**RAJALAKSHMIENGINEERINGCOLLEGE**

**RAJALAKSHMI NAGAR, THANDALAM – 602 105**

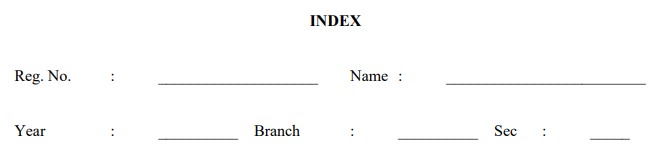


|  |
| --- |
| **CS23331**  **DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY** |
| **Laboratory Manual Note Book**  ***KAIF REHMAN S.T -230701134-CSE-C*** |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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** | |  |  | |
| H  a  r  d  w  a  r e | Intel i3, CPU @ 1.20GHz 1.19 GHz, 4 GB RAM,  32 Bit Operating System |
| S o  f t  w  a  r e | 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.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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.: 230701134** | |  | **Name:KAIF REHMAN S.T** |



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

**{ count++; i++; count++; s+=i; count++;**

**} count++; } int main()**

**{ int n; scanf("%d",&n); function(n); count++; printf("%d",count);**

**}**

**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 count=0; void func(int n){ count++; if(n==1)**

**{ count++; count++; } else**

**{ count++; for(int i=1;i<=n;i++)**

**{ count++; for(int j=1;j<=n;j++)**

**{ count++; count++; count++; break;**

**} count++;**

**}**

**} count--; }**

**int main(){ int n; scanf("%d",&n); func(n); count++; printf("%d",count);**

**}**

**RESULT:**

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

|  |  |  |
| --- | --- | --- |
| **Ex. No. :** | **1C** | **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.

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> void factor(int num); int main()**

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

**factor(n); return 0; } void factor(int num)**

**{ int count=0; int i; for(i=1;i<=num;++i)**

**{ count++; count++; if(num%i==0)**

**{ count++;**

**}**

**}**

**count++; printf("%d",count);**

**}**

**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 count=0; void function(int n)**

**{ count++; int c=0; count++; for(int i=n/2;i<n;i++)**

**{ count++; for(int j=1;j<n;j=j\*2)**

**{ count++; for(int k=1;k<n;k=k\*2)**

**{ count++; c++; count++;**

**} count++;**

**} count++;**

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

**} int main()**

**{ int n; scanf("%d",&n); function(n); return 0;**

**}**

**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 count=0; int rev=0,remainder; count++; count++; while(n!=0)**

**{ count++; remainder=n%10; count++; rev=rev\*10+remainder; count++; n/=10; count++;**

**} count++; return count; } int main()**

**{ int n,c; scanf("%d",&n); c=reverse(n); printf("%d",c);**

**}**

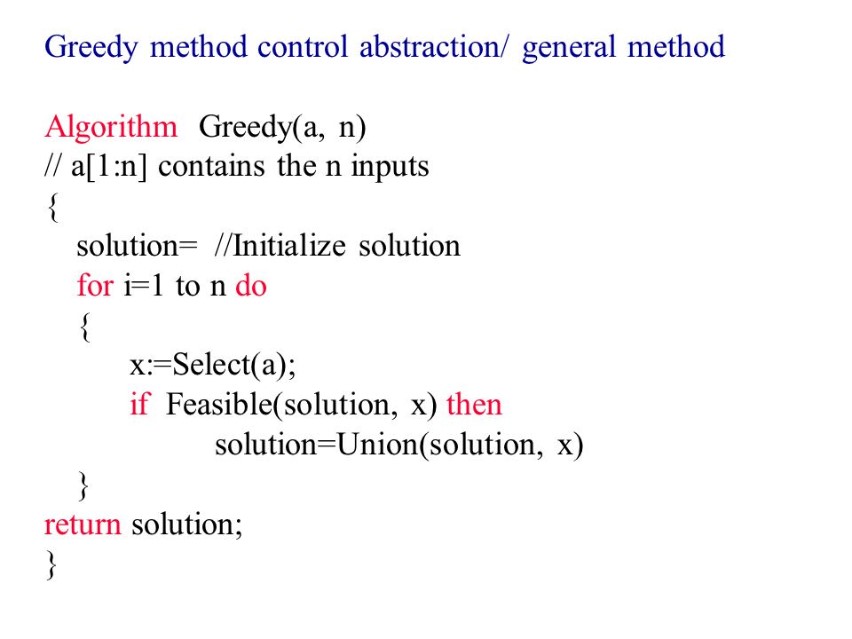
**RESULT:**

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.**



* **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.

**#include<stdio.h> int main()**

**{ int n,count=0; scanf("%d",&n); int a[9]={1,2,5,10,20,50,100,500,1000}; for(int i=8;i>=0;i--)**

**{ if(n>a[i])**

**{ n=n-a[i]; count++;**

**}**

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

**}**

**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

1. <= s.length <= 3 \* 10^4
2. <= g[i], s[j] <= 2^31 - 1

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

**int main()**

**{ int n,m; scanf("%d",&n); int a[n],c=0;**

**for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]);**

**}**

**scanf("%d",&m); int b[m]; for(int i=0;i<m;i++)**

**{ scanf("%d",&b[i]);**

**} for(int i=0;i<n;i++)**

**{ if(a[i]>=b[i])**

**{ c++;**

**} } printf("%d",c);**

**}**

**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 *3i \* 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 (30 \* 1) + (31 \* 3) + (32 \* 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

**#include<stdio.h> #include<math.h> int main()**

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

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

**{ for(int j=i+1;j<c;j++)**

**{ if(a[i]<a[j])**

**{ int t=a[i]; a[i]=a[j]; a[j]=t;**

**}**

**} } for(int i=0;i<c;i++)**

**{ int b=pow(c,i); s+=b\*a[i]; } printf("%d",s);**

**}**

**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> #include<math.h> int main()**

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

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

**{ for(int j=i+1;j<a;j++)**

**{ if(b[i]>b[j])**

**{ int t=b[i]; b[i]=b[j]; b[j]=t;**

**}**

**} } for(int i=0;i<a;i++)**

**{ int u=(b[i]\*i); s+=u; } printf("%d",s);**

**}**

**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> #include<math.h> int main()**

**{ int n; scanf("%d",&n); int a[n],b[n],s=0; for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]);**

**} for(int i=0;i<n;i++)**

**{ scanf("%d",&b[i]);**

**} for(int i=0;i<n;i++) { for(int j=i+1;j<n;j++)**

**{ if(a[i]>a[j])**

**{ int t=a[i]; a[i]=a[j]; a[j]=t;**

**}**

**} } for(int i=0;i<n;i++)**

**{ for(int j=i+1;j<n;j++)**

**{ if(b[i]<b[j])**

**{ int t=b[i]; b[i]=b[j]; b[j]=t;**

**}**

**} } for(int i=0;i<n;i++)**

**{ s+=a[i]\*b[i]; } printf("%d",s); }**

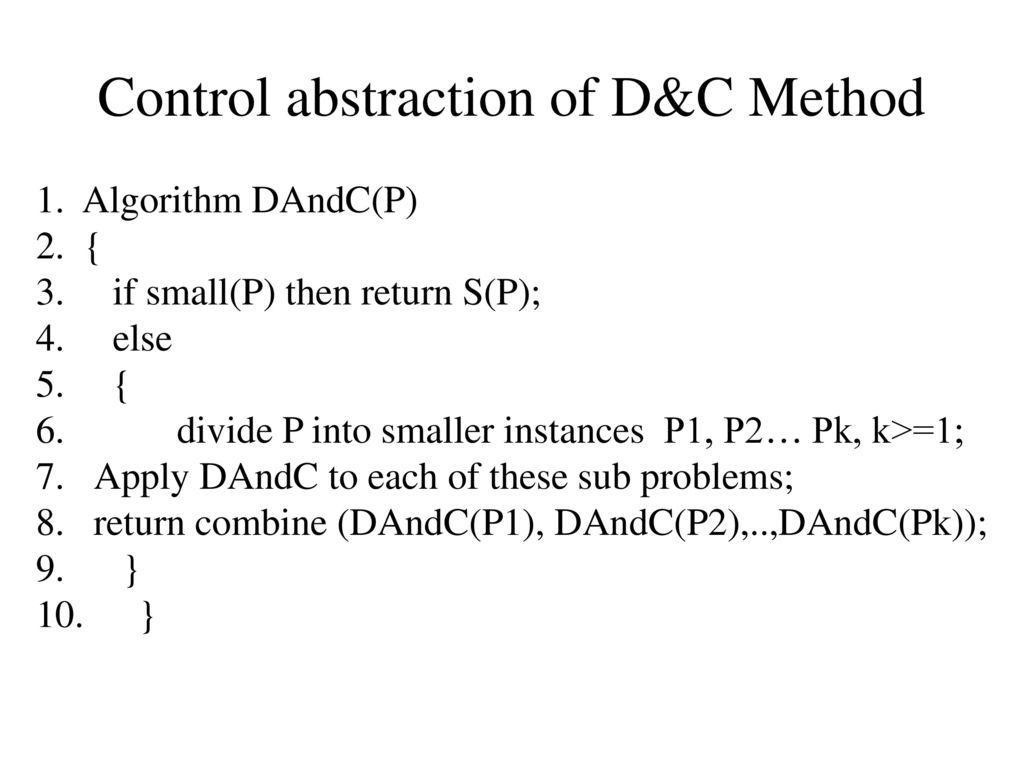
**RESULT:**

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:**



**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 dac(int l,int u); int a[100]; int c=0; int main()**

**{**

**int m; scanf("%d",&m); for(int i=0;i<m;i++)**

**{ scanf("%d",&a[i]);**

**}**

**int u=m-1; int l=0; int k=(dac(l,u)); printf("%d",k); } int dac(int l,int u)**

**{ if(l==u)**

**{ if(a[l]==0) c++; } else**

**{ int mid=(l+u)/2; dac(l,mid); dac(mid+1,u); } return c;**

**}**

**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 a[100]; int maj(int l,int h); int main()**

**{ int n; scanf("%d",&n); for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]);**

**} int l=0; int h=n-1; printf("%d",maj(l,h));**

**}**

**int maj(int l,int h)**

**{ if(l==h)**

**{ return a[l]; } else**

**{ int m=(l+h)/2; int left=maj(l,m); int right=maj(m+1,h); int c1=0,c2=0; for(int i=l;i<h;i++)**

**{ if(a[i]==l)**

**{ c1++; } else c2++; } if(c1>c2)**

**{ return left; } else**

**{ return right;**

**}**

**}**

**}**

**RESULT:**

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

## 3C

**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 a[100]; void find(int l,int h,int k)**

**{ if(l>h)**

**{ printf("%d",a[h]); return; } int m=(l+h)/2; if(a[m]==k)**

**{ printf("%d",a[m]); return;**

**}**

**else if(a[m]<k)**

**{ find(m+1,h,k);**

**} else**

**{ find(l,m-1,k);**

**}**

**}**

**int main()**

**{ int n; scanf("%d",&n); for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]);**

**} int k; scanf("%d",&k); int l=0,h=n-1; find(l,h,k);**

**}**

**RESULT:**

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

## 3D

**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> int a[100]; int main()**

**{ int n; scanf("%d",&n); for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]);**

**} int x; scanf("%d",&x); int flag=0; for(int i=0;i<n;i++) { for(int j=0;j<n;j++)**

**{ if(a[i]+a[j]==x && i!=j)**

**{ printf("%d\n%d",a[i],a[j]); i=n+1; j=n+1; flag=1;**

**}**

**} } if(flag==0)**

**{ printf("No");**

**}**

**}**

**RESULT:**

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

### 3E

**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> #include<stdlib.h> int a[100]; void swap(int \*a,int \*b)**

**{ int t=\*a; \*a=\*b;**

**\*b=t; } int part(int l,int h)**

**{**

**int p=a[h],i=l-1; for(int j=l;j<h;j++)**

**{ if(a[j]<p)**

**{ swap(&a[++i],&a[j]);**

**}**

**} swap(&a[i+1],&a[h]); return i+1;**

**} void sort(int l,int h)**

**{ if(l<h)**

**{ int pa=part(l,h); sort(l,pa-1); sort(pa+1,h);**

**}**

**} int main()**

**{ int n; scanf("%d",&n); for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]); } int l=0,h=n-1; sort(l,h); for(int i=0;i<n;i++)**

**{ printf("%d ",a[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> long int comb(int n)**

**{ long int dp[n+1]; dp[0]=0; dp[1]=1; dp[2]=1; dp[3]=2; for(int i=4;i<=n;i++)**

**{ dp[i]=dp[i-1]+dp[i-3]; } return dp[n]; } int main()**

**{ int n; scanf("%d",&n); printf("%ld",comb(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 a[n][n]; for(int i=0;i<n;i++)**

**{ for(int j=0;j<n;j++)**

**{ scanf("%d",&a[i][j]);**

**} } int dp[n][n]; dp[0][0]=a[0][0]; for(int i=0;i<n;i++)**

**{ for(int j=0;j<n;j++)**

**{ dp[i][j]=0;**

**} } for(int i=0;i<n;i++)**

**{ for(int j=0;j<n;j++)**

**{ if(i!=0||j!=0)**

**{ if(j>0)**

**{ dp[i][j]=(dp[i][j]>dp[i][j-1])?dp[i][j]:dp[i][j-1];**

**} if(i>0)**

**{ dp[i][j]=(dp[i][j]>dp[i-1][j])?dp[i][j]:dp[i-1][j];**

**} dp[i][j]+=a[i][j];**

**}**

**} } printf("%d",(dp[n-1][n-1])+1);**

**}**

**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 cls(char s1[],char s2[])**

**{ int n=strlen(s1),m=strlen(s2); int dp[n+1][m+1]; for(int i=0;i<n;i++)**

**{ for(int j=0;j<n;j++)**

**{ dp[i][j]=0;**

**} } for(int i=1;i<=n;i++)**

**{ for(int j=1;j<=m;j++)**

**{ if(s1[i-1]==s2[j-1])**

**{ dp[i][j]=dp[i-1][j-1]+1;**

**} else**

**{ dp[i][j]=(dp[i-1][j]>dp[i][j-1])?dp[i-1][j]:dp[i][j-1];**

**}**

**}**

**} return dp[n][m];**

**} int main()**

**{**

**char s1[5],s2[5]; scanf("%s %s",s1,s2); printf("%d",cls(s1,s2));**

**}**

**RESULT:**

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

**4D**

# 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 a[],int n)**

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

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

**{ if(a[i]>=a[j]&&dp[i]<dp[j]+1)**

**{ dp[i]=dp[j]+1;**

**}**

**} max=(max<dp[i]?dp[i]:max);**

**}**

**return max;**

**} int main()**

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

**{ scanf("%d",&a[i]);**

**} printf("%d",lnds(a,n));**

**}**

**RESULT:**

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

## 5A

### 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> void rep(int a[],int n)**

**{ int cp[n-1]; for(int i=0;i<n;i++)**

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

**{ cp[a[i]]++; if(cp[a[i]]>1)**

**{**

**printf("%d",a[i]); break;**

**}**

**}**

**} int main()**

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

**{ scanf("%d",&a[i]);**

**}rep(a,n);**

**}**

**RESULT:**

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

## 5B

### 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> void rep(int a[],int n)**

**{ int cp[n+1]; for(int i=0;i<n;i++)**

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

**{ cp[a[i]]++; if(cp[a[i]]>1)**

**{ printf("%d",a[i]); break;**

**}**

**}**

**} int main()**

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

**{ scanf("%d",&a[i]);**

**}rep(a,n);**

**}**

**RESULT:**

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

## 5C

### 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 n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++)**

**{ scanf("%d",&a[i]);**

**}**

**int m; scanf("%d",&m); int b[m]; for(int i=0;i<m;i++)**

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

**{ for(int j=0;j<m;j++)**

**{ if(a[i]==b[j])**

**{ printf("%d ",a[i]); break;**

**}**

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

**{ scanf("%d",&a[i]);**

**}**

**int m; scanf("%d",&m); int b[m]; for(int i=0;i<m;i++)**

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

**{ for(int j=0;j<m;j++)**

**{ if(a[i]==b[j])**

**{ printf("%d ",a[i]); break;**

**}**

**} } 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> int main()**

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

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

**{ for(int j=0;j<n;j++) { if((a[i]-a[j]==k)&&i!=j)**

**{ flag=1;**

**}**

**} } if(flag==1) printf("1"); else 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> int main()**

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

**{ scanf("%d",&a[i]);**

**} int k; int flag=0; scanf("%d",&k); for(int i=0;i<n;i++)**

**{ for(int j=0;j<n;j++)**

**{ if((a[i]-a[j]==k)&&i!=j)**

**{ flag=1;**

**}**

**} } if(flag==1) printf("1"); else printf("0");**

**}**

**RESULT:**

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