**CSCI 532-01B – Algorithm Design**

Name: Rama Krishna Kamma

CWID: 50321021

-----------------------------------------------------------------------------------------------------------------------------------

1. Write a divide and conquer (recursive) algorithm RX(n) that calculates X(n) for any given n>=1. Your algorithm also prints out how many additions (+ operation) it performs in calculating X(n).

**Solution:**

**Algorithm:**

RX (int n, int &operationCount){

if n <=3 then

return 1

else

operationCount +:= 2

return RX (n – 2, operationCount) + RX (n-1, operationCount) + RX (n-3, operationCount)

}

**Program:**

//Importing the required header files

#include <iostream>

using namespace std;

// Divide and Conquer(Recursive) Algorithm

int RX(int n, int& operationCount)

{

//if n value <=3 then return 1

if (n <= 3)

{

return 1;

}

else

{

//Increment the counter value by 2 for each addition operation

operationCount += 2;

//otherwise perform function according to requirements and return it

return RX(n-2, operationCount) + RX(n-1, operationCount) + RX(n-3, operationCount);

}

}

int main() {

int operationCount = 0;

int number;

cout<<"Enter the postive number: ";

cin>>number;

if(number<1)

{

cout<<"Please enter a valid number which is greater than 0"<<endl;

return 0;

}

cout<<"The given number " << number << " value is " << RX(number, operationCount) << endl;

cout<<"The number of addition operations made for given input is " << operationCount << endl;

return 0;

}

**Output:**

**Graphical user interface, text, application

Description automatically generated**

1. Write a dynamic programming algorithm DX(n) that calculates X(n) for any given n>=1. Your algorithm also prints out how many additions (+ operation) it performs in calculating X(n).

**Solution:**

**Algorithm:**

DX(int n, int &operationCount){

int x[n+1]

for i:=1 to 3 do:

x[i] = 1

for i:=4 to n do:  
 operationCount +:= 2

x[i] = x[i-2] + x[i-1] + x[i-3]

print operationCount

return x[n]

}

**Program:**

//Importing the required header files

#include <iostream>

using namespace std;

// Dynamic Programming Algorithm

int DX(int n, int& operationCount)

{

//Declare the array of size n+1

int x[n+1];

//Initalize the first three values to one

x[1] = x[2] = x[3] = 1;

int z = 4;

//Repeat the loop for remaining values and calculate the counter for each addition

while(z <= n)

{

//Calculate the next value by using previous value

x[z] = x[z-2] + x[z-1] + x[z-3];

//Increment counter for addition operation

operationCount += 2;

z += 1;

}

//return the final value

return x[n];

}

int main() {

int operationCount = 0;

int number;

cout<<"Enter the postive number: ";

cin>>number;

if(number<1)

{

cout<<"Please enter a valid number which is greater than 0"<<endl;

return 0;

}

cout<<"The given number " << number << " value is " << DX(number, operationCount) << endl;

cout<<"The number of addition operations made for given input is " << operationCount << endl;

return 0;

}

**Output:**

Graphical user interface, text

Description automatically generated

1. Create a table in which you tabulate the number of additions RX(n) and DX(n) perform for n=5, 10, 15, 20, 25.

**Solution:**

#include <iostream>

using namespace std;

// Divide and Conquer(Recursive) Algorithm

int RX(int n, int& operationCount)

{

//if n value <=3 then return 1

if (n <= 3)

{

return 1;

}

else

{

//Increment the counter value by 2 for each addition operation

operationCount += 2;

//otherwise perform function according to requirements and return it

return RX(n-2, operationCount) + RX(n-1, operationCount) + RX(n-3, operationCount);

}

}

// Dynamic Programming Algorithm

int DX(int n, int& operationCount)

{

//Declare the array of size n+1

int x[n+1];

//Initalize the first three values to one

x[1] = x[2] = x[3] = 1;

int z = 4;

//Repeat the loop for remaining values and calculate the counter for each addition

while(z <= n)

{

//Calculate the next value by using previous value

x[z] = x[z-2] + x[z-1] + x[z-3];

//Increment counter for addition operation

operationCount += 2;

z += 1;

}

//return the final value

return x[n];

}

int main() {

//intialize the counter values to zero to calculate the no of addition operations

int count1 = 0, count2 = 0;

//Store the n values in an array

int table[] = {5, 10, 15, 20, 25};

// Output table headers

cout << " n\tRecursive(n)\t# Additions\tDynamic(n)\t# Additions\n";

int z = 0;

// Output table rows

while(z < 5) {

//Declare the result1 and result2

int result1, result2;

//Call the Recursive function and store in result 1

result1 = RX(table[z], count1);

//Call the Dynamic function and store in result 2

result2 = DX(table[z], count2);

//Display the output values

cout << "|----------------------------------------------------|\n";

cout << "|"<<table[z] << "\t" << result1 << "\t" << count1 << "\t\t"<< result2 << "\t" << count2 << "\t" << " |"<< "\n";

//Reset the counter values to zero

count1 = 0;

count2 = 0;

//Increment the z value by 1

z += 1;

}

cout << " ----------------------------------------------------\n";

return 0;

}

**Output:**

**Text

Description automatically generated**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Recursive(n)** | **No of Additions** | **Dynamic(n)** | **No of Additions** |
| 5 | 5 | 4 | 5 | 4 |
| 10 | 105 | 104 | 105 | 14 |
| 15 | 2209 | 2208 | 2209 | 24 |
| 20 | 46499 | 46498 | 46499 | 34 |
| 25 | 978793 | 978792 | 978793 | 44 |