Group 51- For some set of integers I, define f(I) as the number of pairs a,b in I such that a) a is strictly less than b; b) a divides b without a remainder. Write an algorithm to find a subset I of 1,2,,n such that f(I)=k given the value of n and k.

IHM2016501[1] IHM2016005[2] BIM2016004[3] IIM2016006[4] RIT2015050[5]

***Abstract*— We worked on an efficient algorithm to find a subset of the given set of n integers which has k number of pairs which follow the given constraints.**

I. INTRODUCTION AND LITERATURE SURVEY

In the question given we have to find the numberr of pairs a,b in I such that

- a)is strictly less than b.
- b) a divides b without a remainder.

Where subset I of 1,2,,nand f(I) is the number of pairs a,b in I such that f(I)=k given the value of n and k. In order to proceed furthur we need to find number of divisors of a number for this purpose we have to primefactorise the given number.Our idea is to store the Smallest Prime Factor(SPF) for every number. Then to calculate the prime factorization of the given number by dividing the given number recursively with its smallest prime factor till it becomes 1. To calculate to smallest prime factor for every number we will use the sieve of eratosthenes.

Sieve of Erastothenes is prime numbers generation technique. It is a simple, yet efficient algorithm to find all primes upto a limit by marking multiples of each prime as composite iteratively Thus in original Sieve, every time we mark a number as not-prime, we store the corresponding smallest prime factor for that number

II. ALGORITHM DESIGN

This part of the report can be further divided into following sections.

A. Input Format

In first few lines of the code we use $rand()$ function to generate a number randomly and assign it to the variable n which defines the parent set then we generate a random number and assign it to k.So,we now have two number n,k generated randomly.

- The array $final[]$ tell us if the i th element is inluded in the solution or not.
- The array $prime[]$ stores the smallest prime number which divides it.
- The array $divisor[]$ stores the number of divisors of a particular number.
- All these arrays have been declared globally.

B. Algorithm

Algorithm 1 Check Prime

```
1: Initialization : $i, j$
2: $prime[i] = 0$
3: for i 2 to $\sqrt{n}$ do
4:     if $prime[i] = i$ then
5:         for j i*i to n do
6:             if $prime[j] = j$ then
7:                 $prime[j] \leftarrow i$
8:             end if
9:         end for
10:    end if
11: end for
```

Algorithm 2 Max

```
**Initialization** : $a, b$
**if** $a <= b$ **then**
    return $b$
**end if**
return $a$
```

Algorithm 3 Min

```
**Initialization** : $a, b$
**if** $a >= b$ **then**
    return $b$
**end if**
return $a$
```

Algorithm 4 divisors

```
**Initialization** : $i, exp, fac, x, y$
**for** i 2 to $n$ **do**
    $exp \leftarrow 0$
    $fac \leftarrow 1$
    $x \leftarrow i$
    **while** $x \neq 0$ **do**
        $y \leftarrow x/prime[i]$
        **if** $y mod prime[x] = 0$ **then**
            $exp \leftarrow exp + 1$
        **else**
            $exp \leftarrow exp + 1$
            $fac \leftarrow fac * (exp + 1)$
            $exp \leftarrow 0$
        **end if**
        $x \leftarrow y$
    **end while**
    $divisor[i] \leftarrow fac$
    **if** $prime[i] = i$ **then**
        $divisor[i] \leftarrow 2$
    **end if**
    $divisor[1] \leftarrow 1;$
**end for**
```

Algorithm 5 Main function

```
**Initialization** : $i, cur = 0, i1, i2, pairs, i, l$
**Scan n , k**
**if** $k == 0$ **then**
    print '1'
    **return** 0
**end if**
**for** i 0 to a **do**
    $prime[i] = i$
**end for**
**Call function:-** checkprime( )
**Call function:-** divisors( )
**for** taken 1 to n **do**
    $cur = cur + divisor[taken] - 1$
**end for**
$taken - -$
**if** $taken == n$ AND $k > cur$ **then**
    **return** 0
**end if**
$excess = cur - k$
**if** $taken > 500$ **then**
    **for** $i taken to 1$ **do**
        **if** $prime[i] = i$ **then**
            $final[i] \leftarrow 1$
            $excess - -$
        **end if**
    **end for**
**else**
    $found \leftarrow 0;$
    **for** i = 1 to taken **do**
        $pairs \leftarrow 0;$
        **for** j = 1 to taken **do**
            **if** i != j **then**
                **if** $i mod j = 0 AND j mod i == 0$ **then**
                    $pairs + +$
                **end if**
            **end if**
        **end for**
    **end for**
    **if** $pairs = excess$ **then**
        $final[i] \leftarrow 1$
        $found \leftarrow 1;$
        break;
    **end if**
**end if**
**for** $i1 = 1 to taken$ **do**
    **for** $i2 to taken$ **do**
        **if** $i2 mod i1 = 0$ **then**
            $pairs \leftarrow 1$
        **else**
            $pairs \leftarrow 0;$
        **end if**
```

C. Output Format

As mentioned in the question we are given a set of integers n and k.We found the subset of n which has exactly k pairs which follow the given constraints and we print the subset.

D. Description

In order to find all prime number less than N, we follow the following steps:

- Create an array of size N
- Initially mark 0 and 1 as non prime numbers and assume all numbers greater than 1 to be prime.
- Let P be the smallest prime number known. Initially P will be 2.
- Mark all multiples of P (i.e. 2P, 3P, 4P, ..) as composite, such that multiple of P N.
- Find next greater number than P in the array which has not been marked as composite. If no such number is found, then stop, else update P to be that number and repeat the process of marking its multiple as composite.
- Here we modify this little such that the array holds the least prime which divides it.

Our algorithm mainly depends depends on few mathematical conclusions:

- If we consider a number X the number of divisors of that number is in the order of $X^{1/3}$.
- There are atleast 100 prime numbers between 1 to 500. -*

III. ANALYSIS

A. Analysis

1) Worst Case Analysis : For the worst case analysis major complexity comes from the preprocessing functions i.e. divisors and primecheck functions. It is important to note that for small values of n and k the algorithm will take more time than expected but we donot consider that cause it only happens for some values and for the rest large values of n and k it takes nlog(n)+n.

- t_{worst}(n) = $n(logn)$ for n$\leq$2 (We first find divisor count of all the numbers from 1 to n)

So,Worst case time complexity is in the order of n i.e,$O(nlogn)$

2) Best Case Analysis: The algorithm will have to go through all the above mentioned procedure and it doesn't depend on the input.i.e, Whatever the input is we have to preprocess the input to find the divisors and execute primecheck function. So,we can conclude that $t_{best} = t_{worst}$.

$\implies t_{best}(n) \propto n(logn)$

So,Best case time complexity is in the order of n i.e,$O(n(logn))$

3) Average Case Analysis : We know that $t_{best} \leq t_{avg} \leq t_{worst}$. But we also know that

$t_{best} = t_{worst}$.

$\implies t_{best} = t_{avg} = t_{worst}$

$\implies t_{avg}(n) \propto n(logn)$

So,Average case time complexity is $O(nlogn)$

IV. EXPERIMENTALSTUDY

The best way to study an algorithm is by graphs and profiling.

1) Graphs-Time-Complexity: From the graph it is clear that our algorithm takes same amount of time in all the cases i.e, best case and also in average and worst cases.We have obtained a linear graph i.e, it is proportional to $O(nlogn)$ in all the cases.In the graphs it is clearly shown that the for small values of n time complexity is proportional to n^2 + nlog(n) but for high values of n and k it is proportinal to nlog(n) + (constant)n. Mostly we deal with large values of n and k. So we generalise that complexity of of thisproblem is nlog(n) + n = n(log(n) + 1) which can be written to be O(nlog(n)) Graph plotted are in 3D shape which are between n (number of given elements) v/s k (required number of elements in subset) v/s time.

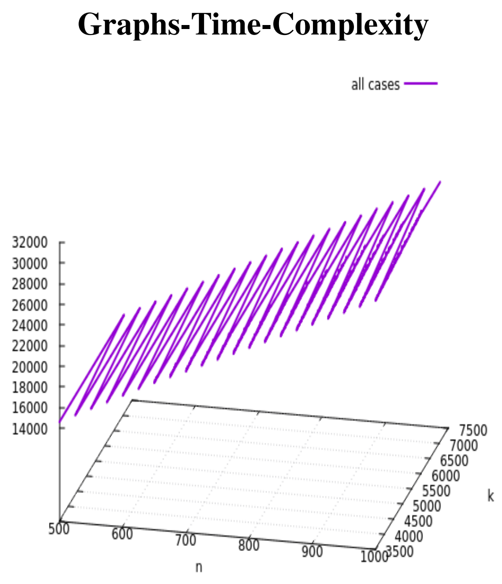

Fig. 1. size of the given set(n) Vs k Vs Time (t)

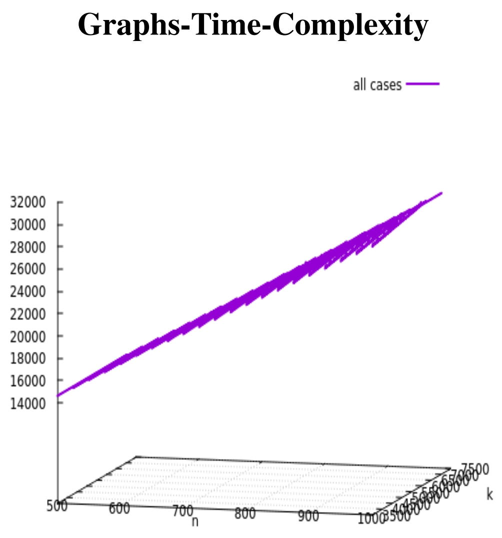

Fig. 2. size of the given set(n) Vs k Vs Time (t)

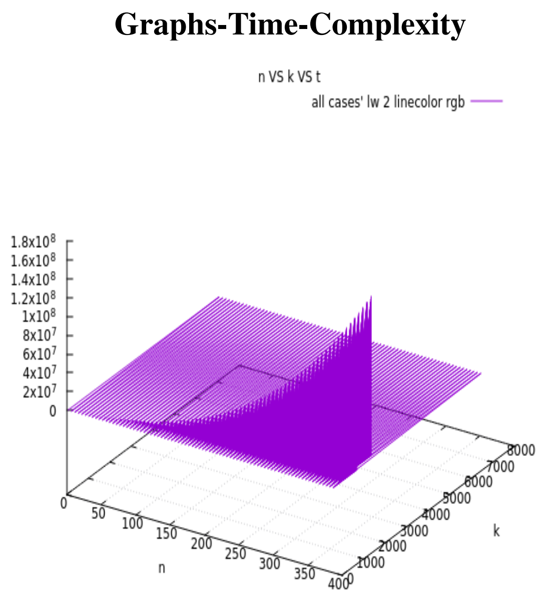

Fig. 3. size of the given set(n) Vs k Vs Time (t)

2) *Profiling:* No of $Computations$

n	k	t_{all}
50	10	1247
50	2260	1258
100	610	2668
150	3460	4101
200	460	28047
300	3760	8857
450	3985	13100
500	3515	14622
575	5690	16925
625	3890	18483
700	3935	29947
750	4343	22359
925	6950	27806
1000	7043	30167

TABLE I
SIZE OF THE SET(N)VS K VS TIME(T)$(t_{avg}, t_{best}, t_{worst})$

- We can also see that time increases as the value of n increases and $t_{worst} \propto nlog(n)$,$tavg \propto nlog(n)$,$t_{best} \propto nlog(n)$.

V. DISCUSSIONS

A. *Understanding the Use Of File Handling and generation of random numbers*

In this context the file handling is used to generate the random test cases through which analysis and experimentation of algorithm becomes easier for handling different test cases. We have used $srand()$ and $rand()$ to generate the random values to fill the matrix with the help of $< time.h >$ header file.

VI. Conclusion

- While plotting the graphs we have taken n as size of input and k as desired number of elements in subset. n varies till 1000 and k varies till 7200 as it is sufficient enough to Analyse the graph and draw conclusions.
- It is shown clearly in the analysis that we have optimized the algorithm using sieve algorithm to find the divisors.
- By making the proper analysis of the algorithm and optimizing the code we can conclude that the order of time taken by the algorithm is not dependent on the input .So we conclude that $t_{worst} = t_{best} = t_{avg}$

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Introduction to Algorithms English By Thomas H. Cormen , By Charles E. Leiserson , Ronald L. Rivest ,Clifford Stein(3rd edition)