Sum of ‘n’ Terms of a Sequence

RHYTHM JAIN (IIT2021505), NITESH KUMAR SHAH (IIB2021002),
SHIVANI (IIT2021502),ADITYA YALAMANCHI (IIB2021001)

4th Semester BTech Information Technology
Indian Institute of Information Technology, Allahabad

***Abstract* - This report introduces the algorithm to print the sum of first n terms of the sequence 1-3+5-7+9…. . The algorithm has O(1) as its best, average, worst complexity.**

INTRODUCTION

A mathematical sequence is an ordered list of numbers or terms that follow a specific pattern or rule. This Algorithm calculates the sum of a mathematical sequence. The sequence is created by alternating between adding and subtracting odd numbers starting from 1. The code takes the value of n, which is the number of terms in the sequence, as input and calculates the sum of the sequence accordingly.

For creating a new algorithm from scratch, we look into the many crucial aspects of algorithm designing and analysis. The algorithm created is tested for different sample test cases for time complexity, space complexity and accuracy. In our case we tend to focus on establishing a rule or a relationship between the time and input data. Our algorithm is an in-place algorithm and hence efficient for space and time.

This report further contains -

II. Algorithm Design
III. Algorithm Analysis
IV. A Posteriori Analysis
V. Conclusion

ALGORITHM DESIGN

ALGORITHM - I

This algorithm calculates the sum of the first n terms of the sequence 1-3+5-7+...n. The algorithm is implemented in C++ and uses a for loop to iterate through the terms of the sequence

1. The algorithm takes input from the user for the value of n and uses a variable "k" which starts from 1 and increases by 2 in each iteration to represent the current term of the sequence.

2. The algorithm uses a variable "sum" to keep track of the running total of the sum and a variable "t" to keep track of the time complexity.

3. In each iteration, the algorithm checks if the current term is odd or even. If the current term is odd, the algorithm adds the value of "k" to the variable "sum" and prints the current term followed by a "-" sign.
4. If the current term is even, the algorithm subtracts the value of "k" from the variable "sum" and prints the current term followed by a "+" sign.

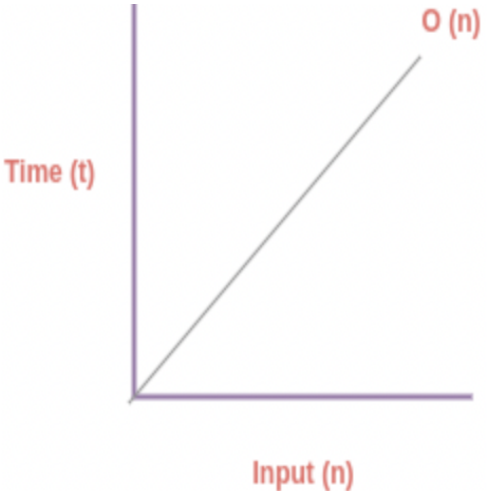
5. After all the terms have been processed, the algorithm prints the final sum. Also,

it prints the value of "t" which represents the total number of operations used in the algorithm.

III. ALGORITHM ANALYSIS

A. Time Complexity (For Algorithm-I)

- The time complexity of the function is O(n) because the number of operations performed is directly proportional to the value of the input n.



B. Space Complexity(For Algorithm-I)

The space complexity of all the algorithms is O(1) for the best case, the average case and the worst case because it uses a single variable and no additional memory is required, as we are not using arrays or vectors of any size.

IV. A POSTERIORI ANALYSIS

We are using a variable t to count the number of operations required for a particular value of n;

For n=10 number of Operations=85

For n=100 number of Operations=850

For n=1000 number of Operations=8500

For n=10000 number of Operations=85000

as n increases by ten folds number of Operations also increase by ten folds hence Time Complexity of this approach is O(n);

```
nitesh_shah@Niteshs-MacBook-Air DAA % cd "/Us
ers/nitesh_shah/Desktop/ComputerScience/DAA/"
 && g
++ Assignment1.cpp -o Assignment1 && "/Users/
nitesh_shah/Desktop/ComputerScience/DAA/"Assi
gnmen
t1
4
Enter the value of n
10
sum of the subsequence = -10
Number of operations are  85
Enter the value of n
100
sum of the subsequence = -100
Number of operations are  850
Enter the value of n
1000
sum of the subsequence = -1000
Number of operations are  8500
Enter the value of n
10000
sum of the subsequence = -10000
Number of operations are  85000
```

PSEUDO CODE (Algorithm - I)

```
long long n
    print Enter the value of n
    take input n
    long long k=1
    long long sum=0;
    long long t=0;
    for(i=1 to i=n)
        t=t+4;
        if(i%2==1){
            t++;
            sum+=k;
            t+=2;
        }
        else{
            sum-=k;
            t+=2;
        }
        k+=2;
        t+=2;
    }
    Print sum of the subsequence = sum
    Print Number of operations are t
```

V. ALGORITHM DESIGN
ALGORITHM - II

The first algorithm's optimized solution is provided by our second algorithm. Below, we'll go over how the upgraded version works.

The general terms of the sequence $f_{(n)}$ is

$$f_{(n)} = (2n-1)(-1)^{n-1}$$

For example

$$f_{(1)} = (2\times 1-1)(-1)^{1-1} = 1$$

$$S_{(1)} = 1$$

$$f_{(2)} = (2\times 2-1)(-1)^{2-1} = -3$$

$$S_{(2)} = 1-3 = -2$$

$$f_{(3)} = (2\times 3-1)(-1)^{3-1} = 5$$

$$S_{(3)} = 1-3+5 = 3$$

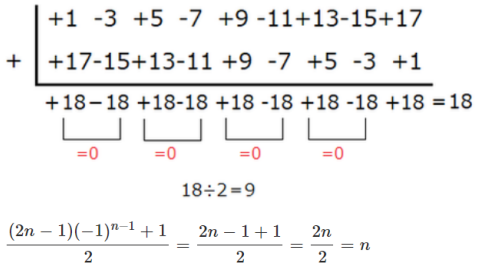
$$f_{(4)} = (2\times 4-1)(-1)^{4-1} = -7$$

$$S_{(4)} = 1-3+5-7 = -4$$

$$f_{(5)} = (2\times 5-1)(-1)^{5-1} = 9$$

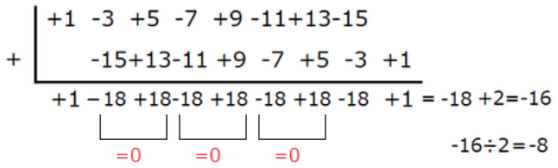
$$S_{(5)} = 1-3+5-7+9 = 5$$

When n is an odd number,the sum is n.
The Example when n=9.



$$\frac{f_{(n)}+1}{2} = \frac{(2n-1)(-1)^{n-1}+1}{2} = \frac{2n-1+1}{2} = \frac{2n}{2} = n$$

When n is an even number, the sum is -n.The example when n=8.



So,this is the idea behind the optimized solution

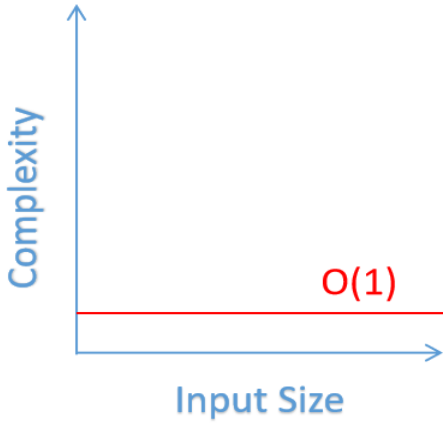
- When n is an odd number,the sum S(n)=n.
- When n is an even number,the sum S(n)= -n.

This code is a simple program that calculates the sum of a subsequence of integers and the number of operations performed to get the sum.
The code prompts the user to enter the value of n, and calculates the sum of the subsequence (either n or -n, depending on whether n is even or odd).
Finally, it outputs the sum and the number of operations

III. ALGORITHM ANALYSIS

C. Time Complexity (For Algorithm-II)

- The time complexity of the function is O(1) because the number of operations performed is constant regardless of the value of the input n



D. Space Complexity(For Algorithm-I)

The space complexity of all the algorithms is O(1) for the best case, the average case and the worst case because it uses a single variable and

A POSTERIORI ANALYSIS

We are using a variable t to count the number of operations required for a particular value of n;

For n=10 number of Operations=1

For n=100 number of Operations=1

For n=1000 number of Operations=1

For n=10000 number of Operations=1

As increases by ten folds number of Operations remains same hence Time Complexity of this approach is O(1);

```
nitesh_shah@Niteshs-MacBook-Air DAA % cd "
ers/nitesh_shah/Desktop/ComputerScience/DA
 && g++ Optimizedassign1.cpp -o Optimizeda
gn1 && "/Users/nitesh_shah/Desktop/Compute
ience/DAA/"Optimizedassign1
4
Enter the value of n
10
sum of the subsequence = -10
number of operations are 2
Enter the value of n
100
sum of the subsequence = -100
number of operations are 2
Enter the value of n
1000
sum of the subsequence = -1000
number of operations are 2
Enter the value of n
100000
sum of the subsequence = -100000
number of operations are 2
```

PSEUDO CODE

```
long long n;
    print Enter the value of n
    take input n
    long long t=0;
    if(n%2==0){
        print sum of the subsequence = -n
        t+=2;
    }
    else{
    print sum of the subsequence = n
        t+=2;
    }
    print number of operations are t
    }
return 0;
}
```

CONCLUSION

In order to tackle the presented problem, we used the ideas learned in class and provided an algorithm with varying temporal complexities. We finally arrived at an algorithm that operates in constant time. Additionally, we examined the method and the amount of operations for various inputs.

CODE

- https://github.com/goasres/Assignment1_DAA