RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

CS23331 - Design and Analysis of Algorithms

LABORATORY RECORD NOTEBOOK

NAME:	RAMYA.J.T
YEAR/SEMESTER:	B-TECH AIDS 2025 - III rd sem
BRANCH/SECTION:	B.TECH - AIDS
REGISTER NO:	211624180 1223
COLLEGE ROLL NO:	241801223
ACADEMIC YEAR:	2025 -2026

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: RAMYA J T BRANCH/SECTION: B.TECH - AIDS

ACADEMIC YEAR: 2025 -2026 SEMESTER: III rd

REGISTER NO: 2116241801223

Certified that this is a Bonafide record of work done by the above student in the **CS23331 - Design and Analysis of Algorithms** during the year 2025- 2026

[illegible]

Signature of Faculty In-charge

Submitted for the Practical Examination Held on: 27/11/25

Internal Examiner External Examiner

Subject Code	**Subject Name (Lab oriented Theory Course)**	**Category**	**L**	**T**	**P**	**C**
CS23331	**Design and Analysis of Algorithms**	**PC**	**3**	**0**	**2**	**4**

Objectives: The student should be made to:	
ὃ	Learn and understand the algorithm analysis techniques and complexity notations
ὃ	Become familiar with the different algorithm design techniques for effective problem solving in computing.
ὃ	Learn to apply the design techniques in solving various kinds of problems in an efficient way.
ὃ	Understand the limitations of Algorithm power.
ὃ	Solve variety of problems using different design techniques

	List of Experiments		
1	Finding Time Complexity of algorithms		
2	Design and implement algorithms using Brute Force Technique		
3	Design and implement algorithms using Divide and Conquer Technique		
4	Design and implement algorithms using **G**reedy Technique		
5	Design and implement algorithms using Dynamic Programming		
6	Design and implement algorithms using Backtracking		
7	Design and implement algorithms using Branch and Bound		
8	Implement String Matching algorithms-Naïve String, Rabin Karp		
	Contact Hours	**:**	**30**

Course Outcomes: On completion of the course, the students will be able to	
ὃ	Analyze the time and space complexity of various algorithms and compare algorithms with respect to complexities.
ὃ	Ability to decide and Apply Brute Force and Divide and Conquer design strategies to Synthesize algorithms for appropriate computing problems.
ὃ	Ability to decide and Apply Greedy and Dynamic Programming techniques to Synthesize algorithms for appropriate computing problems.
ὃ	Ability to decide and Apply Backtracking and Branch and Bound techniques to Synthesize algorithms for appropriate computing problems.
ὃ	Apply string matching algorithms in vital applications

Web links for Lab-Resources for reference and practice	
1	https://www.geeksforgeeks.org/fundamentals-of-algorithms/
2	https://www.hackerrank.com/domains/algorithms

Web Portal for Practice-College LMS	
1	REC Digital Café-https://www.rajalakshmicolleges.net/moodle/

DEPARTMENT OF ARTIFICAIL INTELIGENCE AND DATA SCIENCE

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

FINDING COMPLEXITY OF ALGORITHMS

PROBLEM STATEMENTS

Problem Statement 1:

Given two positive integers, determine the GCD of the numbers.

Solve the above Problem Statement using two algorithms, hence write two functions,

1. Iterative Function 1(Consecutive Integer Checking): pass the 2 integers to the function, and print the GCD and return the no of times the loop gets executed in the function.
2. Iterative Function 1(Euclid's Algorithm): pass the 2 integers to the function, and print the GCD and return the no of times the loop gets executed in the function

Compare the return values and print which function is best for a specific problem instance.

Input Format

First Line Contains the Integer 1

Second line Contains Integer 2

Output Format

First line prints the result in function 1
Second line prints the result in function 2
Third line prints the return value of function 1
Fourth line prints the return value of function 2
Fifth line, Print "Function 1" if return value of function 1 is lesser than return value of function 2
Print: Function 2", if return value of function 2 is lesser than return value of function 1 otherwise print "Equal"

Sample Input

```
10
6
```

Sample Output

```
2
2
5
3
Function 2
```

PROGRAM:

```
#include <stdio.h>
int func1(int ,int);
int func2(int,int);
int main()
{
   int a,b;
   scanf("%d %d",&a,&b);
   int c=func1(a,b);
   int d=func2(a,b);
   printf("%d\n%d\n",c,d);
   if(c<d)
   {
       printf("Function 1");

   }
   else if(c>d)
   {
       printf("Function 2");
   }
   else
    printf("Equal");
   return 0;
}
int func1(int a,int b)
{
    int min;
    min=a<b?a:b;
    //printf("%d",min);
    int count=0;
    for(int i=min;i>=1;i--)
    {
       count++;
        if((a%i==0) && (b%i==0))
        {
               printf("%d\n",i);
               return count;
        }
    }
    return count;
}
int func2(int a,int b)
{
    int count=0;
  while(b!=0)
  {
      count++;
      int r;
      r=a%b;
      a=b;
      b=r;
  }
  printf("%d\n",a);
  return count;
}
```

Test Case 1:

Input

```
    10
    6
```

Output

```
  2
  2
  5
  3
  Function 2
```

Test Case 2:

Input

```
    1000
    24
```

Output

```
  8
  8
  17
  3
  Function 2
```

Test Case 3:

Input

```
    60
    24
```

Output

```
  12
  12
  13
  2
  Function 2
```

Problem Statement 2:

Convert the following algorithm into a program and find its time complexity using the counter method.

```
void function (int n)
{
   int i = 1, s =1;
   while (s <= n)
   {
               i++;
               s += i;
    }

  }
```

Note: No need of counter increment for declaration of 'n' and scanf() statement.

Input:

A positive Integer n

Output:

Print the value of the counter variable

PROGRAM:

```
#include <stdio.h>
int main()
{
   int n;
   int count=0;
   scanf("%d",&n);
   int i=1;
   count++;
   int s=1;
   count++;
   while(s<=n)
   {
      count++;
      i++;
      count++;
      s=s+i;
      count++;

   }
   count++;
   printf("%d",count);
   return 0;
}
```

Test case 1:
Input:
9
Output:
12

Test case 2:
Input 25
Output:
21

Test Case 3:
Input
4
Output
9

Problem Statement 3:

Convert the following algorithm into a program and find its time complexity using counter method.

```
void func(int n)
{

   if (n==1)
         {
                  printf("");
         }
   else
         {
                  for (int i=1; i<=n; i++)
                  {
                           for (int j=1; j<=n; j++)
                           {
                                    printf ("");
                                    printf("");
                                    break;
                           }
                  }
         }
 }
```

Note: No need of counter increment for declaration of 'n' and scanf() statement.
Input:
A positive Integer n
Output:
Print the value of the counter variable

PROGRAM:

```
#include <stdio.h>
int main()
{

  int n;
  int count=0;
  scanf("%d",&n);
   if (n==1)
         {
                  count++;
                  printf("");
                  count++;
```

```
          }
    else
          {
             count++;
                   for (int i=1; i<=n; i++)
                   {
                      count++;
                            for (int j=1; j<=n; j++)
                            {
                               count++;
                                     printf ("");
                                     count++;
                                     printf("");
                                     count++;
                                     break;
                            }

                   }
                   count++;
          }

printf("%d",count);
 }
```

Test case 1:
Input:
2
Output:
10

Test case 2:
Input
1000
Output:
4002

Test Case 3:
Input
143
Output
574

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

BRUTE FORCE AND DIVIDE AND CONQUER

PROBLEM STATEMENTS

Problem Statement 1:

Given an integer array `INTS`, return the number of range sums that lie in `[LOWER, UPPER]` inclusive. Range sum `S(i, j)` is defined as the sum of the elements in `INTS` between indices `i` and `j` (`i ≤ j`), inclusive.

(Note: Write a Brute force algorithm)

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains the LOWER and UPPER values of the range

Output Format

First Line Contains the number of Range sums.

Sample Input
3
-2
5
-1
-2 2

Sample Output

3

Explanation:

The three ranges are : [0,0], [2,2], [0,2] and their respective sums are: -2, -1, 2,these are within the range (-2,2)

Test Case 1:
Sample Input
3
-2
5
-1
-2 2

Sample Output

3

Test Case 2:
Sample Input
1
22

-2 2

Sample Output

0

Test Case 3:

Sample Input

```
5
22
1
-2
4
5
1 4
```

Sample Output

2

Problem Statement 2:

Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to find the *CEILING* of x. *CEILING* of a number is the smallest element in the array which is either greater than or equal to x. If there is no *CEILING* value i.e., all the elements in the array are smaller than x, then return -1, otherwise return the index of the *CEILING* value of number x.

(Note: Write a Divide and Conquer Solution)

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Element to find Ceiling Value

Output Format

First Line Contains Integer – Index Value of Ceiling Value of Integer x

Sample Input

```
7
3
5
7
9
11
13
15
10
```

Sample Output

4

PROGRAM:

```
#include<stdio.h>
int main()
{
   int n,ele,index;
```

```
    scanf("%d", &n);
    int arr[n];
    for(int i=0;i<n;i++)

    {
        scanf("%d",&arr[i]);
    }
    scanf("%d",&ele);
    index = ceilSearch(arr,0,n-1,ele);
    printf("%d",index);
}


int ceilSearch(int arr[], int low, int high, int x)
{
  int mid;

  /* If x is smaller than or equal to the first element,
    then return the first element */

  if(x <= arr[low])
    return low;

  /* If x is greater than the last element, then return -1 */
  if(x > arr[high])
    return -1;

  /* get the index of middle element of arr[low..high]*/
  mid = (low + high)/2;  /* low + (high - low)/2 */

  /* If x is same as middle element, then return mid */
  if(arr[mid] == x)
    return mid;

  /* If x is greater than arr[mid], then either arr[mid + 1]
    is ceiling of x or ceiling lies in arr[mid+1...high] */
  else if(arr[mid] < x)
  {
    if(mid + 1 <= high && x <= arr[mid+1])
      return mid + 1;

    else
      return ceilSearch(arr, mid+1, high, x);
  }

  /* If x is smaller than arr[mid], then either arr[mid]
     is ceiling of x or ceiling lies in arr[mid-1...high] */
  else
  {
    if(mid - 1 >= low && x >arr[mid-1])
      return mid;

    else
      return ceilSearch(arr, low, mid - 1, x);
  }
}
```

Test Case 1:

Input
```
5
2
4
6
8
10
7
```
Output
```
3
```

Test Case 2:

Input
```
8
3
6
9
12
15
18
21
24
5
```
Output
```
1
```

Test Case 3:

Input
```
4
20
30
60
70
79
```
Output
```
-1
```

Problem Statement 2:

Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return their indices, otherwise print as "No Two elements exist".

Note: Write a Divide and Conquer Solution)

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Sum Value

Output Format

First Line Contains Integer – Index of Element1

Second Line Contains Integer – Index of Element2 (Element 1 and Elements 2 together sums to value "x")

Sample Input

```
5
2
4
6
8
10
14
```

Sample Output

```
1
4
```

PROGRAM:

```
#include <stdio.h>
int main()
{

    int n,ele,index;
    scanf("%d", &n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    scanf("%d",&ele);
    for(int j=0;j<n;j++)
    {
        int a, b, index;
        a=arr[j];
        b = ele - arr[j];
        index = binarySearch(arr, 0, n-1, b);
        if((index!=j)&&(index!=-1))
        {
            printf("%d\n",j);
            printf("%d",index);
            return 0;
        }


    }
   printf("No two elements exist");
}

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}
```

Test Case 1:

Input

```
  5
  1
  1
  1
  1
  1
  10
```

Output

```
 No two Elements Exist
```

Test Case 2:

Input

```
3
10
20
30
50
```

Output

```
 1
 2
```

Test Case 3:

Input

```
  7
  1
  1
  2
  24
  36
  48
  51
  60
```

Output

```
 3
 4
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

GREEDY ALGORITHM

PROBLEM STATEMENTS

Problem Statement 1:

A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a distance to burn out his calories. If he has eaten ***i*** burgers with **c** calories each, then he has to run at least $3^i * c$ kilometers to burn out the calories. For example, if he ate 3 burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are $(3^0 * 1) + (3^1 * 3) + (3^2 * 2) = 1 + 9 + 18 = 28$. But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the minimum distance he needs to run. **Note:** He can eat burger in any order and use an efficient sorting algorithm

Input Format

First Line contains the number of burgers

Second line contains calories of each burger which is n space-separated integers

Output Format

Print: Minimum number of kilometers needed to run to burn out the calories

Sample Input

3
5 10 7

Sample Output

44

PROGRAM:

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

void quicksort (int *a, int n) {
   if (n < 2)
      return;
   int p = a[n / 2];
   int *l = a;
   int *r = a + n - 1;
   while (l <= r) {
      if (*l < p) {
         l++;
         continue;
      }
```

```
        if (*r > p) {
            r--;
            continue; // we need to check the condition (l <= r) every time we change the value of l or r
        }
        int t = *l;
        *l++ = *r;
        *r-- = t;
    }
    quicksort(a, r - a + 1);
    quicksort(l, a + n - l);
}


int main(){
    int n;
    scanf("%d",&n);
    int *a = malloc(sizeof(int) * n);
    for(int i = 0; i < n; i++)
   {
      scanf("%d",&a[i]);
    }
    int t;

    quicksort(a,n);
    long int sum=0;
    for(int i=0;i<n;i++)
        {
        sum=sum+a[i]*pow(3,i);
    }
    printf("%ld",sum);
    return 0;
}
```

Test Case 1:

Input
```
    3
    1 3 2
```

Output
```
  18
```

Test Case 2:

Input
```
    4
    7     4          9          6
```

Output
```
  192
```

Test Case 3:

Input
```
3
5 10 7
```

Output

Problem Statement 2:

In a gift shop there are **N** different types of gifts available and the prices for all **N** different types of gifts are given. It is a discount sale where you can buy a single gift and get at-most **M** other gifts for free.

1. Find the minimum amount of money that is needed to buy all the **N** different gifts.
2. Find the maximum amount of money that is needed to buy all the **N** different gifts.

In both these cases utilize the discount and get maximum possible gifts back. If **M** or more gifts are available, for every purchase take **M** gifts. In case less than **M** gifts are in stock, then take all gifts for purchasing a gift.

For example, if there are four gifts in shop with prices 3,2,1,4 respectively and M=2. Since M=2, if we purchase one gift we can take at most two more for free. So in the first case we purchase the gift whose price is 1 and take gifts worth 3 and 4 for free, also you can purchase gift worth 2 as well. Therefore, minimum cost = 1+2 = 3. In the second case we purchase the gift whose price is 4 and take gifts worth 1 and 2 for free, also you can buy gift worth 3 as well. Therefore, maximum cost = 3+4 = 7.

Input Format

First Line contains the number of gifts in the shop.

Second line contains the price of each gift.

The third line contains the value of M.

Output Format

Print: Minimum Cost
       Maximum Cost

Sample Input

```
4
2
3 2 1 4
```

Sample Output

```
3
7
```

PROGRAM:

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

int findMin(int *a, int n, int m)
{
   int res = 0;
   for (int i=0; i<n ; i++)
   {
      // Buy current gift
      res += a[i];

      // And take m gifts for free from the last
      n = n-m;
   }
   return res;
```

```
}



// Function to find the maximum amount to buy all gifts
int findMax(int *a, int n, int m)
{
   int res = 0, index = 0;

   for (int i=n-1; i>=index; i--)
   {
      // Buy gift with maximum amount
      res += a[i];

      // And get m candies for free from the starting
      index += m;
   }
   return res;
}
void quicksort (int *a, int n) {
   if (n < 2)
      return;
   int p = a[n / 2];
   int *l = a;
   int *r = a + n - 1;
   while (l <= r) {
      if (*l < p) {
         l++;
         continue;
      }
      if (*r > p) {
         r--;
         continue; // we need to check the condition (l <= r) every time we change the value of l or r
      }
      int t = *l;
      *l++ = *r;
      *r-- = t;
   }
   quicksort(a, r - a + 1);
   quicksort(l, a + n - l);
}

int main()
{
   int n, M, *a,i;

   scanf("%d", &n);
   a = (int*) malloc(n * sizeof(int));

         for(i = 0; i < n; i++)
         {
                  scanf("%d", &a[i]);
         }
         scanf("%d", &M);
         quicksort(a,n);

         printf("%d\n",findMin(a,n,M));
```

```
        printf("%d",findMax(a,n,M));
        return 0;
}
```

Test Case 1:

Input

```
4
10 30 40 20
2
```

Output

```
30
70
```

Test Case 2:

Input

```
5
3 5 2 4 1
3
```

Output

```
3
9
```

Test Case 3:

Input

```
7
3 6 7 3 9 1 4
```

Output

```
3
4
```

Problem Statement 3:
Given lengths of **n** pipes and a value **m**. You can either increase or decrease the length of every pipe by **m** (only once) where **m > 0**. The task is to minimize the difference between the lengths of the longest and the shortest pipe after modifications, and output this difference.

Input Format

First Line contains the number of pipes.

Second line contains the length of each pipe.

The third line contains the value of m.

Output Format

Print: Minimum difference

Sample Input

```
3
1 15 10
6
```

Sample Output

```
5
```

PROGRAM:

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

void quicksort (int *a, int n) {
   if (n < 2)
      return;
   int p = a[n / 2];
   int *l = a;
   int *r = a + n - 1;
   while (l <= r) {
      if (*l < p) {
         l++;
         continue;
      }
      if (*r > p) {
         r--;
         continue; // we need to check the condition (l <= r) every time we change the value of l or r
      }
      int t = *l;
      *l++ = *r;
      *r-- = t;
   }
   quicksort(a, r - a + 1);
   quicksort(l, a + n - l);
}

int getMinDiff(int *a, int n, int m)
{
   if (n == 1)
     return 0;

   // Sort all elements
   quicksort(a,n);

   // Initialize result
   int ans = a[n-1] - a[0];

   // Handle corner elements
   int small = a[0] + m;
   int big = a[n-1] - m;
   if (small > big)
   {
     int temp = small;
     small = big;
     big = temp;
   }
         // Traverse middle elements
   for (int i = 1; i < n-1; i ++)
   {
      int subtract = a[i] - m;
      int add = a[i] + m;
```

```
          // If both subtraction and addition do not change diff
        if (subtract >= small || add <= big)
            continue;

        // Either subtraction causes a smaller number or addition causes a greater number. Update small or big
// using greedy approach (If big - subtract causes smaller diff, update small Else update big)
        if (big - subtract <= add - small)
            small = subtract;
        else
            big = add;
    }
    if(ans<(big-small))
        return ans;
    else
        return  (big - small);
}

// Driver function to test the above function
int main()
{
    int n, m, *a,i;

    scanf("%d", &n);

            a = (int*) malloc(n * sizeof(int));

            for(i = 0; i < n; i++)
            {
                    scanf("%d", &a[i]);
            }
            scanf("%d", &m);

    printf("%d",getMinDiff(a, n, m));
    return 0;
}
```

Test Case 1:

Input
```
    4
    1 5 15 10
    3
```
Output
```
  8
```

Test Case 2:

Input
```
    6
    1 10 14 14 14 15
    6
```
Output
```
  5
```

Test Case 3:
Input
```
3
1 2 3
```

2

Output
2

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

DYNAMIC PROGRAMMING

PROBLEM STATEMENTS

Problem Statement 1:

Ram and Sita are playing with numbers by giving puzzles to each other. Now it was Ram term, so he gave Sita a positive integer 'n' and two numbers 1 and 3. He asked her to find the possible ways by which the number n can be represented using 1 and 3.Write any efficient algorithm to find the possible ways.

Example 1:

Input: 6

Output:6

Explanation: There are 6 ways to 6 represent number with 1 and 3

1+1+1+1+1+1

3+3

1+1+1+3

1+1+3+1

1+3+1+1

3+1+1+1

Input Format

First Line contains the number n

Output Format

Print: The number of possible ways 'n' can be represented using 1 and 3

Sample Input

6

Sample Output
6

PROGRAM:
```
#include <stdio.h>
int main()
```

```
{
    int n;
    printf("Enter the number:");
    scanf("%d",&n);
     long int result[n+1];
    result[0]=1;
    result[1]=1;
    result[2]=1;
    for(int i=3;i<=n;i++)
    {
        result[i]=result[i-1]+result[i-3];
    }
   printf("%ld",result[n]);
}
```

Test Case 1:

Input
6

Output
6

Test Case 2:

Input
25

Output
8641

Test Case 3:

Input
100

Output
24382819596721629

Problem Statement 2:

Ram is given with a n*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that the position of the top left white rook. He is been given a task to reach the bottom right black rook position (n-1, n-1) constrained that he needs to reach the position by travelling the maximum monetary path under the condition that he can only travel one step right or one step down the board. Help ram to achieve it by providing an efficient DP algorithm.

Example:

Input

3

1 2 4

2 3 4

8 7 1

Output:

19

Explanation:

Totally there will be 6 paths among that the optimal is

Optimal path value:1+2+8+7+1=19

Input Format

First Line contains the integer n

The next n lines contain the n*n chessboard values

Output Format

Print: Maximum monetary value of the path

PROGRAM:

```
#include <stdio.h>

int main()
{
   int n;
   printf("Enter n:");
   scanf("%d",&n);
   int a[n][n];
 //  int b[n][n]={0};;
   int i,j;
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   for(int j=1;j<n;j++)
     a[0][j]=a[0][j]+a[0][j-1];
   for(int i=1;i<n;i++)
     a[i][0]=a[i][0]+a[i-1][0];
   for(i=1;i<n;i++)
   {
      for(j=1;j<n;j++)
      {
         if(a[i][j-1]>=a[i-1][j])
            {
               a[i][j]+=a[i][j-1];
            }
         else
            {
              a[i][j]+=a[i-1][j];
```

```
                }
            }
        }


    printf("%d ",a[n-1][n-1]);
    return 0;
}
```

Test Case 1:

Input

```
  3
  1 3 1
  1 5 1
  4 2 1
```

Output:

```
  12
```

Test Case 2:

Input

```
3
1 2 4
2 3 4
8 7 1
```

Output

```
  19
```

Test Case 3:

Input

```
4
1 1 3 4
1 5 7 8
2 3 4 6
1 6 9 0
```

Output

```
28
```