RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23331

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

Laboratory Manual Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

Vision

To promote highly Ethical and Innovative Computer Professionals through excellence in teaching, training and research.

Mission

• To produce globally competent professionals, motivated to learn the emerging technologies and to be innovative in solving real world problems.

• To promote research activities amongst the students and the members of faculty that could benefit the society.

• To impart moral and ethical values in their profession.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO 1:To equip students with essential background in computer science, basic electronics and applied mathematics.

PEO 2:To prepare students with fundamental knowledge in programming languages, and tools and enable them to develop applications.

PEO 3:To develop professionally ethical individuals enhanced with analytical skills, communication skills and organizing ability to meet industry requirements.

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of Mathematics, Science, Engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

A graduate of the Computer Science and Design Program will have an

PSO 1: Ability to understand, analyze and develop efficient software solutions using suitable algorithms, data structures, and other computing techniques.

PSO 2: Ability to independently investigate a problem which can be solved by a Human Computer Interaction (HCI) design process and then design an end-to-end solution to it (i.e., from user need identification to UI design to technical coding and evaluation). Ability to effectively use suitable tools and platforms, as well as enhance them, to develop applications/products using for new media design in areas like animation, gaming, virtual reality, etc.

PSO 3: Ability to apply knowledge in various domains to identify research gaps and to provide solution to new ideas, inculcate passion towards higher studies, creating innovative career paths to be an entrepreneur and evolve as an ethically social responsible computer science and design professional.

CO – PO and PSO matrices of course

PO/PSO CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO 2	PSO 3
CS23331.1	3	-	-	-	-	-	-	-	-	-	-	1	3	2	2
CS23331.2	2	3	2	2	-	-	-	-	-	-	-	1	3	3	1
CS23331.3	2	3	2	2	-	-	-	-	-	-	-	1	3	3	1
CS23331.4	2	3	2	2	-	-	-	-	-	-	-	1	3	3	1
CS23331.5	1	2	2	2	-	-	-	-	-	-	-	1	3	3	1
Average	2	2.75	2	2	-	-	-	-	-	-	-	1	3	2.8	1.2

List of Experiments	
1	Finding Time Complexities of Algorithms
2	Implement Algorithms using Greedy Technique
3	Implement Algorithms using Divide and Conquer Technique
4	Implement Algorithms using Dynamic Programming
5	Implement Competitive Programming

	Contact Hours		**: 30**
	Total Contact Hours		**: 75**

Requirements

Hardware	Intel i3, CPU @ 1.20GHz 1.19 GHz, 4 GB RAM, 32 Bit Operating System
Software	REC Digital Café Portal

Safety Precautions

- Regular Backups: Ensure regular backups of all databases to prevent data loss.
- Secure Passwords: Use complex and unique passwords for database access and change them regularly.
- Antivirus Protection: Install and maintain updated antivirus software on all laboratory computers.
- Data Encryption: Encrypt sensitive data both in transit and at rest to protect against data breaches.
- Software Updates: Keep all database management software and operating systems up to date with the latest security patches.
- Environment Control: Ensure proper environmental controls, such as temperature and humidity, to protect hardware.
- Power Protection: Use Uninterruptible Power Supplies (UPS) to prevent data loss due to power outages.

Dos:

- Regular Maintenance: Perform regular maintenance and updates on the database systems to ensure optimal performance.
- Documentation: Maintain comprehensive documentation of database structures, procedures, and security policies.
- Monitoring: Continuously monitor database performance and security to detect and respond to issues promptly.
- Training: Provide regular training to staff and students on database management best practices and security measures.
- Data Integrity: Implement and enforce data integrity constraints to maintain accurate and reliable data.

Don'ts

- Sharing Passwords: Do not share passwords or leave them written down in accessible places.
- Ignoring Errors: Do not ignore system errors or warnings; investigate and resolve them promptly.
- Unauthorized Software: Do not install unauthorized software on lab computers as it may pose security risks.
- Neglecting Backups: Do not neglect regular backups; always have a backup strategy in place.
- Weak Passwords: Do not use weak or easily guessable passwords.
- Bypassing Security: Do not bypass or disable security features for convenience.
- Unverified Sources: Do not download or install software from unverified sources as they may contain malware.
- Public Wi-Fi: Avoid accessing the database from public Wi-Fi networks to prevent unauthorized interception of data.

INDEX

Reg. No. : _____ Name : _____

Year : _____ Branch : _____ Sec : _____

S. No.	Date	Title	Page No.	Teacher's Signature / Remarks
1		Finding Time Complexities of Algorithms-01		
2		Finding Time Complexities of Algorithms-02		
3		Finding Time Complexities of Algorithms-03		
4		Finding Time Complexities of Algorithms-04		
5		Finding Time Complexities of Algorithms-05		
6		Implement Algorithms using Greedy Technique-01		
7		Implement Algorithms using Greedy Technique-02		
8		Implement Algorithms using Greedy Technique-03		
9		Implement Algorithms using Greedy Technique-04		
10		Implement Algorithms using Greedy Technique-05		
11		Implement Algorithms using Divide and Conquer Technique-01		
12		Implement Algorithms using Divide and Conquer Technique-02		
13		Implement Algorithms using Divide and Conquer Technique-03		
14		Implement Algorithms using Divide and Conquer Technique-04		
15		Implement Algorithms using Divide and Conquer Technique-05		
16		Implement Algorithms using Dynamic Programming-01		
17		Implement Algorithms using Dynamic Programming-02		
18		Implement Algorithms using Dynamic Programming-03		
19		Implement Algorithms using Dynamic Programming-04		
20		Implement Competitive Programming-01		
21		Implement Competitive Programming-02		
22		Implement Competitive Programming-03		
23		Implement Competitive Programming-04		
24		Implement Competitive Programming-05		
25		Implement Competitive Programming-06		

Ex. No. : 1A **Date:**

Register No.: **Name:**

FINDING TIME COMPLEXITYOF ALGORITHMS

PROBLEM STATEMENT:

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

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

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

PROCEDURE:

- Introduce a Count variable to find the total number of executions that takes place in the given algorithm.
- Consider the following table to know the step count value for a type of statement

STATEMENT	STEP COUNT
Comments and Declarative	0 Steps
Assignment	1 Step
Conditional	1 Step
Loop Condition (for, while – true-n times and false-1 time)	(n + 1) steps
Body of Loop	n steps
Break, return	1 Step(need to increment the count variable before its occurrence)

- Print the Complexity of the algorithm.

Sample Input:

9

Sample Output:

12

PROGRAM:

```
#include<stdio.h>

int function(int n){
   int count = 0;
   int i = 1;count++;
   int s = 1;count++;
   while(s <= n){
      count++;
      i++;
      count++;
      s += i;
      count++;
   }
   count++;

   return count;
}
int main(){
   int n;
   scanf("%d", &n);
   printf("%d", function(n));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 1B **Date:**

Register No.: **Name:**

FINDING TIME COMPLEXITYOF ALGORITHMS

PROBLEM STATEMENT:

Convert the following algorithm into a program and find its time complexity using the 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 declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

PROCEDURE:

- Introduce a Count variable to find the total number of executions that takes place in the given algorithm.
- Consider the following table to know the step count value for a type of statement

STATEMENT	STEP COUNT
Comments and Declarative	0 Steps
Assignment	1 Step
Conditional	1 Step
Loop Condition (for, while – true-n times and false-1 time)	(n + 1) steps
Body of Loop	n steps
Break, return	1 Step(need to increment the count variable before its occurrence)

- Print the Complexity of the algorithm.

Sample Input:

2

Sample Output:

12

PROGRAM:

```
#include<stdio.h>


int func(int n){
    int c = 0;
    if (n == 1){
        c++;
        // printf("*");
    }else{
        c++;
        for (int i=1; i<=n; i++){
            c++;
            for (int j=1; j<=n; j++){
                c++;
                // printf("*");
                // printf("*");
                c++;
```

```
            break;
        }
        c++;
        c++;
    }
    c++;
  }

  return c;
}

int main(){
  int n;
  scanf("%d",&n);
  printf("%d", func(n));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found

Ex. No. : 1C Date:

Register No.: Name:

FINDING TIME COMPLEXITY OF ALGORITHMS

PROBLEM STATEMENT:

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

```
Factor(num) {
 {
   for (i = 1; i <= num;++i)
   {
    if (num % i== 0)
       {
         printf("%d ", i);
       }
    }
 }
```

Note: No need of counter increment for declarations and scanf() and counter variable printf() statement.

Input Format:

A positive Integer n

Output Format:

Print the value of the counter variable

PROCEDURE:

- Introduce a Count variable to find the total number of executions that takes place in the given algorithm.
- Consider the following table to know the step count value for a type of statement

- Print the Complexity of the algorithm.

STATEMENT	STEP COUNT
Comments and Declarative	0 Steps
Assignment	1 Step
Conditional	1 Step
Loop Condition (for, while – true-n times and false-1 time)	(n + 1) steps
Body of Loop	n steps
Break, return	1 Step(need to increment the count variable before its occurrence)

Sample Input:

12

Sample Output:

25

PROGRAM:

```
#include<stdio.h>


int Factor(int n){
   int c = 0;
   for (int i=1; i<=n; i++){
      c++;
      if (n % i == 0){
         c++;
         // printf("%d", i);
      }
      c++;
   }
   c++;


   return c;
}
```

```
int main(){
   int n;
   scanf("%d",&n);
   printf("%d", Factor(n));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 1D **Date:**

Register No.: **Name:**

FINDING TIME COMPLEXITYOF ALGORITHMS

PROBLEM STATEMENT:

Convert the following algorithm into a program and find its time

complexity using counter method.

```
void function(int n)
{
   int c= 0;
   for(int i=n/2; i<n; i++)
      for(int j=1; j<n; j = 2 * j)
         for(int k=1; k<n; k = k * 2)
            c++;
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input Format:

A positive Integer n

Output Format:

Print the value of the counter variable

PROCEDURE:

- Introduce a Count variable to find the total number of executions that takes place in the given algorithm.
- Consider the following table to know the step count value for a type of statement

- Print the Complexity of the algorithm.

STATEMENT	STEP COUNT
Comments and Declarative	0 Steps
Assignment	1 Step
Conditional	1 Step
Loop Condition (for, while – true-n times and false-1 time)	(n + 1) steps
Body of Loop	n steps
Break, return	1 Step(need to increment the count variable before its occurrence)

Sample Input:

4

Sample Output:

30

PROGRAM:

```
#include<stdio.h>


int func(int n){
   int count = 0;
   int c = 0;count++;


   for (int i=n/2; i<n; i++){
      count++;
      for (int j=1; j<n; j = 2 * j){
         count++;
         for (int k=1; k<n; k = k * 2){
            count++;
            c++;count++;
         }
         count++;
      }
```

```
        count++;
    }
    count++;

    return count;
}

int main(){
    int n;
    scanf("%d", &n);

    printf("%d", func(n));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 1E **Date:**

Register No.: **Name:**

FINDING TIME COMPLEXITYOF ALGORITHMS

PROBLEM STATEMENT:

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

```
void reverse(int n)
{
  int rev = 0, remainder;
  while (n != 0)
   {
      remainder = n % 10;
      rev = rev * 10 + remainder;
      n/= 10;

   }
print(rev);
}
```

Input Format:

A positive Integer n

Output Format:

Print the value of the counter variable

PROCEDURE:

- Introduce a Count variable to find the total number of executions that takes place in the given algorithm.
- Consider the following table to know the step count value for a type of statement

STATEMENT	STEP COUNT
Comments and Declarative	0 Steps
Assignment	1 Step
Conditional	1 Step
Loop Condition (for, while – true-n times and false-1 time)	(n + 1) steps
Body of Loop	n steps
Break, return	1 Step(need to increment the count variable before its occurrence)

- Print the Complexity of the algorithm.

Sample Input:

12

Sample Output:

10

PROGRAM:

```
#include<stdio.h>

int reverse(int n){
   int c = 0;
   int rev = 0, remainder;c++;
   while(n != 0){
      c++;
      remainder = n % 10;c++;
      rev = rev * 10 + remainder;c++;
      n /= 10;c++;
   }
   c++;

   // printf("%d", rev);
   c++;

   return c;
}

int main(){
   int n;
   scanf("%d", &n);
   printf("%d", reverse(n));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

IMPLEMENT ALGORITHMS USING GREEDY TECHNIQUE

PROCEDURE:

- Follow Greedy Technique abstraction to solve the problems
- **Greedy Technique Abstraction:**
- **Note: Refer Example problems solved in classroom.**

Greedy method control abstraction/ general method

```
Algorithm  Greedy(a, n)
// a[1:n] contains the n inputs
{
   solution=  //Initialize solution
   for i=1 to n do
   {
        x:=Select(a);
        if  Feasible(solution, x) then
                solution=Union(solution, x)
   }
return solution;
}
```

- **Note: Refer Example problems solved in classroom.**

Ex. No. : 2A **Date:**

Register No.: **Name:**

Greedy Technique

PROBLEM STATEMENT:

Write a program to take value V and we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change.

Input Format:

Take an integer from stdin.

Output Format:

print the integer which is change of the number.

Example Input :

64

Output:

4

Explanaton:

We need a 50 Rs note and a 10 Rs note and two 2 rupee coins.

PROGRAM:

```
#include <stdio.h>

int main(){
   int n;
   scanf("%d", &n);

   int supply[] = {1000, 500, 100, 50, 20, 10, 5, 2, 1};
   int len = 9, res = 0;

   int i = 0;
   while(i < len){
      while (supply[i] < n){
         n -= supply[i];
         res++;
      }
      i++;
   }

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

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 2B **Date:**

Register No.: **Name:**

Greedy Technique

PROBLEM STATEMENT:

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Input:

3

1 2 3

2

1 1

Output:

1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Constraints:

1 <= g.length <= 3 * 10^4

0 <= s.length <= 3 * 10^4

1 <= g[i], s[j] <= 2^31 - 1

PROGRAM:

```
#include <stdio.h>

int main(){
    int children;
    scanf("%d", &children);

    int g[children];
    for (int i=0; i<children; i++){
        scanf("%d", &g[i]);
    }

    int cookies;
    scanf("%d", &cookies);

    int s[cookies];
    for (int i=0; i<cookies; i++){
        scanf("%d", &s[i]);
    }

    int res = 0;
    for (int i=0; i<cookies; i++){
        for (int j=0; j<children; j++){
            if (g[j] != -1 && g[j] <= s[i]){
                res++;
                g[j] = -1;
            }
        }
    }

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

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 2C **Date:**

Register No.: **Name:**

Greedy Technique

PROBLEM STATEMENT:

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-separate integers

Output Format

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

Sample Input

3

5 10 7

Sample Output

76

PROGRAM:

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

int main(){
    int n;
    scanf("%d", &n);

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

    //sort
    for (int i=0; i<n-1; i++){
        for (int j=i+1; j<n; j++){
            if (nums[i] < nums[j]){
                int temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
            }
        }
    }

    int res = 0;
    for (int i=0; i<n; i++){
        res += (pow(n, i) * nums[i]);
    }

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

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 2D **Date:**

Register No.: **Name:**

Greedy Technique

PROBLEM STATEMENT:

Given an array of N integer, we have to maximize the sum of arr[i] * i, where i is the index of the element (i = 0, 1, 2, ..., N).Write an algorithm based on Greedy technique with a Complexity O(nlogn).

Input Format:

First line specifies the number of elements-n

The next n lines contain the array elements.

Output Format:

Maximum Array Sum to be printed.

Sample Input:

5

2 5 3 4 0

Sample output:

40

PROGRAM:

```
#include <stdio.h>

int main(){
    int n;
    scanf("%d", &n);

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

    for (int i=0; i<n-1; i++){
        for (int j=i+1; j<n; j++){
            if (arr[i] > arr[j]){
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }


    int res = 0;
    for (int i=0; i<n; i++){
        res += arr[i] * i;
    }
    printf("%d", res);
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 2E **Date:**

Register No.: **Name:**

Greedy Technique

PROBLEM STATEMENT:

Given two arrays array_One[] and array_Two[] of same size N. We need to first rearrange the arrays such that the sum of the product of pairs(1 element from each) is minimum. That is SUM (A[i] * B[i]) for all i is minimum.

For example:

Input	Result
3 1 2 3 4 5 6	28

PROGRAM:

```
#include <stdio.h>


int main(){
   int n;
   scanf("%d", &n);


   int arr1[n], arr2[n];
   for (int i=0; i<n; i++){
      scanf("%d", &arr1[i]);
   }



   for (int i=0; i<n; i++){
      scanf("%d", &arr2[i]);
```

```
    }

    for (int i=0; i<n-1; i++){
        for (int j=0; j<n-i-1; j++){
            if (arr1[j] > arr1[j+1]){
                int temp = arr1[j];
                arr1[j] = arr1[j+1];
                arr1[j+1] = temp;
            }
        }
    }

    for (int i=0; i<n-1; i++){
        for (int j=0; j<n-i-1; j++){
            if (arr2[j] < arr2[j+1]){
                int temp = arr2[j];
                arr2[j] = arr2[j+1];
                arr2[j+1] = temp;
            }
        }
    }

    int res = 0;
    for (int i=0; i<n; i++){
        res += arr1[i] * arr2[i];
    }

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

RESULT:

Hence the time complexity of the given algorithm has been found.

IMPLEMENT ALGORITHMS USING DIVIDE AND CONQUER TECHNIQUE

PROCEDURE:

- Follow Divide and Conquer abstraction to solve the problems
- **Divide and Conquer Abstraction:**

Control abstraction of D&C Method

```
1.  Algorithm DAndC(P)
2.  {
3.     if small(P) then return S(P);
4.     else
5.     {
6.           divide P into smaller instances  P1, P2… Pk, k>=1;
7.    Apply DAndC to each of these sub problems;
8.    return combine (DAndC(P1), DAndC(P2),..,DAndC(Pk));
9.      }
10.     }
```

Note: Refer Example problems solved in classroom.

Ex. No. : 3A **Date:**

Register No.: **Name:**

DIVIDE AND CONQUER

PROBLEM STATEMENT:

Given an array of 1s and 0s this has all 1s first followed by all 0s. Aim is to find the number of 0s. Write a program using Divide and Conquer to Count the number of zeroes in the given array.

Input Format

First Line Contains Integer m – Size of array

Next m lines Contains m numbers – Elements of an array

Output Format

First Line Contains Integer – Number of zeroes present in the given array.

PROGRAM:

```
#include <stdio.h>

int divide (int arr[], int l, int h){
   if (l == h){
      if (arr[l] == 0)    return 1;
      else    return 0;
   }

   int mid = (l+h)/2;

   return divide (arr, l, mid) + divide (arr, mid+1, h);
}

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

   printf("%d",divide(arr, 0, n-1));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 3B **Date:**

Register No.: **Name:**

DIVIDE AND CONQUER

PROBLEM STATEMENT:

Given an array nums of size n, return the majority element.The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]

Output: 3

Example 2:

Input: nums = [2,2,1,1,1,2,2]

Output: 2

PROGRAM:

```
#include <stdio.h>

int count (int arr[], int start, int end, int n){
   int c = 0;
   for (int i=start; i<=end; i++){
      if (arr[i] == n)    c++;
   }
   return c;
}

int majorityElement(int arr[], int start, int end){
   if (start == end)   return arr[start];

   int mid = (start + end)/2;
```

```
    int leftMajority = majorityElement(arr, start, mid);
    int rightMajority = majorityElement(arr, mid+1, end);

    if (leftMajority == rightMajority)  return leftMajority;

    int leftCount = count(arr, start, end, leftMajority);
    int rightCount = count(arr, start, end, rightMajority);

    int n = end - start + 1;
    if (leftCount > n/2)    return leftMajority;
    if (rightCount > n/2)   return rightMajority;

    return -1;
}

int main(){
    int n;
    scanf("%d", &n);

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

    printf("%d", majorityElement(arr, 0, n-1));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 3C **Date:**

Register No.: **Name:**

DIVIDE AND CONQUER

PROBLEM STATEMENT:

Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write divide and conquer algorithm to find floor of x.

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 – Value for x

Output Format

First Line Contains Integer – Floor value for x

PROGRAM:

```
#include <stdio.h>

int binSearch(int arr[], int start, int end, int x){
   if (start == end){
      if (arr[start] == x)    return x;
   }

   int mid = (start + end) / 2;

   if (arr[mid] == x){
      return x;
   }else if(arr[mid] > x){
      return binSearch(arr, start, mid-1, x);
   }else{
      if (arr[mid+1] > x) return arr[mid];
```

```
        return binSearch(arr, mid+1, end, x);
    }
}

int main(){
    int n;
    scanf("%d", &n);

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

    int x;
    scanf("%d", &x);

    printf("%d", binSearch(arr, 0, n-1, x));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 3D **Date:**

Register No.: **Name:**

DIVIDE AND CONQUER

PROBLEM STATEMENT:

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 the numbers, otherwise print as “No”.

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 – Element1

Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value “x”)

PROGRAM:

```
#include <stdio.h>

void rec(int arr[], int start, int end, int x){

  if (start >= end){
    printf("No\n");
    return;
  }

  int val = arr[start] + arr[end];

  if (val == x){
    printf("%d\n%d\n",arr[start], arr[end]);
  }else if (val > x){
```

```
        rec(arr, start, end-1, x);
    }else{
        rec(arr, start+1, end, x);
    }
}

int main(){
    int n;
    scanf("%d", &n);

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

    int x;
    scanf("%d", &x);

    rec(arr, 0, n-1, x);

    return 0;
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 3E **Date:**

Register No.: **Name:**

DIVIDE AND CONQUER

PROBLEM STATEMENT:

Write a Program to Implement the Quick Sort Algorithm

Input Format:
The first line contains the no of elements in the list-n
The next n lines contain the elements.

Output:
Sorted list of elements

For example:

Input	Result
5 67 34 12 98 78	12 34 67 78 98

PROGRAM:

```
#include <stdio.h>


void quickSort(int arr[], int low, int high){
   if (low < high){
      int pivot = low;
      int i = low + 1;
      int j = high;


      while(i < j){
         while (i <= high && arr[i] < arr[pivot]){
            i++;
```

```
        }

        while (j >= low && arr[j] > arr[pivot]){
            j--;
        }

        if (i < j){
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    if (arr[pivot] > arr[j]){
        int temp = arr[pivot];
        arr[pivot] = arr[j];
        arr[j] = temp;
    }

    quickSort(arr, low, j-1);
    quickSort(arr, j+1, high);
  }
}

int main(){
  int n;
  scanf("%d", &n);

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

```
    quickSort(arr, 0, n-1);

    for (int i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

IMPLEMENT ALGORITHMS USING DYNAMIC PROGRAMMING

PROCEDURE

Steps to solve a problem using dynamic programming technique

- ✔ Identify if it is a DP problem
- ✔ Formulate state relationship -Recursive function-formula
- ✔ Do tabulation (or add memoization)-Storing intermediate results
- ✔ Bottom-up computation

Note: Refer Example problems solved in classroom.

Ex. No. : 4A **Date:**

Register No.: **Name:**

DYNAMIC PROGRAMMING

PROBLEM STATEMENT:

the possible ways.

Example:

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:

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;
    scanf("%d", &n);

    long int dp[n+1];
    dp[0] = 0;
    dp[1] = 1;
    dp[2] = 1;
    dp[3] = 2;

    for (long int i=4; i<=n; i++){
        dp[i] = dp[i-1] + dp[i-3];
    }

    printf("%ld", dp[n]);
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 4B **Date:**

Register No.: **Name:**

DYNAMIC PROGRAMMING

PROBLEM STATEMENT:

Ram is given with an n*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that is the position of the top left white rook. He has 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 traveling 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:

3

1 2 4

2 3 4

8 7 1

Solution :19

Explanation:

Totally there will be 6 paths, among that the optimal path value is :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

Sample Input:

3

1 2 4

2 3 4

8 7 1

Sample Output:

19

PROGRAM:

```
#include <stdio.h>


int main(){
    int n;
    scanf("%d", &n);

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

    int max = 0;
    int dp[n+1][n+1];

    for (int i=0; i<n+1; i++){
        for (int j=0; j<n+1; j++){
            dp[i][j] = 0;
        }
    }

    for (int i=n-1; i>=0; i--){
        for (int j=n-1; j>=0; j--){
            if (dp[i+1][j] > dp[i][j+1]){
                max = dp[i+1][j];
            }else{
                max = dp[i][j+1];
            }
            dp[i][j] = arr[i][j] + max;
        }
```

```
    }

    printf("%d", dp[0][0]);
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 4C **Date:**

Register No.: **Name:**

DYNAMIC PROGRAMMING

PROBLEM STATEMENT:

Given two strings, find the length of the common longest subsequence(need not be contiguous) between the two.

Example:

s1: ggtabe

s2: tgatasb

s1	a	g	**g**	**t**	**a**	**b**	
s2	**g**	x	**t**	x	**a**	y	**b**

The length is 4

Solve it using Dynamic Programming

Input Format:

First line contains the first String input and Second line contains the next string input.

Output Format:

Length(in int) of longest common subsequence of two strings.

Sample Input:

aab

azb

Sample Output:

2

PROGRAM:

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

int LCS(char s1[], char s2[]){
    int m = strlen(s1);
    int n = strlen(s2);

    int dp[m][n];
    for (int i=0; i<=m; i++){
        for (int j=0; j<=n; j++){
            if (i == 0 || j == 0){
                dp[i][j] = 0;
            }else if (s1[i-1] == s2[j-1]){
                dp[i][j] = dp[i-1][j-1] + 1;
            }else{
                if (dp[i-1][j] > dp[i][j-1]){
                    dp[i][j] = dp[i-1][j];
                }else{
                    dp[i][j] = dp[i][j-1];
                }
            }
        }
    }

    return dp[m][n];
}

int main(){
    char s1[50];
    scanf("%s", s1);
```

```
    char s2[50];
    scanf("%s", s2);

    printf("%d", LCS(s1, s2));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 4D **Date:**

Register No.: **Name:**

DYNAMIC PROGRAMMING

PROBLEM STATEMENT:

Find the length of the Longest Non-decreasing Subsequence in a given Sequence.

Example:

Sequence:[-1,3,4,5,2,2,2,2,3] the Subsequence is [-1,2,2,2,2,3]

Input Format:
First line contains the input sequence.
Output Format:
Print the length of the longest non-decreasing Subsequence in sequence.
Sample Input:
9
-1 3 4 5 2 2 2 2 3
Sample Output:
6

PROGRAM:

```
#include <stdio.h>


int LNDS(int arr[], int n){
   int dp[n];
   int max = 1;


   for (int i=0; i<n; i++){
      dp[i] = 1;
   }


   for (int i=1; i<n; i++){
      for (int j=0; j<i; j++){
         if (arr[j] <= arr[i] && dp[i] < dp[j] + 1){
            dp[i] = dp[j] + 1;
```

```
            }
        }
        if (dp[i] > max){
            max = dp[i];
        }
    }

    return max;
}

int main(){
    int n;
    scanf("%d", &n);

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

    printf("%d", LNDS(arr, n));
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 5A **Date:**

Register No.: **Name:**

Competitive Programming

Finding Duplicates-O(n^2) Time Complexity,O(1) Space Complexity

PROBLEM STATEMENT:

Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

n Lines - n Elements

Output Format:
Element x - That is repeated

For example:

Input	Result
5 1 1 2 3 4	1

PROGRAM:

```
#include <stdio.h>


int main(){
   int n;
   scanf("%d", &n);


   int arr[n];
   int dp[n+1];
   for (int i=1; i<n+1; i++){
      dp[i] = 0;
   }
   for (int i=0; i<n; i++){
```

```
        scanf("%d", &arr[i]);
        if (dp[arr[i]] == 1){
            printf("%d", arr[i]);
            break;
        }
        dp[arr[i]] ++;
    }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 5B **Date:**

Register No.: **Name:**

Competitive Programming

Finding Duplicates-O(n) Time Complexity,O(1) Space Complexity

PROBLEM STATEMENT:

Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

n Lines - n Elements

Output Format:
Element x - That is repeated

For example:

Input	Result
5 1 1 2 3 4	1

PROGRAM:

```
#include <stdio.h>


int main(){
   int n;
   scanf("%d", &n);


   int arr[n];
   for (int i=0; i<n; i++){
      arr[i] = 0;
   }
   int num;
   for (int i=0; i<n; i++){
```

```
        scanf("%d", &num);
        if (arr[num-1] == 1){
            printf("%d", num);
            break;
        }
        arr[num-1]++;
    }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 5C **Date:**

Register No.: **Name:**

Competitive Programming

Print Intersection of 2 sorted arrays-O(m*n)Time Complexity,O(1) Space Complexity

PROBLEM STATEMENT:

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

· The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

1

6 1 2 3 4 5 6

2 1 6

Output:

1 6

For example:

Input	Result
1 3 10 17 57 6 2 7 10 15 57 246	10 57

PROGRAM:

```
#include <stdio.h>

int main(){
    int t;
    scanf("%d", &t);

    while(t > 0){
        int n1;
        scanf("%d", &n1);

        int arr1[n1];
        for (int i=0; i<n1; i++){
            scanf("%d", &arr1[i]);
        }

        int n2;
        scanf("%d", &n2);

        int arr2[n2];
        for (int i=0; i<n2; i++){
            scanf("%d", &arr2[i]);
        }

        int p1 = 0, p2 = 0;
```

```
        while (p1 < n1 && p2 < n2){
            int num1 = arr1[p1];
            int num2 = arr2[p2];

            if (num1 == num2){
                printf("%d ", num1);
                p1++;
                p2++;
            }else if (num1 > num2){
                p2++;
            }else{
                p1++;
            }
        }

        t--;
    }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 5D **Date:**

Register No.: **Name:**

Competitive Programming

Print Intersection of 2 sorted arrays-O(m+n)Time Complexity,O(1) Space Complexity

PROBLEM STATEMENT:

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

· The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

1

6 1 2 3 4 5 6

2 1 6

Output:

1 6

For example:

Input	Result
1 3 10 17 57 6 2 7 10 15 57 246	10 57

PROGRAM:

```
#include <stdio.h>


int main(){
    int t;
    scanf("%d", &t);


    while (t > 0){
        int n1;
        scanf("%d", &n1);
        int arr1[n1];
        for (int i=0; i<n1; i++){
            scanf("%d", &arr1[i]);
        }
        int n2;
        scanf("%d", &n2);
        int arr2[n2];
        for (int i=0; i<n2; i++){
            scanf("%d", &arr2[i]);
        }
        int p1 = 0, p2 = 0;
        while (p1 < n1 && p2 < n2){
            int num1 = arr1[p1];
            int num2 = arr2[p2];
            if (num1 == num2){
```

```
            printf("%d ", num1);
            p1++;p2++;
        }else if (num1 > num2){
            p2++;
        }else{
            p1++;
        }
    }
    t--;
  }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 5E **Date:**

Register No.: **Name:**

Competitive Programming

Pair with Difference-O(n^2)Time Complexity,O(1) Space Complexity

PROBLEM STATEMENT:

Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:
1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

YES as 5 - 1 = 4

So Return 1.

For example:

Input	Result
3 1 3 5 4	1

PROGRAM:

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

int main(){
    int n;
    scanf("%d", &n);

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

    int k;
    scanf("%d", &k);

    bool flag = true;
    for (int i=1; i<n && flag; i++){
        for (int j=0; j<i; j++){
            if (arr[i] - arr[j] == k){
                printf("1");
                flag = false;
                break;
            }
        }
    }

    if (flag){
        printf("0");
    }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.

Ex. No. : 5F **Date:**

Register No.: **Name:**

Competitive Programming

Pair with Difference -O(n) Time Complexity,O(1) Space Complexity

PROBLEM STATEMENT:

Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:
1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

YES as 5 - 1 = 4

So Return 1.

For example:

Input	Result
3 1 3 5 4	1

PROGRAM:

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

int main(){
    int n;
    scanf("%d", &n);

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

    int k;
    scanf("%d", &k);

    int l = 0, r = 1;
    bool flag = true;
    while (r < n){
        int diff = arr[r] - arr[l];

        if (diff == k){
            printf("1");
            flag = false;
            break;
        }else if (diff < k){
            r++;
        }else{
            l++;
            if (l == r) r++;
        }
    }
```

```
    if (flag){
        printf("0");
    }
}
```

RESULT:

Hence the time complexity of the given algorithm has been found.