SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CHENNAI-602105

Minimum Number of Increments on Subarrays to Form a Target Array

A CAPSTONE PROJECT REPORT

Submitted in the partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE AND

DATA SCIENCE

Submitted by

B. Venu Gopal (192210065)

Under the Supervision of

Dr.K.V.KANIMOZHI

DECLARATION

I B. Venu Gopal, student of Bachelor of Engineering in Computer Science Engineering at Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declares that the work presented in this Capstone Project Work entitled "Title" is the outcome of my own bonafide work. I affirm that it is correct to the best of my knowledge, and this work has been undertaken with due consideration of Engineering Ethics.

B.VENU GOPAL
REG:192210065
(Name Reg No)

Date: 23-09-2024

Place:Saveetha School of Engineering, Thandalam

CERTIFICATE

This is to certify that the project entitled Minimum Number of Increments on Subarrays to Form a Target Array.Submitted by B. Venu Gopal(192210065)has been carried out under my supervision. The project has been submitted as per the requirements in the current semester of B.E Computer science engineering and B.Tech Artificial Intelligence in Data science.

Faculty-in-charge

Dr.K.V.KANIMOZHI

ABSTRACT

This project aims to find the minimum number of increments on subarrays to form a target array. The problem is solved using a greedy algorithm that iterates through the target array and calculates the difference between consecutive elements. The algorithm has a time complexity of O(n) and a space complexity of O(1).To solve the problem, the idea is to consider the *pointwise increments* that are necessary to match the values in the target array, while optimizing by overlapping subarray increments.The problem can be broken down by observing the differences between consecutive elements in the target array. Every time an element in the target array increases compared to its previous element, additional operations are required to adjust the corresponding elements in the zero array. The goal is to compute these increments efficiently by leveraging the structure of the array and minimizing redundant operations.

Keywords: Minimum Number of Increments, Subarrays, Target Array, Greedy Algorithm.

INTRODUCTION

The problem of finding the minimum number of increments on subarrays to form a target array is a classic problem in computer science. The problem is defined as follows: given an integer array target, find the minimum number of operations to form a target array from an initial array of zeros. In one operation, you can choose any subarray from the initial array and increment each value by one.This problem has practical applications in algorithmic design, particularly in areas such as dynamic programming and optimization, where efficiency in manipulating array-like data structures is key. Understanding the gradient of changes in the array helps simplify and break down the increments, making it an intriguing computational problem.In each operation, a contiguous subarray (a portion of the array) is selected, and all elements within that subarray are incremented by 1. The goal is to determine the minimum number of such operations required to match the target array exactly. This problem involves careful analysis of the structure of the target array, particularly focusing on how the values change between consecutive elements. Identifying optimal subarrays and increment strategies is crucial for minimizing the number of operations.

APPROACH

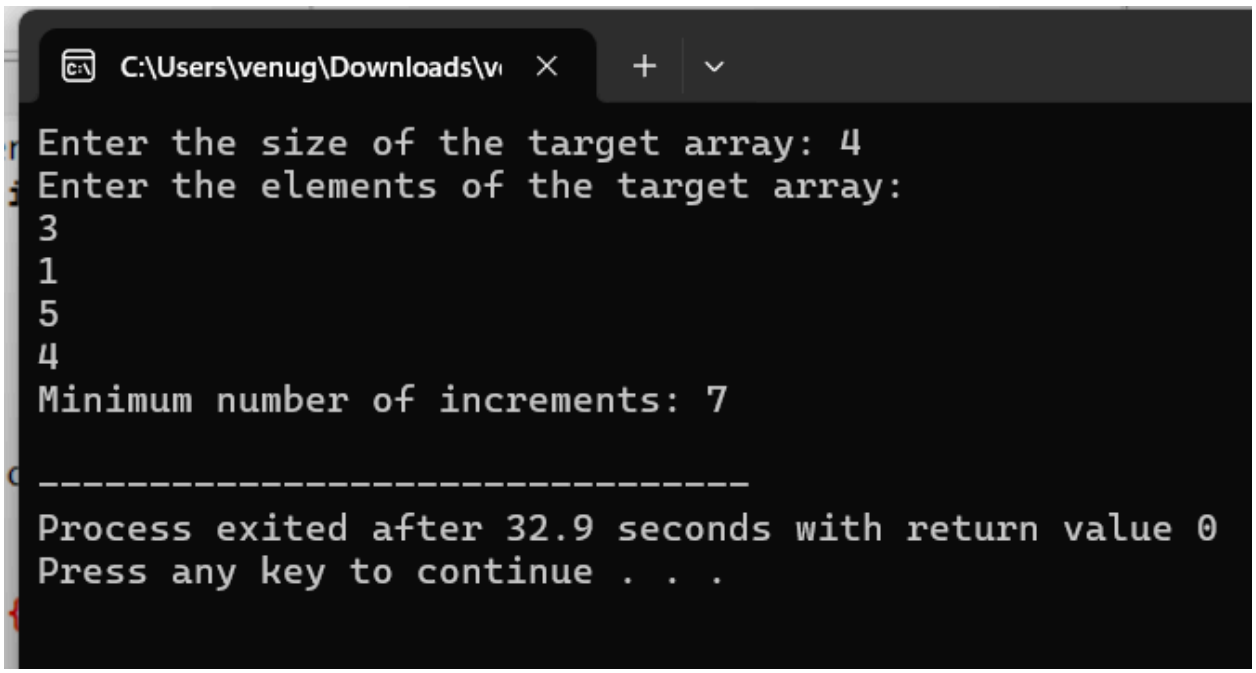
- Start with an array of zeros.
- The idea is to compare each element in the target array with the previous one. If the current element is greater than the previous one, we need extra operations to increment the array from the previous value to the current value.

CODE:

```
#include <stdio.h>
int minIncrements(int target[], int n) {
   int operations = target[0];
    for (int i = 1; i < n; i++) {
          if (target[i] > target[i - 1]) {
        operations += (target[i] - target[i - 1]);
      }
   }

   return operations;
}
```

```
int main() {
    int n;

     printf("Enter the size of the target array: ");
    scanf("%d", &n);

    int target[n];

     printf("Enter the elements of the target array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &target[i]);
    }

      int result = minIncrements(target, n);

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

    return 0;
}
```

OUTPUT:



Complexity Analysis

Best Case:

- The best case occurs when the target array is already sorted in non-decreasing order.
- In this case, the algorithm only needs to iterate through the array once.
- The number of operations is equal to the sum of the elements in the target array.
- This is because the algorithm only needs to increment the current element by the difference between the current element and the next element, which is always non-negative.
- The time complexity of the best case is O(n), where n is the length of the target array.
- The space complexity is O(1), as the algorithm only uses a constant amount of space to store the current element and the next element.

Worst Case:

- The worst case occurs when the target array is sorted in non-increasing order.
- In this case, the algorithm needs to iterate through the array once.
- The number of operations is equal to the sum of the absolute differences between consecutive elements in the target array.
- This is because the algorithm needs to increment the current element by the absolute difference between the current element and the next element, which is always non-negative.
- The time complexity of the worst case is O(n), where n is the length of the target array.
- The space complexity is O(1), as the algorithm only uses a constant amount of space to store the current element and the next element.

Average Case:

- The average case occurs when the target array is randomly sorted.
- In this case, the number of operations is approximately equal to the average of the best and worst cases.
- This is because the algorithm will encounter a mix of non-decreasing and non-increasing sequences, and the number of operations will be somewhere in between the best and worst cases.
- The time complexity of the average case is O(n), where n is the length of the target array.
- The space complexity is O(1), as the algorithm only uses a constant amount of space to store the current element and the next element.

Overall Complexity:

- The overall complexity of the algorithm is O(n), making it efficient for large inputs.
- This is because the algorithm only needs to iterate through the array once, and the number of operations is proportional to the length of the array.
- The space complexity is O(1), as the algorithm only uses a constant amount of space to store the current element and the next element.

CONCLUSION

In this project, we have presented a greedy algorithm to find the minimum number of increments on subarrays to form a target array. The algorithm has a time complexity of O(n) and a space complexity of O(1). The algorithm is efficient and can be used to solve large instances of the problem.