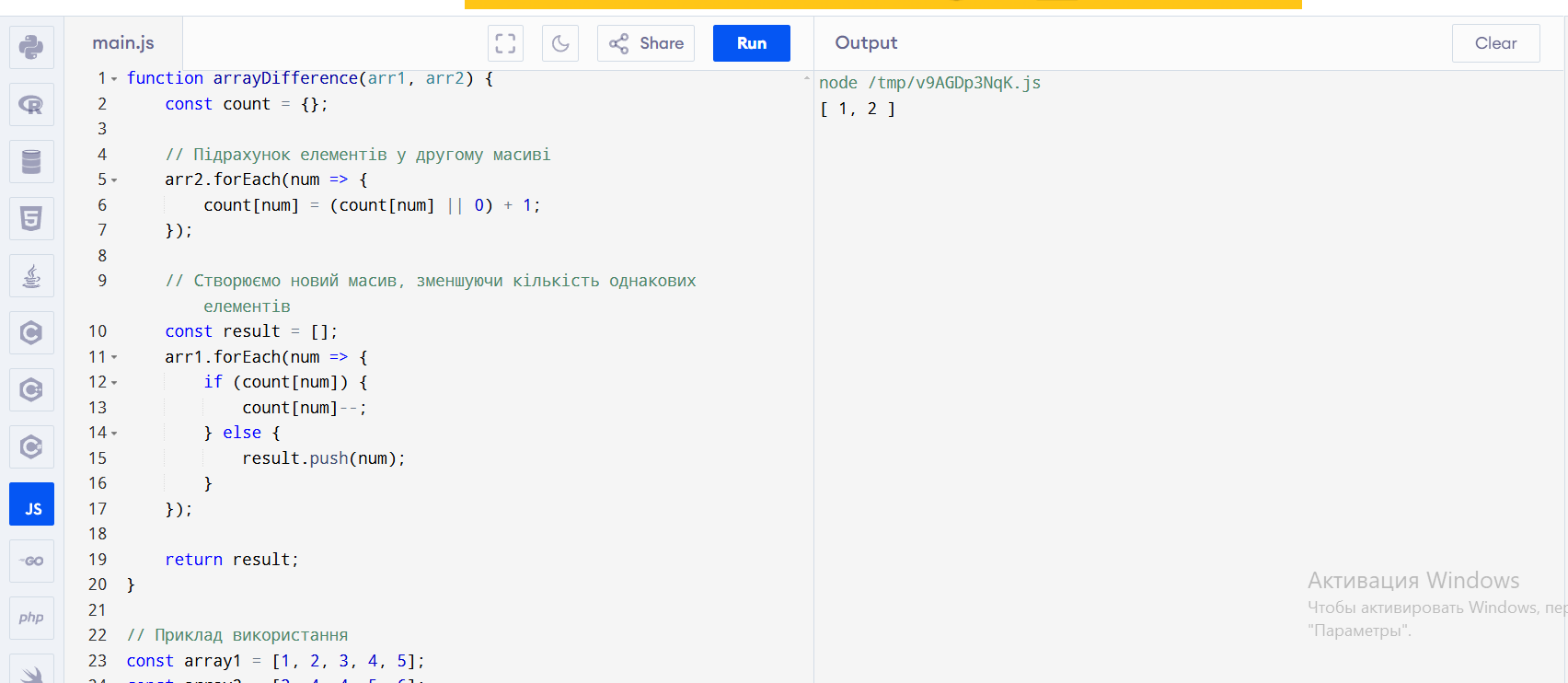
result.push(num);

}

});

return result;

}



Для цього вам потрібно створити функцію, яка:

* Приймає два масиви як аргументи.
* Створює об'єкт (або карту) для зберігання кількостей елементів обох масивів.
* Проходить через обидва масиви, зменшуючи кількість елементів з першого масиву на кількість, знайдену в другому.
* Повертає масив, що містить елементи, які залишилися після обчислення різниці.

Цей метод показує, як можна працювати з колекціями даних, а також демонструє важливість роботи з кількістю елементів у масивах.

**Висновок:**

У процесі виконання лабораторної роботи була досліджена задача знаходження максимальної суми неперервного підмасиву з використанням алгоритму Кадане. Було встановлено, що цей алгоритм, завдяки своїй лінійній складності O(n), є надзвичайно ефективним для вирішення даної проблеми, оскільки він дозволяє швидко обробляти великі масиви даних.

Реалізація алгоритму на мові JavaScript дозволила не лише закріпити теоретичні знання, а й отримати практичний досвід у програмуванні, що є важливим для подальшого розвитку в галузі комп'ютерних наук. Дослідження показало, що розуміння принципів роботи алгоритмів може суттєво вплинути на здатність ефективно вирішувати складні задачі, що виникають у реальному житті.

Отримані результати підкреслюють значення алгоритмічного мислення та оптимізації рішень у програмуванні, що відкриває нові можливості для подальшого навчання та професійної діяльності. В майбутньому, знання про алгоритм Кадане може бути корисним не лише в контексті аналізу даних, а й у розробці інших програмних рішень.