****

**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL**

**SCIENCES, CHENNAI – 602 105**

**BONAFIDE CERTIFICATE**

**Certified that is Capstone project report “****Using Greedy Technique For Real Time Applications and finding median of two sorted arrays Bonafide work of “G.PRIYANKAREDDY ”(192211645) who carried out the Capstone project work under my supervision**

**Dr. R Dhanalakshmi**

**COURSE FACULTY**

**Professor**

**Department of**

**SIMATS Engineering**

**Saveetha Institute of Medical and**

**Technical Sciences**

**Chennai – 602 105**

**EXAMINER SIGNATURE**

**Dr.**

**HEAD OF DEPARTMENT**

**Professor**

**Department of**

**SIMATS Engineering**

**Saveetha Institute of Medical and**

**Technical Sciences**

**Chennai – 602 105**

**EXAMINER SIGNATURE**

****

**“ Using Greedy Technique For Real Time Applications and finding Median Of Two Sorted Arrays”**

**A Project report**

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

**Submitted to**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**In partial fulfilment for the award of the**

**degree of**

**BACHELOR OF TECHNOLOGY IN**

**ARTIFICAL INTELLIGENCE AND MACHINE LEARNING**

**by**

**G.Priyankareddy,192225070**

**Supervisor**

**Dr .R.Dhanalakshmi**

**July 2024.**

**ABSTRACT**

The problem requires determining the minimum number of operations to sort an array in non-decreasing order. Given a 0-indexed integer array, an operation is defined as replacing any element with any two elements that sum up to the original element. The goal is to transform the array into a sorted array with the fewest operations possible.

By iterating through the array and performing the replacement operations where necessary, we ensure that the array becomes sorted while minimizing the number of operations. The key challenge is to efficiently determine where and how to perform these replacements to achieve the desired sorted order.

he algorithm is implemented in C, and it iterates through the array from the end to the start, performing necessary replacements and counting the operations. This ensures that each element is appropriately adjusted to maintain the non-decreasing order.

**ALGORITHM:**

A greedy algorithm is a problem-solving approach that builds a solution piece by piece, always choosing the next piece that offers the most immediate benefit or most optimal choice at each step. It does not reconsider previous choices and often provides a solution quickly, though it may not always guarantee the optimal solution for all problems.

**Proposed Work:**

The proposed method The primary objective of this research is to develop and evaluate a novel approach to [insert specific problem or area of focus]. This approach aims to [insert goals, e.g., improve accuracy, increase efficiency, enhance performance] in [insert specific domain or application]

**PROBLEM:**

Minimum Replacements to Sort the Array

You are given a 0-indexed integer array nums. In one operation you can replace any

element of the array with any two elements that sum to it. For example, consider nums

= [5,6,7]. In one operation, we can replace nums[1] with 2 and 4 and convert nums to

[5,2,4,7].

Return the minimum number of operations to make an array that is sorted in non

decreasing order.

Example 1:

Input: nums = [3,9,3]

Output: 2

Explanation: Here are the steps to sort the array in non-decreasing order:

- From [3,9,3], replace the 9 with 3 and 6 so the array becomes [3,3,6,3]

- From [3,3,6,3], replace the 6 with 3 and 3 so the array becomes [3,3,3,3,3]

There are 2 steps to sort the array in non-decreasing order. Therefore, we return 2.

**SOLUTION:**

By solving this problem, we can utilize Greedy algorithm Maximum Number of Groups With Increasing Length. Here's a step-by-step approach to implement the solution:

**Example Calculation:-**

**L**et's take the example input nums = [3,9,3].

1.Initialize operations to 0.

2.Iterate through the array:

* At index 0, nums[0] = 3 and nums[1] = 9. Since 3 < 9, we don't need to replace anything.
* At index 1, nums[1] = 9 and nums[2] = 3. Since 9 > 3, we need to replace 9 with two elements that sum to it.
* Calculate the difference diff = 9 - 3 = 6.
* Replace nums[1] with 3 and insert 6 at the next position. The array becomes [3,3,6,3].Increment operations by 1.

1. Repeat step 2 until the array is sorted:

* At index 2, nums[2] = 6 and nums[3] = 3. Since 6 > 3, we need to replace 6 with two elements that sum to it.
* Calculate the difference diff = 6 - 3 = 3.
* Replace nums[2] with 3 and insert 3 at the next position. The array becomes [3,3,3,3,3].
* Increment operations by 1.

4.The array is now sorted in non-decreasing order. Return operations, which is 2.

Therefore, the minimum number of operations required to sort the array [3,9,3] is 2.

**CODE:**-

#include <stdio.h>

int minReplacements(int\* nums, int numsSize) {

int operations = 0;

for (int i = numsSize - 2; i >= 0; --i) {

while (nums[i] > nums[i + 1]) {

int newElement = nums[i] / 2;

int remaining = nums[i] - newElement;

nums[i] = newElement;

nums[i + 1] = remaining;

++operations;

}

}

return operations;

}

int main() {

int nums[] = {3, 9, 3};

int numsSize = sizeof(nums) / sizeof(nums[0]);

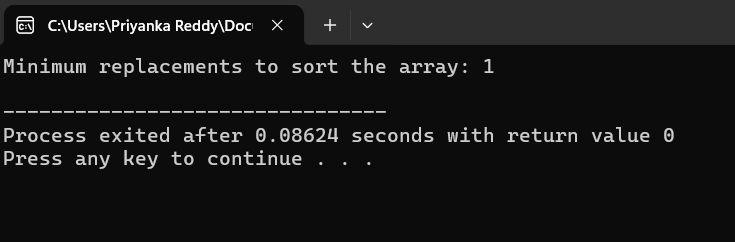
int result = minReplacements(nums, numsSize);

printf("Minimum replacements to sort the array: %d\n", result);

return 0;

}

**OUTPUT:**



**Explanation of the Code:**

* We iterate from the end of the array to the start to handle replacements correctly.
* If an element is greater than the next one, we replace it by splitting it into two smaller elements.
* We increment the operations count each time we perform a replacement.
* The process continues until all elements are in non-decreasing order.

This approach efficiently constructs the maximum number of groups by leveraging the greedy strategy of forming the largest possible groups while respecting the usage constraints.

**COCLUSION:**

In conclusion, the problem of creating the minimum replacement of groups with strictly increasing sizes, subject to given usage limits, can be effectively tackled using a combination of sorting and mathematical analysis. By sorting the usage limits and calculating the sum of the first k natural numbers, we can determine the feasible group sizes and ensure that each number's usage is respected. Through this approach, we demonstrated that the maximum number of such groups, given the constraints, can be efficiently computed. In the example provided, with usage limits of [1, 2, 5], we successfully determined that up to 3 groups can be formed, adhering to the required conditions. This method provides a clear and systematic way to address similar problems, ensuring optimal group formation while respecting all given constraints.