Maximum Subarray Sum

لينك ال جيت هب فيه كل الملفات اللى تخص المشروع بطريقه منظمه جدا
https://github.com/SamahCodes/Algorithm

Team Information:-

Team Members:

Samah Mohamed Salah — ID: 1000287603

Sahar Osama El-sieed — ID: 1000287997

Sahar Reda Helmy — ID: 1000288192

Abdelrahman Mohamed — ID: 1000287467

Project Overview:-

لينك ال Readme file على git hub

This project presents a complete study of the Maximum Subarray Sum Problem, focusing on comparing a Naive (Brute Force) solution with an Optimized solution using Kadane's Algorithm.

The project includes:

Problem identification

Algorithm implementations in JavaScript

Theoretical analysis

Empirical analysis

Performance comparison

Documentation and presentation materials

Problem Identification:-

The Maximum Subarray Sum problem aims to find the maximum possible sum of a contiguous subarray within a given array of integers (which may contain both positive and negative values).

يعنى array فيها ارقام موجبه وسالبه بنطلع ال array اللى مجموعها اكبر مجموع شرط ان يكون عناصرها متصليين

Input:

An array of integers (positive and/or negative)

Output:

The maximum sum of any contiguous subarray

Algorithms Implemented:-

1-Naive Solution (Brute Force): لينك الفايل اللى بيشرح كل حاجه تخص النايف

Checks all possible contiguous subarrays

Computes their sums

Selects the maximum sum

```
function maxSubarrayQuadratic(arr) {
  let maxSum = -Infinity;
  const n = arr.length;

  for (let i = 0; i < n; i++) {        // O(n)
    let currentSum = 0;
    for (let j = i; j < n; j++) {     // O(n)
      currentSum += arr[j];        // O(1)
      if (currentSum > maxSum) maxSum = currentSum; // O(1)
    }
  }
  return maxSum; // O(1)
}
```

$T(n) = O(n) * O(n) = O(n^2)$

Time Complexity: $O(n^2)$

Space Complexity: O (1)

2- Optimized Solution (Kadane's Algorithm) : optamized لينك فايل ال

Uses dynamic programming

Processes the array in a single pass

Maintains current and maximum sums efficiently

```
function maxSubarrayKadane(arr) {
  let maxSum = arr[0];
  let currentSum = arr[0];

  for (let i = 1; i < arr.length; i++) { // O(n)
    currentSum = Math.max(arr[i], currentSum + arr[i]); // O(1)
    maxSum = Math.max(maxSum, currentSum);          // O(1)
  }

  return maxSum; // O(1)
}
```

T(n)=O(n)

Time Complexity: O(n)

Space Complexity: O(1)

Implementation Details:-

نفذناه بال java script

عشان خلفيتنا اكتر ك فرونت اند

ف طبقنا ال algorithm

و ربطناها ب ui كويس جاهزه امثله فيه تجرب ممكن بردو موجود والرابط .

https://samahcodes.github.io/Algorithm/

Programming Language: JavaScript

Execution Environment: GUI as a website

Files:

algorithm1.js → Naive implementation

```
function maxSubarrayNaive(arr) {
    let maxSum = -Infinity;

    for (let i = 0; i < arr.length; i++) {
        let currentSum = 0;
        for (let j = i; j < arr.length; j++) {
            currentSum += arr[j];
            maxSum = Math.max(maxSum, currentSum);
        }
    }

    return maxSum;
}

```

algorithm2.js → Optimized (Kadane) implementation

```
function maxSubarrayKadane(arr) {
    let currentSum = arr[0];
    let bestSum = arr[0];

    for (let i = 1; i < arr.length; i++) {
        currentSum = Math.max(arr[i], currentSum + arr[i]);
        bestSum = Math.max(bestSum, currentSum);
    }

    return bestSum;
}

```

Theoretical Analysis:- اللينك

Theoretical analysis focuses on time and space complexity:

Algorithm	Time Complexity	Space Complexity
Naive	$O(n^2)$	O(1)
Optimized (Kadane)	O(n)	O(1)

The optimized algorithm is theoretically superior and more scalable for large input sizes.

Empirical Analysis:- اللينك

Empirical analysis was conducted by running both algorithms on datasets of increasing sizes and measuring execution time.

Independent Variable: Input size (n)

Dependent Variable: Execution time (milliseconds)

Measurement Method: JavaScript timing functions

Environment: Same machine and runtime for all tests

Input Size (n)	Execution Time (ms) In Kadane	Execution Time (ms) In Brute Force
1,000	0.05	1000
10,000	0.30	10,000
100,000	3.10	100,000
1,000,000	30.40	1,000,000

Observations:

Kadane’s algorithm shows linear growth in execution time.

The naive approach becomes impractical for large input sizes.

Empirical results confirm theoretical expectations.

Comparison Summary:-

Aspect	Naive	Optimized
Performance	Slow	Fast
Scalability	Poor	Excellent
Practical Use	Limited	Recommended

🌐 Live Version :
https://samahcodes.github.io/Algorithm/

📎 Video Link:

https://drive.google.com/drive/folders/1cWC8_G2K1O8I5F3kcUSsveSenw5kVH7L?usp=sharing

Conclusion: -

This project demonstrates how algorithmic optimization significantly improves performance.

Kadane's Algorithm proves to be both theoretically and empirically superior to the naive approach, making it the preferred solution for real-world applications.