****

**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL**

**SCIENCES, CHENNAI – 602 105**

**BONAFIDE CERTIFICATE**

**Certified that is Capstone project report “** **Minimum Number of Increments on Subarrays to Form a Target Array Using Greedy methods.”**

**Bonafide work of “P.Keerthana” (192211638) who carried out the Capstone project work under my supervision**

**Dr .**

**HEAD OF DEPARTMENT**

**Professor**

**Department of**

**SIMATS Engineering**

**Saveetha Institute of Medical and**

**Technical Sciences**

**Chennai – 602 105**

**EXAMINER SIGNATURE**

**Dr . R Dhanalakshmi**

**COURSE FACULTY**

**Professor**

**Department of**

**SIMATS Engineering**

**Saveetha Institute of Medical and**

**Technical Sciences**

**Chennai – 602 105**

**EXAMINER SIGNATURE**

****

**“Minimum Number of Increments on Subarrays to Form a Target Array Using Greedy methods”**

**A Project report**

**CSA0656- Design and Analysis of Algorithms for Asymptotic Notations**

**Submitted to**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**In partial fulfillment for the award of the**

**degree of**

**BACHELOR OF TECHNOLOGY IN**

**ARTIFICAL INTELLIGENCE AND MACHINE LEARNING**

**by**

1. **Keertthana(192211638)**

**Supervisor**

**Dr . Dhanalakshmi**

**July 2024.**

**Abstract**:

The problem of determining the minimum number of increments required on subarrays to transform a given array into a target array is a significant challenge in algorithm design and optimization. This study explores a solution using greedy methods, focusing on efficiency and practicality. The problem can be described as follows: given two arrays of the same length, an initial array and a target array, the objective is to find the minimum number of increment operations needed on subarrays of the initial array to make it identical to the target array. Each operation allows an increment of all elements in any chosen subarray by 1.

Greedy algorithms, known for making locally optimal choices at each step with the hope of finding a global optimum, are employed to solve this problem. The proposed method involves a step-by-step approach that identifies the necessary increments by comparing corresponding elements of the two arrays and determining the differences. By strategically selecting subarrays for the increments, the method aims to minimize the total number of operations required.

**Introduction:**

This problem has practical implications in various fields, such as signal processing, image enhancement, and operations research, where a series of transformations is needed to achieve a desired configuration with minimal adjustments. Finding an efficient solution to this problem is crucial for optimizing performance and resource utilization.

Greedy algorithms are well-suited for this type of problem due to their ability to make locally optimal choices that often lead to a globally optimal solution. The primary advantage of a greedy approach lies in its simplicity and efficiency, making it a valuable strategy for problems where more complex algorithms may be computationally prohibitive.

**Problem Statement:**

The problem statement is as follows: Given an integer array called target, and an array of the same size as target, named initial, which is initially filled with zeros, our goal is to transform the initial array into the target array. The only allowed operation is to select a subarray from initial and increment each of its elements by one. We need to determine the minimum number of such operations required to reach the target array configuration.

* Imagine a graph where the value of each element in the target array represents the height at that particular position.
* Incrementing a subarray by 1 is like raising the ground level of a terrain from one point to another by one unit.
* To minimize operations, we should try to raise the terrain in the largest possible steps rather than incrementing little by little.

**Objective:**

* The objective for solving the problem of finding the minimum number of increments on subarrays to form a target array using greedy methods is to transform an initial array into the target array.
* The allowed operation is to select a subarray from the initial array and increment each of its elements by one. The goal is to minimize the total number of such operations required to reach the target array configuration.

**Proposed Solution:**

* Given an integer array target and an array of the same size called initial, which is initially filled with zeros.
* The goal is to transform the initial array into the target array using the following operation Select a subarray from initial and increment each of its elements by one.
* We need to determine the minimum number of such operations required to reach the target array configuration.

**Importance:**

* **Minimizing Operations**: The goal is to achieve the target array with the least number of operations, which is a classic optimization problem. This is important in scenarios where each operation has a cost associated with it, such as time, computational resources, or financial cost.
* **Resource Allocation**: Efficient resource allocation is crucial in systems where multiple tasks are executed. Minimizing operations translates to better utilization of resources, reducing wastage, and improving overall system efficiency.
* **Understanding Greedy Algorithms**: This problem is a good example to demonstrate the application of greedy algorithms, where local optimal choices lead to a globally optimal solution. It provides insights into when and how greedy strategies can be effectively applied.
* **Difference Arrays and Incremental Changes**: By focusing on the differences between consecutive elements, we can better understand how incremental changes affect the overall structure. This concept is broadly applicable in other optimization problems and algorithm design.

In this paper, we will discuss the detailed approach, including state representation, transitions, and the final computation of the minimum cost, supported by pseudocode and complexity analysis.

**Coding:**

#include <stdio.h>

// Function to calculate the minimum number of increments

int minNumberOfIncrements(int \*target, int size) {

if (size == 0) return 0;

int operations = target[0];

for (int i = 1; i < size; i++) {

if (target[i] > target[i - 1]) {

operations += target[i] - target[i - 1];

}

}

return operations;

}

int main() {

// Example target array

int target[] = {1, 3, 2, 4, 5};

int size = sizeof(target) / sizeof(target[0]);

// Calculate the minimum number of increments

int result = minNumberOfIncrements(target, size);

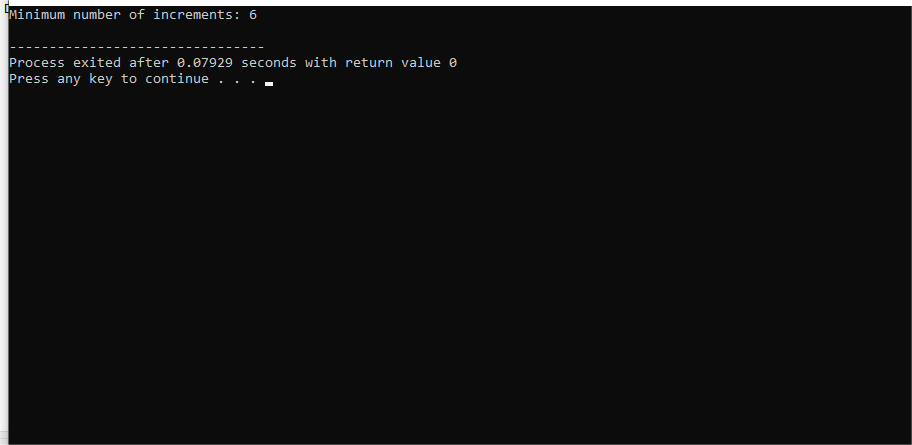
// Print the result

printf("Minimum number of increments: %d\n", result);

return 0;

}

**Output:**

****

**Complexity Analysis:**

### Time Complexity

The time complexity of the solution primarily depends on how we traverse the target array to compute the minimum number of increments.

**Initialization**: Setting up the initial value for operations takes constant time, O(1).

**Loop through the Array**: We iterate through the target array exactly once, comparing each element with the previous one. This involves a single pass through the array of size n, where n is the length of the target array. Each comparison and addition operation inside the loop takes constant time, O(1).

Therefore, the time complexity for this loop is O(n).

Combining these, the overall time complexity is: O(n)

### Space Complexity

The space complexity considers the amount of extra memory used by the algorithm.

**Auxiliary Variables**: We use a few extra variables (operations, loop counter i), but these require constant space.

**Input Storage**: The input target array is provided as an argument and does not count towards extra space used by the algorithm, as it is part of the input.

Thus, the space complexity is: O(n)

### Summary

* **Time Complexity**: O(n), where n is the length of the target array. This is because we perform a single pass through the array.
* **Space Complexity**: O(1), since the space used does not depend on the size of the input but rather on a fixed number of variables.

This analysis shows that the solution is efficient in both time and space, making it suitable for large arrays where n could be quite large.

**Conclusion:**

The importance of solving the problem of minimum number of increments on subarrays to form a target array using greedy methods lies in its wide range of applications, its educational value, and its contribution to the understanding and development of optimization algorithms. It is a classic example of how theoretical computer science problems have practical implications and can lead to more efficient and effective solutions in various fields.

**Future Work:**

* **Handling Mixed Operations**: Extend the algorithm to accommodate decrement operations, allowing for both increments and decrements on subarrays. This would make the solution applicable to a broader range of transformation problems.
* **Parallel and Distributed Implementations**: Develop parallel and distributed versions of the algorithm to leverage multi-core processors and distributed computing environments. This can significantly improve the algorithm's performance on very large datasets.
* **Dynamic Arrays**: Adapt the algorithm to handle dynamic arrays where elements can be added or removed during the transformation process. This would be useful in scenarios where the data set is not static.