**NAME : PAVITHRA N**

**DEPT : CSE – D**

**ROLL NO : 230701225**

DAA Lab Record

**Week 1**

1. Finding Complexity using Counter Method

#include<stdio.h>

int main()

{

int count=0;

int n;

scanf("%d",&n);

count++;

int i=1;

count++;

int s=1;

count++;

while(s<=n){

count++;

i++;

count++;

s+=i;

count++;

}

printf("%d",count);

}

1. Finding Complexity using Counter Method

#include<stdio.h>

int main(){

int count=0;

count++;

int n;

count ++;

scanf("%d",&n);

if(n==1){

count++;

}

else{

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

count++;

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

count++;

count++;

count++;

break;

}

count++;

}

}

printf("%d",count);

}

1. Finding Complexity using Counter Method

#include<stdio.h>

int main(){

int count=0;

int n;

scanf("%d",&n);

count++;

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

count++;

if(n%i==0){

count++;

}

count++;

}

printf("%d",count);

}

1. Finding Complexity using Counter Method

#include<stdio.h>

int main(){

int count=0;

count++;

int n;

scanf("%d",&n);

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++;

count++;

}

count++;

}

count++;

}

printf("%d",count);

}

1. Finding Complexity using Counter Method

#include<stdio.h>

int main(){

int count=0;

int n;

scanf("%d",&n);

count++;

int rev=0;

int rem;

count++;

while(n!=0){

count++;

rem=n%10;

count++;

rev=rev\*10 +rem;

count++;

n/=10;

count++;

}

count++;

printf("%d",count);

}

Week 2

1. G-Coin Problem

#include <stdio.h>

int minCoins(int V) {

int denoms[] = {1, 2, 5, 10, 20, 50, 100, 500, 1000};

int n = sizeof(denoms) / sizeof(denoms[0]);

int count = 0;

for (int i = n - 1; i >= 0; i--) {

int currCount = V / denoms[i];

count += currCount;

V %= denoms[i];

}

return count;

}

int main() {

int V;

scanf("%d", &V);

int minCount = minCoins(V);

printf("%d\n", minCount);

return 0;

}

1. G-Cookies Problem

#include <stdio.h>

#include <stdlib.h>

int compare(const void\* a, const void\* b) {

return (\*(int\*)a - \*(int\*)b);

}

int findContentChildren(int\* g, int gSize, int\* s, int sSize) {

qsort(g, gSize, sizeof(int), compare);

qsort(s, sSize, sizeof(int), compare);

int i = 0, j = 0;

int satisfied = 0;

while (i < gSize && j < sSize) {

if (s[j] >= g[i]) {

satisfied++;

i++;

}

j++;

}

return satisfied;

}

int main() {

int g[] = {1, 2, 3};

int s[] = {1, 2};

int gSize = sizeof(g) / sizeof(g[0]);

int sSize = sizeof(s) / sizeof(s[0]);

int result = findContentChildren(g, gSize, s, sSize);

printf("%d\n", result);

return 0;

}

1. G-Burger Problem

#include <stdio.h>

#include<math.h>

int minDistance(int calories[], int n) {

for (int i = 0; i < n - 1; i++) {

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

if (calories[j] < calories[j + 1]) {

int temp = calories[j];

calories[j] = calories[j + 1];

calories[j + 1] = temp;

}

}

}

int distance = 0;

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

distance += pow(n,i)\* calories[i];

}

return distance;

}

int main() {

int n;

scanf("%d", &n);

int calories[n];

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

scanf("%d", &calories[i]);

}

int min = minDistance(calories, n);

printf("%d\n", min);

return 0;

}

1. G-Array Sum max problem

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pivot = partition(arr, low, high);

quickSort(arr, low, pivot - 1);

quickSort(arr, pivot + 1, high);

}

}

int maximizeSum(int arr[], int n) {

quickSort(arr, 0, n - 1);

int sum = 0;

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

sum += arr[i] \* i;

}

return sum;

}

int main() {

int n;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

int maxSum = maximizeSum(arr, n);

printf("%d\n", maxSum);

return 0;

}

1. G-Product of Array elements -Minimum

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void sortArrayOne(int array\_One[], int n) {

for (int i = 0; i < n - 1; i++) {

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

if (array\_One[j] > array\_One[j + 1]) {

swap(&array\_One[j], &array\_One[j + 1]);

}

}

}

}

void sortArrayTwo(int array\_Two[], int n) {

for (int i = 0; i < n - 1; i++) {

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

if (array\_Two[j] < array\_Two[j + 1]) {

swap(&array\_Two[j], &array\_Two[j + 1]);

}

}

}

}

int minSumOfProducts(int array\_One[], int array\_Two[], int n) {

sortArrayOne(array\_One, n);

sortArrayTwo(array\_Two, n);

int sum = 0;

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

sum += array\_One[i] \* array\_Two[i];

}

return sum;

}

int main() {

int n;

scanf("%d", &n);

int array\_One[n], array\_Two[n];

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

scanf("%d", &array\_One[i]);

}

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

scanf("%d", &array\_Two[i]);

}

int minSum = minSumOfProducts(array\_One, array\_Two, n);

printf("%d\n", minSum);

return 0;

}

Week 3

1. Number of Zeros in a Given Array

#include <stdio.h>

int countZeroes(int arr[], int a, int b) {

if (a == b) {

return 1 - arr[a];

}

int mid = (a + b) / 2;

int leftZeroes = countZeroes(arr, a, mid);

int rightZeroes = countZeroes(arr, mid + 1, b);

return leftZeroes + rightZeroes;

}

int main() {

int n;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

int zeroes = countZeroes(arr, 0, n - 1);

printf("%d\n", zeroes);

return 0;

}

1. Majority Element

#include <stdio.h>

#include <stdlib.h>

int majorityElement(int\*nums, int numsSize) {

int count = 0;

int candidate = 0;

for (int i = 0; i < numsSize; i++) {

if (count == 0) {

candidate = nums[i];

}

count += (nums[i] == candidate) ? 1 : -1;

}

return candidate;

}

int main() {

int n;

scanf("%d", &n);

int\* nums = (int\*)malloc(n \* sizeof(int));

if (nums == NULL) {

return 1;

}

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

scanf("%d", &nums[i]);

}

int result = majorityElement(nums, n);

printf("%d\n", result);

free(nums);

return 0;

}

1. Finding Floor Value

#include<stdio.h>

int findfloor(int arr[],int n,int x){

int low=0,high = n -1;

int floor = -1;

while(low <= high){

int mid = (low+high)/2;

if(arr[mid] == x){

return arr[mid];

} else if(arr[mid] < x){

floor=arr[mid];

low = mid + 1;

}else{

high= mid -1;

}

}

return floor;

}

int main(){

int n,x;

scanf("%d",&n);

int arr[n];

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

scanf("%d",&arr[i]);

}

scanf("%d",&x);

printf("%d\n", findfloor(arr,n,x));

return 0;

}

1. Two Elements sum to x

#include <stdio.h>

void sum(int arr[], int n, int x) {

int left = 0;

int right = n - 1;

while (left < right) {

int csum = arr[left] + arr[right];

if (csum == x) {

printf("%d\n", arr[left]);

printf("%d\n", arr[right]);

return;

} else if (csum < x) {

left++;

} else {

right--;

}

}

printf("No\n");

}

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);

sum(arr, n, x);

return 0;

}

1. Implementation of Quick Sort

#include<stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int p = arr[low];

int i = low;

int j = high;

while (i < j) {

while (arr[i] <= p && i < high) {

i++;

}

while (arr[j] > p && j > low) {

j--;

}

if (i < j) {

swap(&arr[i], &arr[j]);

}

}

swap(&arr[low], &arr[j]);

return j;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 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]);

}

return 0;

}

Week 4

1. DP-Playing with Numbers

#include<stdio.h>

int main()

{

int n;

scanf("%d",&n);

long long arr[n+1];

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

if(i==0){

arr[0]=1;

}

else{

arr[i]=0;

}

}

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

if(i-1>=0)

arr[i]+=arr[i-1];

if(i-3>=0)

arr[i]+=arr[i-3];

}

printf("%lld",arr[n]);

}

1. DP-Playing with chessboard

#include<stdio.h>

int main()

{

int n,i,j;

scanf("%d",&n);

int arr[n][n];

for(i=0;i<n;i++){

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

scanf("%d",&arr[i][j]);

}

}

int max(int a,int b){

if(a>b){

return a;

}

else{

return b;

}

}

int dp[n][n];

dp[0][0]=arr[0][0];

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

dp[j][0]=dp[j-1][0]+arr[j][0];

}

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

dp[0][i]=dp[0][i-1]+arr[0][i];

}

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

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

dp[i][j]=arr[i][j]+max(dp[i-1][j],dp[i][j-1]);

}

}

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

}

1. DP-Longest Common Subsequence

#include <stdio.h>

#include <string.h>

#define MAX\_LENGTH 100

int main() {

char s1[MAX\_LENGTH], s2[MAX\_LENGTH];

int dp[MAX\_LENGTH][MAX\_LENGTH];

int m, n;

// Input the two strings

scanf("%s", s1);

scanf("%s", s2);

m = strlen(s1);

n = strlen(s2);

// Initialize the DP table

for (int i = 0; i <= m; i++) {

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

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

dp[i][j] = 0; // LCS of any string with an empty string is 0

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

dp[i][j] = dp[i - 1][j - 1] + 1; // Characters match

} else {

dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1]; // Characters do not match

}

}

}

// The length of the longest common subsequence

printf("%d\n", dp[m][n]);

return 0;

}

1. DP-Longest non-decreasing Subsequence

#include <stdio.h>

#define MAX\_LENGTH 100

int main() {

int n;

int arr[MAX\_LENGTH];

int dp[MAX\_LENGTH];

// Input the size of the sequence

scanf("%d", &n);

// Input the sequence

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

scanf("%d", &arr[i]);

}

// Initialize the DP table

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

dp[i] = 1; // Each element is a subsequence of length 1

}

// Fill the DP table

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

for (int j = 0; j < i; j++) {

if (arr[j] <= arr[i]) {

dp[i] = (dp[i] > dp[j] + 1) ? dp[i] : (dp[j] + 1);

}

}

}

// Find the maximum value in the DP table

int max\_length = 0;

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

if (dp[i] > max\_length) {

max\_length = dp[i];

}

}

// Output the length of the longest non-decreasing subsequence

printf("%d\n", max\_length);

return 0;

}

Week 5

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

#include<stdio.h>

int main()

{

int n;

int j;

scanf("%d",&n);

int arr[n];

int i;

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

{

scanf("%d",&arr[i]);

}

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

{

for(j=i+1;j<n;j++)

{

if(arr[j]==arr[i])

{

printf("%d",arr[i]);

}

}

}

return 0;

}

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

#include <stdio.h>

int findDuplicate(int nums[], int size) {

int tortoise = nums[0];

int hare = nums[0];

do {

tortoise = nums[tortoise];

hare = nums[nums[hare]];

} while (tortoise != hare);

int pointer = nums[0];

while (pointer != tortoise) {

pointer = nums[pointer];

tortoise = nums[tortoise];

}

return pointer;

}

int main() {

int n;

scanf("%d", &n);

int nums[n];

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

scanf("%d", &nums[i]);

}

int duplicate = findDuplicate(nums, n);

printf("%d\n", duplicate);

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

void find\_intersection(int arr1[], int n1, int arr2[], int n2) {

int i = 0, j = 0;

int first = 1; // Flag to manage spacing in output

while (i < n1 && j < n2) {

if (arr1[i] == arr2[j]) {

// Print the intersection element

if (!first) {

printf(" "); // Print space before next number

}

printf("%d", arr1[i]);

first = 0; // Set flag to false after first element is printed

i++;

j++;

} else if (arr1[i] < arr2[j]) {

i++;

} else {

j++;

}

}

printf("\n"); // New line after each test case output

}

int main() {

int T;

scanf("%d", &T); // Read number of test cases

while (T--) {

int n1;

scanf("%d", &n1); // Read size of first array

int arr1[n1]; // Declare the first array

for (int i = 0; i < n1; i++) {

scanf("%d", &arr1[i]); // Read elements of first array

}

int n2;

scanf("%d", &n2); // Read size of second array

int arr2[n2]; // Declare the second array

for (int i = 0; i < n2; i++) {

scanf("%d", &arr2[i]); // Read elements of second array

}

// Find and print the intersection of the two arrays

find\_intersection(arr1, n1, arr2, n2);

}

return 0;

}

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

#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {

int i = 0, j = 0;

int found = 0; // To check if we found any intersection

while (i < n1 && j < n2) {

if (arr1[i] == arr2[j]) {

// Print the element if it's part of the intersection

if (found == 0 || arr1[i] != arr1[i - 1]) { // Avoid duplicates

printf("%d ", arr1[i]);

found = 1; // Mark that we found at least one intersection

}

i++;

j++;

} else if (arr1[i] < arr2[j]) {

i++;

} else {

j++;

}

}

if (found) {

printf("\n"); // New line after each test case result

} else {

printf("No intersection\n"); // If no common elements were found

}

}

int main() {

int T; // Number of test cases

scanf("%d", &T);

for (int t = 0; t < T; t++) {

int n1;

scanf("%d", &n1); // Size of the first array

int arr1[n1];

for (int i = 0; i < n1; i++) {

scanf("%d", &arr1[i]); // Elements of the first array

}

int n2;

scanf("%d", &n2); // Size of the second array

int arr2[n2];

for (int i = 0; i < n2; i++) {

scanf("%d", &arr2[i]); // Elements of the second array

}

// Find and print intersection

findIntersection(arr1, n1, arr2, n2);

}

return 0;

}

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

#include <stdio.h>

int has\_pair\_with\_difference(int A[], int n, int k) {

int i = 0, j = 0;

while (j < n) {

int diff = A[j] - A[i];

if (diff == k && i != j) {

return 1; // Pair found

} else if (diff < k) {

j++; // Increase the larger index

} else {

i++; // Increase the smaller index

}

// Ensure i is always less than j

if (i == j) {

j++;

}

}

return 0; // No pair found

}

int main() {

int n;

scanf("%d", &n); // Read number of elements in the array

int A[n];

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

scanf("%d", &A[i]); // Read elements of the array

}

int k;

scanf("%d", &k); // Read the non-negative integer k

// Check if a pair exists

int result = has\_pair\_with\_difference(A, n, k);

printf("%d\n", result);

return 0;

}

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

Sum 1:-

#include <stdio.h>

int has\_pair\_with\_difference(int A[], int n, int k) {

int i = 0, j = 0;

while (j < n) {

int diff = A[j] - A[i];

if (diff == k && i != j) {

return 1; // Pair found

} else if (diff < k) {

j++; // Increase the larger index to find a larger difference

} else {

i++; // Increase the smaller index to find a smaller difference

}

// Ensure i is always less than j

if (i == j) {

j++;

}

}

return 0; // No pair found

}

int main() {

int n;

scanf("%d", &n); // Read number of elements in the array

int A[n];

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

scanf("%d", &A[i]); // Read elements of the array

}

int k;

scanf("%d", &k); // Read the non-negative integer k

// Check if a pair exists

int result = has\_pair\_with\_difference(A, n, k);

printf("%d\n", result);

return 0;

}

Sum 2:-

#include <stdio.h>

int difference(int A[], int n, int k) {

int i = 0, j = 0;

while (j < n) {

int diff = A[j] - A[i];

if (diff == k && i != j) {

return 1;

} else if (diff < k) {

j++;

} else {

i++;

}

if (i == j) {

j++;

}

}

return 0;

}

int main() {

int n;

scanf("%d", &n);

int A[n];

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

scanf("%d", &A[i]);

}

int k;

scanf("%d", &k);

int result =difference(A, n, k);

printf("%d\n", result);

}