

DIGITAL ASSIGNMENT - 3

1. Write an Open MP program using C for the following 1-D array of n-elements and perform the following task. (Find the time for each sub task and print the total time)

1.1. To initialize and print the array elements and to add and multiply a value to each element and print the same.

1.2. To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array.

1.3. To find the sum of square's, cubes of array elements and to find the maximum and minimum in an array and to display the second largest and smallest element in an array.

1.4. To print and count the prime numbers and their sum in the array.

1.5. To find the mean, standard deviation and variance of the given array elements.

1.6. To count the duplicate elements of the array and to find the frequency count of each elements.

Code:

```
#include <stdio.h>
#include <limits.h>
#include <omp.h>
float find_sqrt(float number)
{
    float sqrt=number/2,temp = 0;
    while(sqrt != temp)
    {
        temp = sqrt;
        sqrt = (number/temp + temp) / 2;
    }

    return sqrt;
}
void task1(int array[],int size)
{
    printf("Task1 - To initialize and print the array elements and to add and multiply a value to each element and print the same");
    int arr1[size],sum=0;
    long int mul=1;
    for(int i=0;i<size;i++)
    {
        arr1[i]=array[i];
        printf("%d\t",arr1[i]);
        sum+=array[i];
        mul*=array[i];
    }
}
```

```
}
printf("\nSum: %d\nMultiplication: %lu",sum,mul);
}
void task2(int array[],int size)
{
    printf("Task2 - To find the sum, odd sum, even sum of the array elements and to count
the odd and even elements in the array");
    int sum=0,odd_sum=0,even_sum=0,odd_count=0,even_count=0;
    for(int i=0;i<size;i++)
    {
        sum+=array[i];
        if(array[i]%2==0)
        {
            even_sum+=array[i];
            even_count++;
        }
        else
        {
            odd_sum+=array[i];
            odd_count++;
        }
    }
    printf("\nSum: %d\nOdd: %d\nSum of odd: %d\nEven: %d\nSum of even:
%d",sum,odd_count,odd_sum,even_count,even_sum);
}
void task3(int array[],int size)
{
    printf("Task3 - To find the sum of square's, cubes of array elements and to find the
maximum and minimum in an array and to display the second largest and smallest element
in an array");
    int
sum_square=0,sum_cube=0,max=INT_MAX,min=INT_MAX,second_smallest=INT_MAX,seco
nd_largest=INT_MAX;
    for(int i=0;i<size;i++)
    {
        sum_square+=(array[i]*array[i]);
        sum_cube+=(array[i]*array[i]*array[i]);
        if(array[i]<min)
        {
            second_smallest=min;
            min=array[i];
        }
        else if (array[i] < second_smallest && array[i] != min)
        {
            second_smallest = array[i];
        }
    }
}
```

```
}
if(array[i]>max)
{
    second_largest=max;
    max=array[i];
}
else if (array[i] > second_largest && array[i] != max)
{
    second_largest = array[i];
}
}

printf("\nSum of square: %d\nSum of cube: %d\nMaximum: %d\nMinimum: %d\nSecond
largest: %d\nSecond smallest:
%d",sum_square,sum_cube,max,min,second_largest,second_smallest);
}
void task4(int array[],int size)
{
    printf("Task4 - -To print and count the prime numbers and their sum in the array");
    int sum=0,flag=0,count=0;
    for(int i=0;i<size;i++)
    {
        flag=0;
        for(int j=2;j<array[i];j++)
        {
            if(array[i]%j==0)
            {
                flag=1;
                break;
            }
        }
        if(flag==0)
        {
            printf("\t%d",array[i]);
            count++;
            sum+=array[i];
        }
    }
    printf("\nPrime number: %d\nSum of prime no: %d",count,sum);
}
void task5(int array[],int size)
{
    printf("Task5 - To find the mean, standard deviation and variance of the given array
elements");
}
```

```
int sum=0,sum1=0;
float mean,variance,standard_deviation;
for(int i=0;i<size;i++)
{
    sum+=array[i];
}
mean=sum/(float)size;
for(int i=0;i<size;i++)
{
    sum1+=((array[i] - mean)*(array[i] - mean));
}
variance = sum1 / (float)size;
standard_deviation = find_sqrt(variance);
printf("\nMean: %.2f\nVariance: %.2f\nStandard deviation:
%.2f",mean,variance,standard_deviation);
}
void task6(int array[],int size)
{
    printf("Task6 - To count the duplicate elements of the array and to find the frequency
count of each elements");
    int fr[size],visited = -1,count=0,duplicate=0;
    for(int i = 0; i < size; i++){
        int count = 1;
        for(int j = i+1; j < size; j++){
            if(array[i] == array[j]){
                count++;
                duplicate++;
                fr[j] = visited;
            }
        }
        if(fr[i] != visited)
        {
            fr[i] = count;
        }
    }
    printf("\nTotal duplicate number: %d",duplicate);
    printf("\nElement | Frequency\n");
    for(int i = 0; i < size; i++){
        if(fr[i] != visited){
            printf("\t%d\t|\t%d\n", array[i],fr[i]);
        }
    }
}
```

```
int main()
{
    double itime,ftime,total_time;
    int array1[]={11,22,33,44,55,66,77,88,99};
    int size=sizeof(array1)/sizeof(array1[0]);
    itime = omp_get_wtime();

    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                double start = omp_get_wtime();
                printf("\nThread:%d\t|\t", omp_get_thread_num());
                task1(array1,size);
                double stop = omp_get_wtime();
                double exec_time = stop - start;
                printf("\n\nTime taken for task 1: %f\n\n", exec_time);
            }
            #pragma omp section
            {
                double start = omp_get_wtime();
                printf("\nThread:%d\t|\t",omp_get_thread_num());
                task2(array1,size);
                double stop = omp_get_wtime();
                double exec_time = stop - start;
                printf("\n\nTime taken for task 2: %f\n\n", exec_time);
            }
            #pragma omp section
            {
                double start = omp_get_wtime();
                printf("\nThread:%d\t|\t", omp_get_thread_num());
                task3(array1,size);
                double stop = omp_get_wtime();
                double exec_time = stop - start;
                printf("\n\nTime taken for task 3: %f\n\n", exec_time);
            }
            #pragma omp section
            {
                double start = omp_get_wtime();
                printf("\nThread:%d\t|\t",omp_get_thread_num());
                task4(array1,size);
            }
        }
    }
}
```

```

        double stop = omp_get_wtime();
        double exec_time = stop - start;
        printf("\n\nTime taken for task 4: %f\n\n", exec_time);

    }
    #pragma omp section
    {
        double start = omp_get_wtime();
        printf("\nThread:%d\t|\t", omp_get_thread_num());
        task5(array1,size);
        double stop = omp_get_wtime();
        double exec_time = stop - start;
        printf("\n\nTime taken for task 5: %f\n\n", exec_time);

    }
    #pragma omp section
    {
        double start = omp_get_wtime();
        printf("\nThread:%d\t|\t", omp_get_thread_num());
        task6(array1,size);
        double stop = omp_get_wtime();
        double exec_time = stop - start;
        printf("\n\nTime taken for task 6: %f\n\n", exec_time);

    }
}

ftime = omp_get_wtime();
total_time = ftime - itime;
printf("\n\nTime taken for perform the all task: %f\n\n", total_time);
}

```

The screenshot shows a C++ IDE with a file named `f1.c`. The code includes `<stdio.h>`, `<limits.h>`, and `<omp.h>`. It defines a function `find_sqrt` and two tasks, `task1` and `task2`. `task1` initializes an array, calculates the square root of each element, and prints the results. `task2` calculates the sum, odd sum, and even sum of the array elements. The compilation output window on the right shows the results of the compilation, including the thread number, total duplicated elements, and the time taken for each task.

```

1 #include <stdio.h>
2 #include <limits.h>
3 #include <omp.h>
4 float find_sqrt(float number)
5 {
6     float sqrt=number/2,temp = 0;
7     while(sqrt != temp)
8     {
9         temp = sqrt;
10        sqrt = (number/temp + temp) / 2;
11    }
12    return sqrt;
13 }
14 void task1(int array[],int size)
15 {
16     printf("Task1 - To initialize and print the array elements and to add and multiply a value to each element and print the same");
17     int arr1[size],sum=0;
18     long int mul=1;
19     for(int i=0;i<size;i++)
20     {
21         arr1[i]=array[i];
22         printf("%d\t",arr1[i]);
23         sum+=array[i];
24         mul*=array[i];
25     }
26     printf("\nSum: %d\nMultiplication: %lu",sum,mul);
27 }
28 void task2(int array[],int size)
29 {
30     printf("Task2 - To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array");
31 }
32

```

Compilation Output:

```

Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99

Time takei

Thread:25
Sum: 495
Multiplicat

Time takei

```

```

31 printf("Task2 - To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array");
32 int sum=0,odd_sum=0,even_sum=0,odd_count=0,even_count=0;
33 for(int i=0;i<size;i++)
34 {
35     sum+=array[i];
36     if(array[i]%2==0)
37     {
38         even_sum+=array[i];
39         even_count++;
40     }
41     else
42     {
43         odd_sum+=array[i];
44         odd_count++;
45     }
46 }
47 printf("\nSum: %d\nOdd: %d\nSum of odd: %d\nEven: %d\nSum of even: %d",sum,odd_count,odd_sum,even_count,even_sum);
48 }
49 void task3(int array[],int size)
50 {
51     printf("Task3 - To find the sum of square's, cubes of array elements and to find the maximum and minimum in an array and to display the second largest :");
52     int sum_square=0,sum_cube=0,max=INT_MAX,min=INT_MIN,second_smallest=INT_MAX,second_largest=INT_MIN;
53     for(int i=0;i<size;i++)
54     {
55         sum_square+=(array[i]*array[i]);
56         sum_cube+=(array[i]*array[i]*array[i]);
57         if(array[i]<min)
58         {
59             second_smallest=min;
60             min=array[i];
61         }
62     }

```

```

61     min=array[i];
62 }
63 else if (array[i] < second_smallest && array[i] != min)
64 {
65     second_smallest = array[i];
66 }
67 if(array[i]>max)
68 {
69     second_largest=max;
70     max=array[i];
71 }
72 else if (array[i] > second_largest && array[i] != max)
73 {
74     second_largest = array[i];
75 }
76 }
77 printf("\nSum of square: %d\nSum of cube: %d\nMaximum: %d\nMinimum: %d\nSecond largest: %d\nSecond smallest: %d",sum_square,sum_cube,max,min,second_largest,second_smallest);
78 }
79 void task4(int array[],int size)
80 {
81     printf("Task4 - To print and count the prime numbers and their sum in the array");
82     int sum=0,flag=0,count=0;
83     for(int i=0;i<size;i++)
84     {
85         flag=0;
86         for(int j=2;j<array[i];j++)
87         {
88             if(array[i]%j==0)
89             {
90                 flag=1;
91                 break;
92             }
93         }
94         if(flag==0)
95         {
96             printf("\t%d",array[i]);
97             count++;
98             sum+=array[i];
99         }
100     }
101     printf("\nPrime number: %d\nSum of prime no: %d",count,sum);
102 }
103 void task5(int array[],int size)
104 {
105     printf("Task5 - To find the mean, standard deviation and variance of the given array elements");
106     int sum=0,sum1=0;
107     float mean,variance,standard_deviation;
108     for(int i=0;i<size;i++)
109     {
110         sum+=array[i];
111     }
112     mean=sum/(float)size;
113     for(int i=0;i<size;i++)
114     {
115         sum1+=(array[i] - mean)*(array[i] - mean);
116     }
117     variance = sum1 / (float)size;
118     standard_deviation = sqrt(variance);
119 }
120 }
121 }
122 }

```

```

91     {
92         flag=1;
93         break;
94     }
95     if(flag==0)
96     {
97         printf("\t%d",array[i]);
98         count++;
99         sum+=array[i];
100     }
101     printf("\nPrime number: %d\nSum of prime no: %d",count,sum);
102 }
103 void task5(int array[],int size)
104 {
105     printf("Task5 - To find the mean, standard deviation and variance of the given array elements");
106     int sum=0,sum1=0;
107     float mean,variance,standard_deviation;
108     for(int i=0;i<size;i++)
109     {
110         sum+=array[i];
111     }
112     mean=sum/(float)size;
113     for(int i=0;i<size;i++)
114     {
115         sum1+=(array[i] - mean)*(array[i] - mean);
116     }
117     variance = sum1 / (float)size;
118     standard_deviation = sqrt(variance);
119 }
120 }
121 }
122 }

```

```
183 }
184 #pragma omp section
185 {
186     double start = omp_get_wtime();
187     printf("\nThread:%d\t\t", omp_get_thread_num());
188     task3(array1,size);
189     double stop = omp_get_wtime();
190     double exec_time = stop - start;
191     printf("\nTime taken for task 3: %f\n\n", exec_time);
192 }
193 #pragma omp section
194 {
195     double start = omp_get_wtime();
196     printf("\nThread:%d\t\t", omp_get_thread_num());
197     task4(array1,size);
198     double stop = omp_get_wtime();
199     double exec_time = stop - start;
200     printf("\nTime taken for task 4: %f\n\n", exec_time);
201 }
202 #pragma omp section
203 {
204     double start = omp_get_wtime();
205     printf("\nThread:%d\t\t", omp_get_thread_num());
206     task5(array1,size);
207     double stop = omp_get_wtime();
208     double exec_time = stop - start;
209     printf("\nTime taken for task 5: %f\n\n", exec_time);
210 }
211 }
212 }
213 }
```



```

200 double exec_time = stop - start;
201 printf("\n\nTime taken for task 4: %f\n\n", exec_time);
202
203 }
204 #pragma omp section
205 {
206 double start = omp_get_wtime();
207 printf("\nThread:%d\t|\t", omp_get_thread_num());
208 task5(array1,size);
209 double stop = omp_get_wtime();
210 double exec_time = stop - start;
211 printf("\n\nTime taken for task 5: %f\n\n", exec_time);
212
213 #pragma omp section
214 {
215 double start = omp_get_wtime();
216 printf("\nThread:%d\t|\t",omp_get_thread_num());
217 task6(array1,size);
218 double stop = omp_get_wtime();
219 double exec_time = stop - start;
220 printf("\n\nTime taken for task 6: %f\n\n", exec_time);
221
222 }
223 }
224
225 ftime = omp_get_wtime();
226 total_time = ftime - itime;
227 printf("\n\nTime taken for perform the all task: %f\n\n", total_time);
228
229 }
230 }
  
```

Compilation

Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99

Time take:
Thread:25
Sum: 495
Multipliat

Output:-

```

182 printf("\n\nTime taken for task 1: %f\n\n", exec_time);
183
184 #pragma omp section
185 {
186 double start = omp_get_wtime();
187 printf("\nThread:%d\t|\t", omp_get_thread_num());
188 task1(array1,size);
189 double stop = omp_get_wtime();
190 double exec_time = stop - start;
191 printf("\n\nTime taken for task 1: %f\n\n", exec_time);
192
193 }
194 #pragma omp section
195 {
196 double start = omp_get_wtime();
197 printf("\nThread:%d\t|\t",omp_get_thread_num());
198 task2(array1,size);
199 double stop = omp_get_wtime();
200 double exec_time = stop - start;
201 printf("\n\nTime taken for task 2: %f\n\n", exec_time);
202
203 }
204 #pragma omp section
205 {
206 double start = omp_get_wtime();
207 printf("\nThread:%d\t|\t", omp_get_thread_num());
208 task3(array1,size);
209 double stop = omp_get_wtime();
210 double exec_time = stop - start;
211 printf("\n\nTime taken for task 3: %f\n\n", exec_time);
212
213 }
214 #pragma omp section
215 {
216 double start = omp_get_wtime();
217 printf("\nThread:%d\t|\t",omp_get_thread_num());
218 task4(array1,size);
219 double stop = omp_get_wtime();
220 double exec_time = stop - start;
221 printf("\n\nTime taken for task 4: %f\n\n", exec_time);
222
223 }
224 }
  
```

Compilation

Thread:19 | Task6 - To count the duplicate elements of the array and to find the frequency count of each elements
Total duplicate number: 0
Element | Frequency
11 | 1
22 | 1
33 | 1
44 | 1
55 | 1
66 | 1
77 | 1
88 | 1
99 | 1

Time taken for task 6: 0.000200

Thread:25 | Task1 - To initialize and print the array elements and to add and multiply a value to each element and print the same
Sum: 495
Multiplication: 855632058110080
Time taken for task 1: 0.002699

Thread:24 | Task2 - To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array
Sum: 495
Odd: 5
Sum of odd: 275
Even: 4
Sum of even: 220

Thread:20 | Task4 - To print and count the prime numbers and their sum in the array
Prime number: 1
Sum of prime no: 11
Time taken for task 4: 0.002903

```

182 printf("\n\nTime taken for task 1: %f\n\n", exec_time);
183
184 #pragma omp section
185 {
186 double start = omp_get_wtime();
187 printf("\nThread:%d\t|\t", omp_get_thread_num());
188 task1(array1,size);
189 double stop = omp_get_wtime();
190 double exec_time = stop - start;
191 printf("\n\nTime taken for task 1: %f\n\n", exec_time);
192
193 }
194 #pragma omp section
195 {
196 double start = omp_get_wtime();
197 printf("\nThread:%d\t|\t",omp_get_thread_num());
198 task2(array1,size);
199 double stop = omp_get_wtime();
200 double exec_time = stop - start;
201 printf("\n\nTime taken for task 2: %f\n\n", exec_time);
202
203 }
204 #pragma omp section
205 {
206 double start = omp_get_wtime();
207 printf("\nThread:%d\t|\t",omp_get_thread_num());
208 task3(array1,size);
209 double stop = omp_get_wtime();
210 double exec_time = stop - start;
211 printf("\n\nTime taken for task 3: %f\n\n", exec_time);
212
213 }
214 #pragma omp section
215 {
216 double start = omp_get_wtime();
217 printf("\nThread:%d\t|\t",omp_get_thread_num());
218 task4(array1,size);
219 double stop = omp_get_wtime();
220 double exec_time = stop - start;
221 printf("\n\nTime taken for task 4: %f\n\n", exec_time);
222
223 }
224 }
  
```

Compilation

Thread:24 | Task2 - To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array
Sum: 495
Odd: 5
Sum of odd: 275
Even: 4
Sum of even: 220

Thread:20 | Task4 - To print and count the prime numbers and their sum in the array
Prime number: 1
Sum of prime no: 11
Time taken for task 4: 0.002903

Time taken for task 2: 0.002853

Thread:30 | Task5 - To find the mean, standard deviation and variance of the given array elements
Mean: 55.00
Variance: 806.67
Standard deviation: 28.40
Time taken for task 5: 0.004050

Thread:19 | Task3 - To find the sum of square's, cubes of array elements and to find the maximum and minimum in an array and to display the second largest and smallest element in an array
Sum of square: 34485
Sum of cube: 2693275
Maximum: 2147483647
Minimum: 11
Second largest: 2147483647
Second smallest: 22
Time taken for task 3: 0.004631

Time taken for perform the all task: 0.013299

2. Find the vector sum and vector product of n– elements in an array using open MP reduction clause.

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

/* Main Program */

int main()
{
    int *v1,*v2,*sum,*mul,arraysize, i;
    arraysize=10;

    /* Dynamic Memory Allocation */

    v1 = (int *) malloc(sizeof(int) * arraysize);
    v2 = (int *) malloc(sizeof(int) * arraysize);
    sum = (int *) malloc(sizeof(int) * arraysize);
    mul = (int *) malloc(sizeof(int) * arraysize);
    for (i = 0; i < arraysize; i++) {
        sum[i] = 0.0;
        mul[i] = 1.0;
    }
    printf("\nA:");
    for (i = 0; i < arraysize; i++) {
        v1[i] = i+1;
        printf(" %d,", v1[i]);
    }
    printf("\nB:");

    for (i = 0; i < arraysize; i++) {
        v2[i] = i+2;
        printf(" %d,", v2[i]);
    }
    printf("\n");

    #pragma omp parallel for reduction(+: sum[i])
    for (i = 0; i < arraysize; i++)
    {
        sum[i] = v1[i]+v2[i];
        printf("\n%d + %d = %d",v1[i],v2[i],sum[i]);
    }

    #pragma omp parallel for reduction(*: mul[i])
```

```

        for (i = 0; i < arraysize; i++)
        {
            mul[i] = v1[i]*v2[i];
            printf("\n%d * %d = %d",v1[i],v2[i],mul[i]);
        }

        free(v1);
        free(v2);
        free(mul);
        free(sum);
    }
}

```

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<omp.h>
4
5  /* Main Program */
6
7  int main()
8  {
9      int *v1,*v2,*sum,*mul,arraysize, i;
10     arraysize=10;
11
12     /* Dynamic Memory Allocation */
13
14     v1 = (int *) malloc(sizeof(int) * arraysize);
15     v2 = (int *) malloc(sizeof(int) * arraysize);
16     sum = (int *) malloc(sizeof(int) * arraysize);
17     mul = (int *) malloc(sizeof(int) * arraysize);
18     for (i = 0; i < arraysize; i++) {
19         sum[i] = 0.0;
20         mul[i] = 1.0;
21     }
22     printf("\nA:");
23

```

Compilation Output:

```

A: [ 1, 2, 3, 4
B: [ 2, 3, 4, 5
1 + 2 = 3
/home/p123

```

```

22     }
23     printf("\nA:");
24     for (i = 0; i < arraysize; i++) {
25         v1[i] = i+1;
26         printf(" %d", v1[i]);
27     }
28     printf("\nB:");
29
30     for (i = 0; i < arraysize; i++) {
31         v2[i] = i+2;
32         printf(" %d", v2[i]);
33     }
34     printf("\n");
35
36     #pragma omp parallel for reduction(+ : sum[i])
37     for (i = 0; i < arraysize; i++)
38     {
39         sum[i] = v1[i]+v2[i];
40         printf("\n%d * %d = %d",v1[i],v2[i],sum[i]);
41     }
42
43

```

Compilation Output:

```

A: [ 1, 2, 3, 4
B: [ 2, 3, 4, 5
1 + 2 = 3
/home/p123

```

```

f1.c
40
41 {
42     sum[i] = v1[i]+v2[i];
43     printf("\n%d + %d = %d",v1[i],v2[i],sum[i]);
44 }
45
46
47 #pragma omp parallel for reduction(* : mul[i])
48 for (i = 0; i < arraysize; i++)
49 {
50     mul[i] = v1[i]*v2[i];
51     printf("\n%d * %d = %d",v1[i],v2[i],mul[i]);
52 }
53
54
55 free(v1);
56 free(v2);
57 free(mul);
58 free(sum);
59
60
61 }
62

```

Compilation

```

A:[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]
B:[ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]
1 + 2 = 3
/home/p123/

```

Output:-

```

f1.c
40
41 {
42     sum[i] = v1[i]+v2[i];
43     printf("\n%d + %d = %d",v1[i],v2[i],sum[i]);
44 }
45
46
47 #pragma omp parallel for reduction(* : mul[i])
48 for (i = 0; i < arraysize; i++)
49 {
50     mul[i] = v1[i]*v2[i];
51     printf("\n%d * %d = %d",v1[i],v2[i],mul[i]);
52 }
53
54
55 free(v1);
56 free(v2);
57 free(mul);
58 free(sum);
59
60
61 }
62

```

Compilation

```

A:[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]
B:[ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]
1 + 2 = 3
/home/p12341/vpl_run.sh: line 3: 17315 Segmentation fault (core dumped) ./f1

```

```

60.
61. }

```

Success #stdin #stdout 0s 5464KB

stdin

Standard Input is empty

stdout

```

A:[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]
B:[ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]
1 + 2 = 3
2 + 3 = 5
3 + 4 = 7
4 + 5 = 9
5 + 6 = 11
6 + 7 = 13
7 + 8 = 15
8 + 9 = 17
9 + 10 = 19
10 + 11 = 21
1 * 2 = 2
2 * 3 = 6
3 * 4 = 12
4 * 5 = 20
5 * 6 = 30
6 * 7 = 42
7 * 8 = 56
8 * 9 = 72
9 * 10 = 90
10 * 11 = 110

```