1 Title

1.1 Abstract

The goal of this project is to implement a simple and efficient algorithm to solve the problem of finding the maximum subarray sum in a given array of integers. The algorithm is based on the divide and conquer paradigm and has a time complexity of O(n log n). The algorithm is implemented in Python and tested on a variety of test cases to ensure its correctness and efficiency. The results show that the algorithm is able to find the maximum subarray sum in a given array of integers in a reasonable amount of time.

1.2 Introduction

The maximum subarray sum problem is a classic problem in computer science that involves finding the maximum sum of a contiguous subarray in a given array of integers. The problem can be solved using a variety of algorithms, including brute force, divide and conquer, and dynamic programming. In this project, we implement a simple and efficient algorithm based on the divide and conquer paradigm to solve the maximum subarray sum problem. The algorithm has a time complexity of O(n log n) and is implemented in Python. We test the algorithm on a variety of test cases to ensure its correctness and efficiency.

1.3 Methodology

The algorithm is based on the divide and conquer paradigm and works as follows: 1. Divide the array into two halves. 2. Recursively find the maximum subarray sum in the left half, right half, and the subarray that crosses the midpoint. 3. Return the maximum of the three sums.

The algorithm is implemented in Python as follows:

```
def max_subarray_sum(arr, low, high):
    if low == high:
        return arr[low]

    mid = (low + high) // 2

    left_sum = max_subarray_sum(arr, low, mid)
    right_sum = max_subarray_sum(arr, mid + 1, high)
    cross_sum = max_crossing_sum(arr, low, mid, high)

    return max(left_sum, right_sum, cross_sum)
```

The ‘max_crossing_sum‘ function is used to find the maximum subarray sum that crosses the midpoint of the array. It is implemented as follows:

```
def max_crossing_sum(arr, low, mid, high):
    left_sum = float('-inf')
```

```
sum = 0
for i in range(mid, low - 1, -1):
    sum += arr[i]
    left_sum = max(left_sum, sum)

right_sum = float('-inf')
sum = 0
for i in range(mid + 1, high + 1):
    sum += arr[i]
    right_sum = max(right_sum, sum)

return left_sum + right_sum
```