

**Academic Year 2021-2022**

**Department: Computer Science & Engineering**

**LAB TERM WORK**

**OF**

**DESIGN AND ANALYSIS OF**

**ALGORITHMS**

Submitted to: Submitted by:

**Dr. Upma Jain Mohd Nasir (F-14)**

**DAA LAB PROGRAM**

**Name: - Mohd Nasir University Roll No: -2016855**

**Section: - ‘F’ Student Id: -20021595**

**Roll No: - 14 Enrolment No: -GE202016855**

WEEK 1

**/\*Ques1. Given an array of nonnegative integers, design a linear**

**algorithm and implement it using a program to find whether**

**given key element is present in the array or not. Also, find**

**total number of comparisons for each input case. (Time Complexity**

**= O(n), where n is the size of input)\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

bool LinearSearch(int \*arr, int size, int \*comparisons);

int main()

{

int test;

printf("Please Enter no of Test :");

scanf("%d", &test);

while (test--)

{

int size, comparisons = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

if (LinearSearch(arr, size, &comparisons))

printf("Present ");

else

printf("Not Present ");

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

}

return 0;

}

bool LinearSearch(int \*arr, int size, int \*comparisons)

{

int key;

printf("Please Enter a key :");

scanf("%d", &key);

for (int i = 0; i < size; i++)

{

\*(comparisons) = \*(comparisons) + 1;

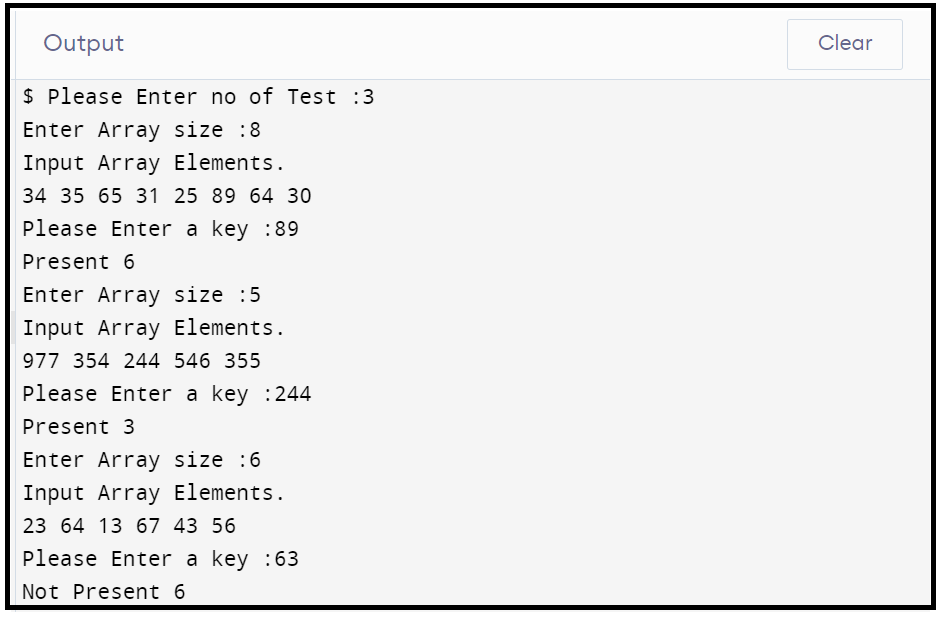
if (arr[i] == key)

return true;

}

return false;

}



**/\*Ques2(Week1). Given an already sorted array of positive**

**integers,design an algorithm and implement it using a program**

**to find whether given key element is present in the**

**array or not. Also, find total number of comparisons**

**for each input case. (Time Complexity = O(nlogn),**

**where n is the size of input).\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

bool BinarySearch(int \*arr, int low, int high, int \*comparisons);

int main()

{

int size, comparisons = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

if (BinarySearch(arr, 0, size, &comparisons))

printf("Present ");

else

printf("Not Present ");

printf("%d", comparisons);

return 0;

}

bool BinarySearch(int \*arr, int low, int high, int \*comparisons)

{

int key, mid;

printf("Please Enter a key :");

scanf("%d", &key);

while (low < high)

{

mid = (low + high) / 2;

\*(comparisons) = \*(comparisons) + 1;

if (arr[mid] == key)

return true;

else if (arr[mid] > key)

high = mid - 1;

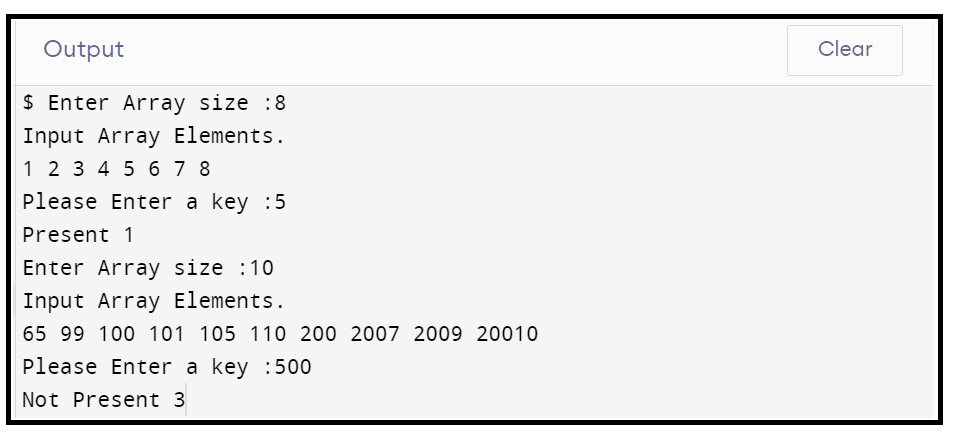
else

low = mid + 1;

}

return false;

}



**/\*Ques3(WEEK1). Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether**

**a given key element is present in the sorted array or not. For**

**an array arr[n], search at the indexes arr[0], arr[2], arr[4],**

**.....,arr[2k] and so on. Once the interval (arr[2k] < key < arr**

**[ 2k+1] ) is found, perform a linear search operation from the**

**index 2k to find the element key. (Complexity < O(n), where n is**

**the number of elements need to be scanned for searching): \*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <math.h>

bool BinarySearch(int \*arr, int size, int \*comparisons);

int main()

{

int test;

printf("Please Enter No of Test :");

scanf("%d", &test);

while (test--)

{

int size, comparisons = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

if (BinarySearch(arr, size, &comparisons))

printf("Present\n");

else

printf("Not Present\n");

printf("%d", comparisons);

}

return 0;

}

bool BinarySearch(int \*arr, int size, int \*comparisons)

{

int key, mid, j;

printf("Please Enter a key :");

scanf("%d", &key);

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

{

j = pow(2, i);

\*(comparisons) = \*(comparisons) + 1;

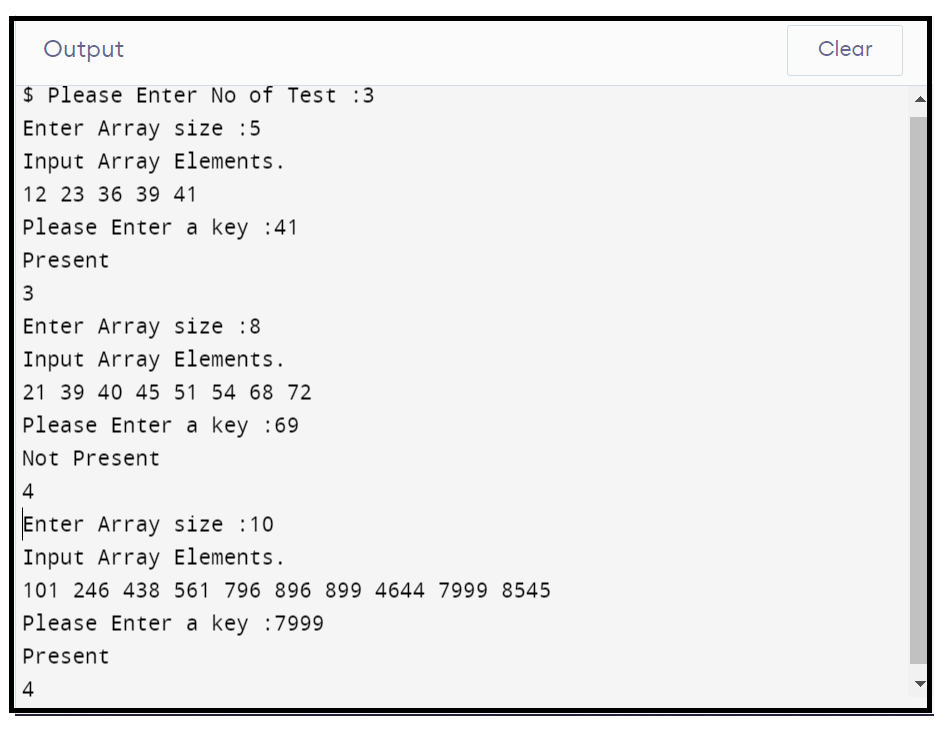
if (arr[j] == key)

return true;

}

return false;

}



**WEEK 2**

**/\*Ques1(Week2). Given a sorted array of positive integers containing few**

**duplicate elements, design an algorithm and implement it using**

**a program to find whether the given key element is present in**

**the array or not. If present, then also find the number of copies**

**of given key. (Time Complexity = O(log n))\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

int BinarySearch(int \*arr, int low, int high, int \*comparisons);

int main()

{

int test;

printf("Input No of Test :");

scanf("%d", &test);

while (test--)

{

int size, comparisons = 0, index = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

index = BinarySearch(arr, 0, size, &comparisons);

int i = index - 1, j = index + 1, count = 1;

if (!index)

printf("Not Present ");

else

{

while (arr[i] == arr[index] || arr[j] == arr[index])

{

if ((i < 0 && j >= size) || (arr[i] != arr[index] && arr[j] != arr[index]))

break;

if (i >= 0 && arr[i] == arr[index])

{

i--;

count++;

}

if (j < size && arr[j] == arr[index])

{

j++;

count++;

}

}

printf("%d -> %d\n", arr[index], count);

}

}

return 0;

}

int BinarySearch(int \*arr, int low, int high, int \*comparisons)

{

int key, mid;

printf("Please Enter a key :");

scanf("%d", &key);

while (low < high)

{

mid = (low + high) / 2;

\*(comparisons) = \*(comparisons) + 1;

if (arr[mid] == key)

return mid;

else if (arr[mid] > key)

high = mid - 1;

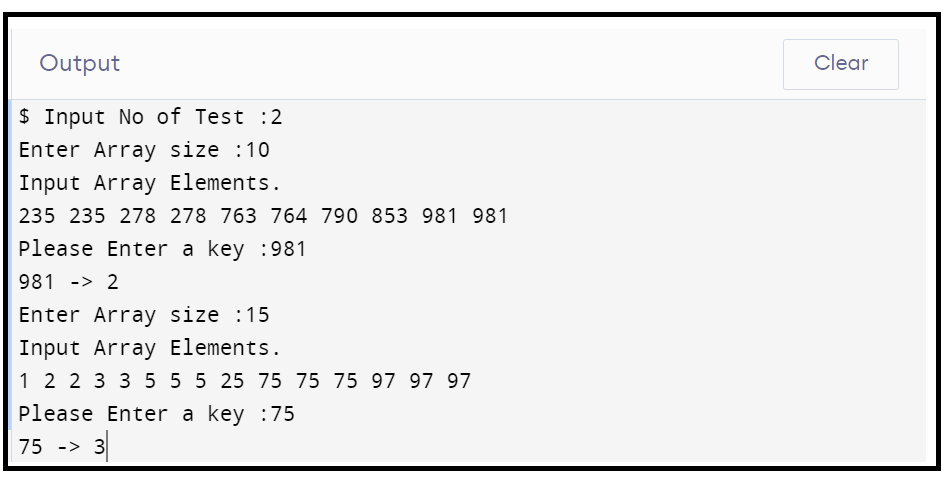
else

low = mid + 1;

}

return 0;

}



**/\*Ques2(Week2).Given a sorted array of positive integers, design an**

**algorithm and implement it using a program to find three**

**indices i, j, k such that arr[i] + arr[j] = arr[k]\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void IndexFound(int \*arr, int size);

int main()

{

int test;

printf("Please Input No of Test Caase :");

scanf("%d", &test);

while (test--)

{

int size;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

IndexFound(arr, size);

}

return 0;

}

void IndexFound(int \*arr, int size)

{

int i, j, k;

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

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

for (k = j + 1; k < size; k++)

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

{

printf("%d, %d, %d\n", i + 1, j + 1, k + 1);

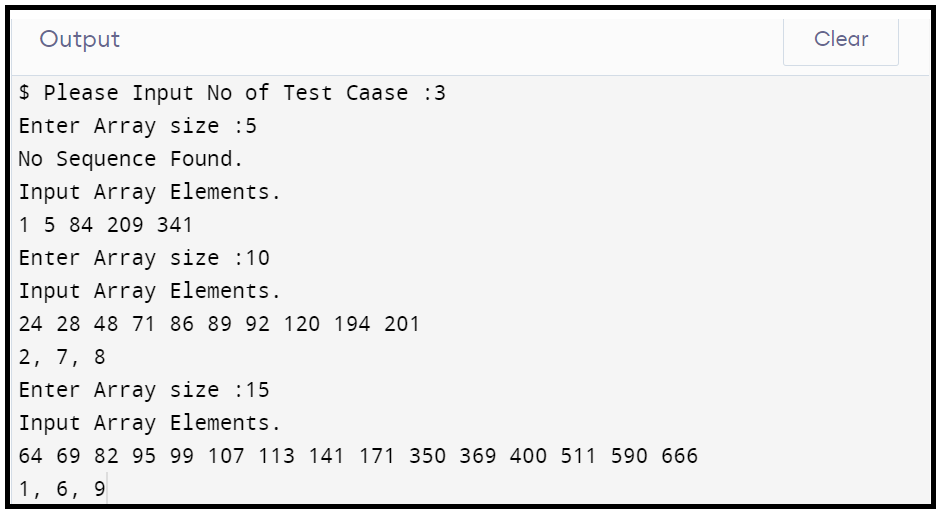
break;

}

if (i == size)

printf("No Sequence Found\n");

}



/\*Ques3(Week2).Given an array of nonnegative integers, design an

algorithm and a program to count the number of pairs of integers

such that their difference is equal to a given key, K.\*/

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

int DiffrencePairs(int \*arr, int size);

int main()

{

int test;

printf("Please input no of test case :");

scanf("%d", &test);

while (test--)

{

int size;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

printf("%d\n", DiffrencePairs(arr, size));

}

return 0;

}

int DiffrencePairs(int \*arr, int size)

{

int diff, count = 0;

printf("Please Enter a Difference :");

scanf("%d", &diff);

for (int i = 0; i < size; i++)

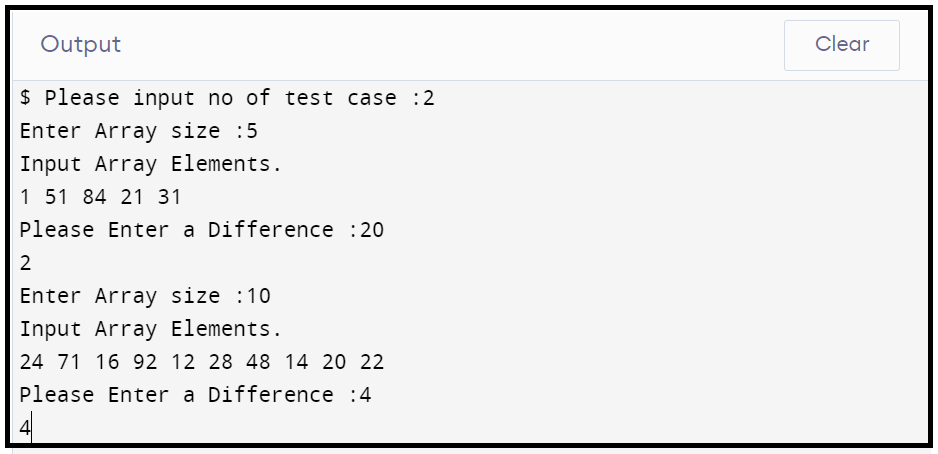
for (int j = i + 1; j < size; j++)

if (abs(arr[i] - arr[j]) == diff)

count++;

return count;

}



**WEEK 3**

**/\*Ques1(Week3).Given an unsorted array of integers, design an algorithm and**

**a program to sort the array using insertion sort. Your program should be able to**

**find number of comparisons and shifts (shifts total number of times the array**

**elements are shifted from their place) required for sorting the array.\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void InsertionSort(int \*arr, int size, int \*shifts, int \*comparisions);

void PrintArray(int \*arr, int size);

int main()

{

int test;

printf("Please Input No of Test Case :");

scanf("%d", &test);

while (test--)

{

int size, shifts = 0, comparisions = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

InsertionSort(arr, size, &shifts, &comparisions);

PrintArray(arr, size);

printf("\nComparisions = %d\nShifts = %d\n", comparisions, shifts);

}

return 0;

}

void InsertionSort(int \*arr, int size, int \*shifts, int \*comparisions)

{

int i, j, key;

for (i = 1; i < size; i++)

{

key = arr[i];

j = i - 1;

while (j >= 0 && key < arr[j])

{

\*(comparisions) = \*(comparisions) + 1;

arr[j + 1] = arr[j];

j--;

}

\*(shifts) = \*(shifts) + 1;

arr[j + 1] = key;

}

\*(shifts) = \*(shifts) + \*(comparisions);

}

void PrintArray(int \*arr, int size)

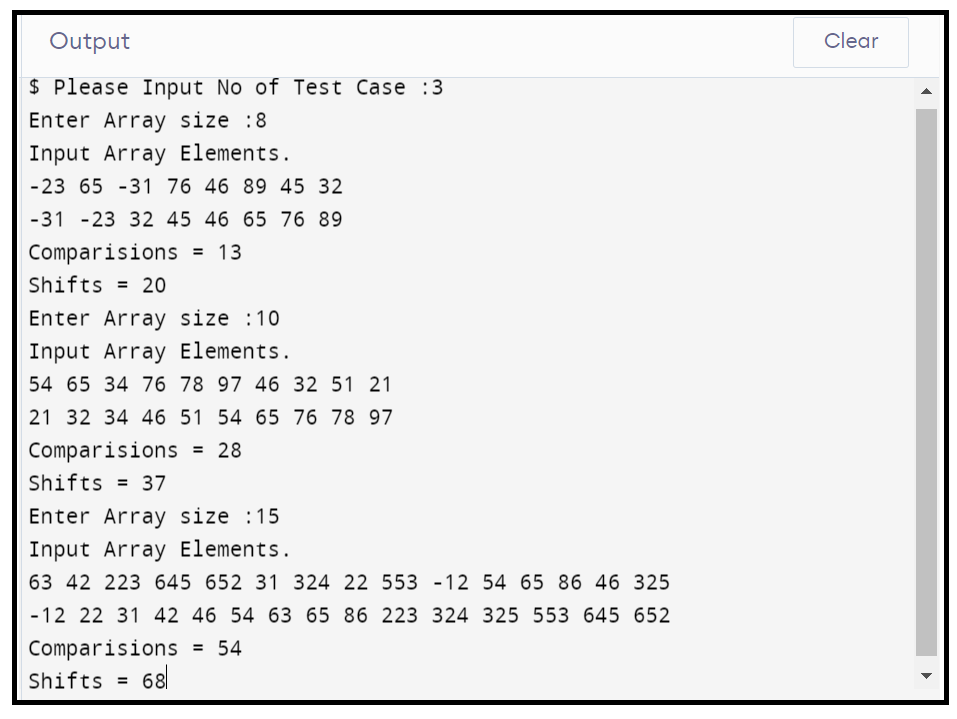
{

int i;

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

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

}



**/\*Ques2(Week3) .Given an unsorted array of integers, design an algorithm and**

**implement a program to sort this array using selection sort. Your program should**

**also find number of comparisons and number of swaps required\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void SelectionSort(int \*arr, int size, int \*swap, int \*comparisions);

void PrintArray(int \*arr, int size);

int main()

{

int test;

printf("Please Input no of Test :");

scanf("%d", &test);

while (test--)

{

int size, swap = 0, comparisions = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

SelectionSort(arr, size, &swap, &comparisions);

PrintArray(arr, size);

printf("\nComparisions = %d\nSwaps = %d\n", comparisions, swap);

}

return 0;

}

void SelectionSort(int \*arr, int size, int \*swap, int \*comparisions)

{

int min, pos, i, j;

for (i = 0; i < size - 1; i++)

{

min = arr[i];

pos = i;

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

{

\*(comparisions) = \*(comparisions) + 1;

if (min > arr[j])

{

min = arr[j];

pos = j;

}

}

/\*\*(swap) = \*(swap) + 1;\*/

if (pos != i)

{

\*(swap) = \*(swap) + 1;

arr[pos] = arr[i];

arr[i] = min;

}

}

printf("%d %d\n", comparisions, swap);

}

void PrintArray(int \*arr, int size)

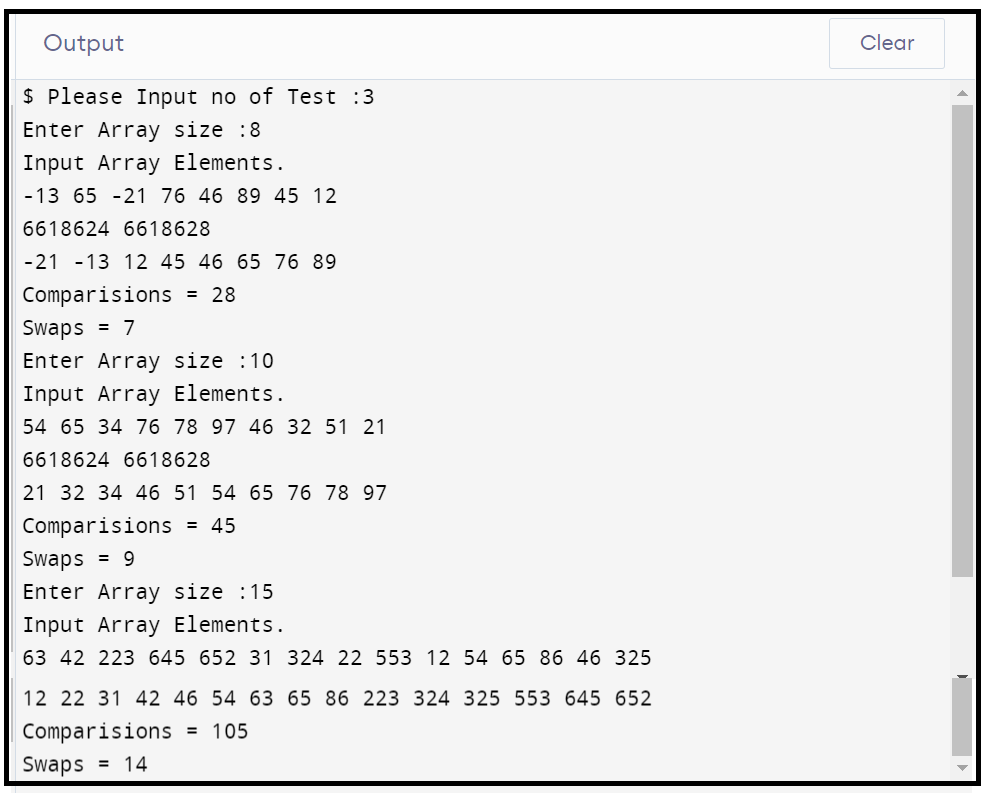
{

int i;

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

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

}



**/\*Ques3(Week3). Given an unsorted array of positive integers, design an algorithm**

**and implement it using a program to find whether there are any duplicate elements**

**in the array or not. (use sorting) (Time Complexity = O(n log n))\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

bool Dublicates(int \*arr, int size);

int MaximumElement(int \*arr, int size);

int main()

{

int Test;

printf("Please Input no of Test Case :");

scanf("%d", &Test);

while (Test--)

{

int size;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

if (Dublicates(arr, size))

printf("YES\n");

else

printf("NO\n");

}

return 0;

}

bool Dublicates(int \*arr, int size)

{

int \*temp = (int \*)malloc((MaximumElement(arr, size) + 1) \* sizeof(int));

int i = 0;

for (i = 0; i <= (MaximumElement(arr, size)); i++)

temp[i] = 0;

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

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

for (i = 0; i <= (MaximumElement(arr, size)); i++)

if (temp[i] >= 2)

return true;

return false;

}

int MaximumElement(int \*arr, int size)

{

int max = INT\_MIN;

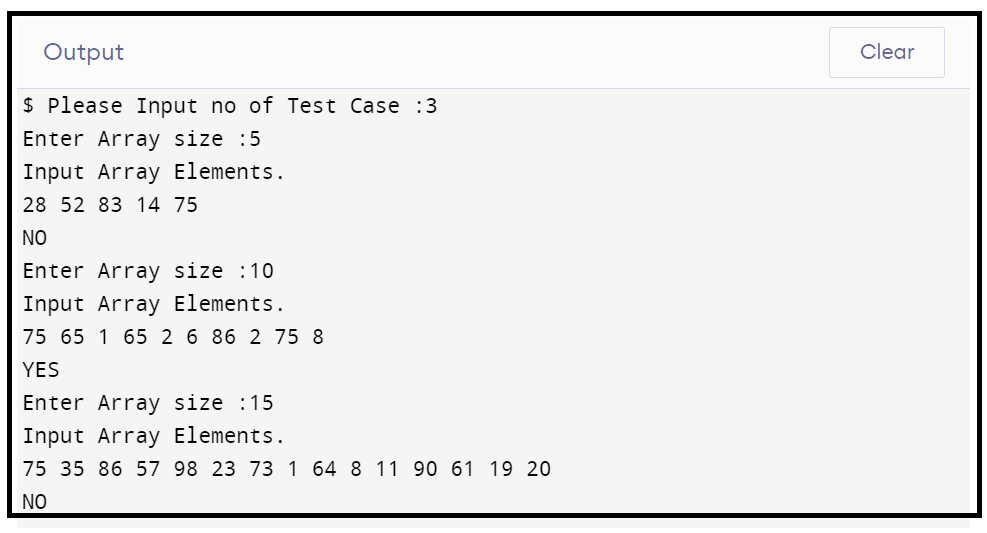
for (int i = 0; i < size; i++)

if (max < arr[i])

max = arr[i];

return max;

}



**WEEK 4**

**/\*Ques1(Week4).Given an unsorted array of integers, design an algorithm and**

**implement it using a program to sort an array of elements by dividing the array**

**into two subarrays and combining these subarrays after sorting each one of them.**

**Your program should also find number of comparisons and inversions during**

**sorting the array\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void MergeSort(int \*arr, int \*temp, int low, int high);

void MergeElements(int \*arr, int \*temp, int low, int mid, int high);

void PrintArray(int \*arr, int size);

int comparisions = 0;

int main()

{

int test;

printf("Please Input No of Test :");

scanf("%d", &test);

while (test--)

{

comparisions = 0;

int size;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

int \*temp = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

MergeSort(arr, temp, 0, size - 1);

PrintArray(arr, size);

printf("\nComparisions = %d\n", comparisions);

}

return 0;

}

void MergeSort(int \*arr, int \*temp, int low, int high)

{

int mid;

if (low < high)

{

mid = (low + high) / 2;

MergeSort(arr, temp, low, mid);

MergeSort(arr, temp, mid + 1, high);

MergeElements(arr, temp, low, mid, high);

}

}

void MergeElements(int \*arr, int \*temp, int low, int mid, int high)

{

int i = low, j = mid + 1, k = low;

while (i <= mid && j <= high)

{

comparisions++;

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

{

temp[k] = arr[i];

i++;

k++;

}

else

{

temp[k] = arr[j];

j++;

k++;

}

}

while (i <= mid)

{

temp[k] = arr[i];

k++;

i++;

}

while (j <= high)

{

temp[k] = arr[j];

j++;

k++;

}

for (i = low; i <= high; i++)

arr[i] = temp[i];

}

void PrintArray(int \*arr, int size)

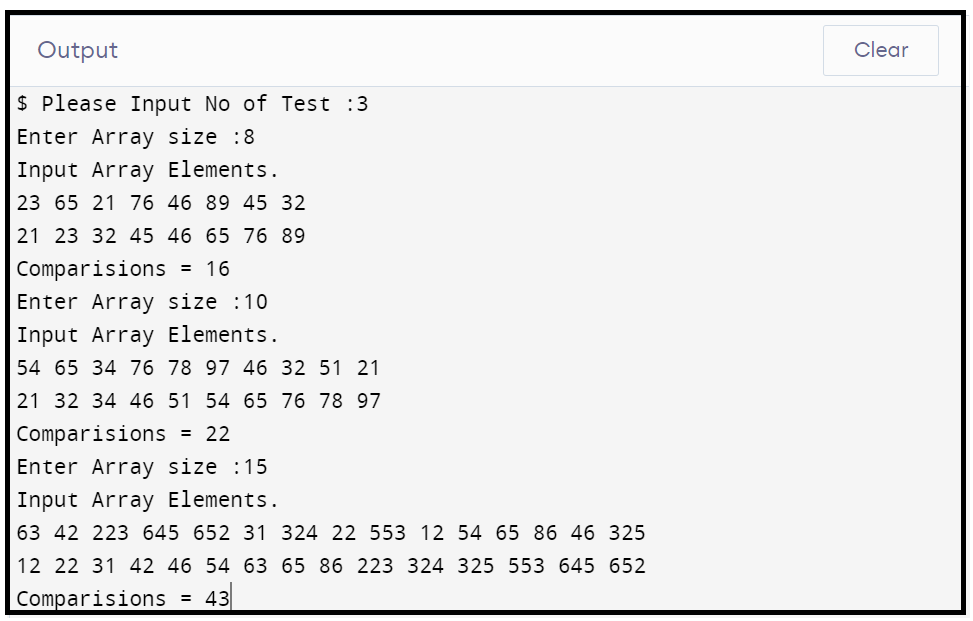
{

int i;

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

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

}



**/\*Ques2(Week4). Given an unsorted array of integers, design an algorithm**

**and implement it using a program to sort an array of elements by**

**partitioning the array into two subarrays based on a pivot element**

**such that one of the sub array holds values smaller than the pivot**

**element while another sub array holds values greater than the pivot**

**element. Pivot element should be selected randomly from the array.**

**Your program should also find number of comparisons and swaps required**

**for sorting the array.\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void QuickSort(int \*arr, int low, int high);

int ArrayPartition(int \*arr, int low, int high);

void swap(int \*arr, int start, int end);

void PrintArray(int \*arr, int size);

int comparisions = 0, swaps = 0;

int main()

{

int test;

printf("Please input no of test case :");

scanf("%d", &test);

while (test--)

{

int size;

comparisions = 0, swaps = 0;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

int \*temp = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

QuickSort(arr, 0, size - 1);

PrintArray(arr, size);

printf("\nComparision = %d\nSwaps = %d\n", comparisions, swaps);

}

return 0;

}

int ArrayPartition(int \*arr, int low, int high)

{

int pivot = arr[low];

int i = low + 1, j = high;

while (i <= j)

{

while (arr[i] <= pivot)

{

i++;

comparisions++;

}

while (arr[j] > pivot)

{

j--;

comparisions++;

}

/\*swaps++;\*/

if (i < j)

{

swaps++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

swaps++;

arr[low] = arr[j];

arr[j] = pivot;

return j;

}

void swap(int \*arr, int start, int end)

{

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

}

void QuickSort(int \*arr, int low, int high)

{

int PartitionIndex;

if (low < high)

{

int randomly = (rand() % (high - low + 1)) + low;

swap(arr, low, randomly);

PartitionIndex = ArrayPartition(arr, low, high);

QuickSort(arr, low, PartitionIndex - 1);

QuickSort(arr, PartitionIndex + 1, high);

}

}

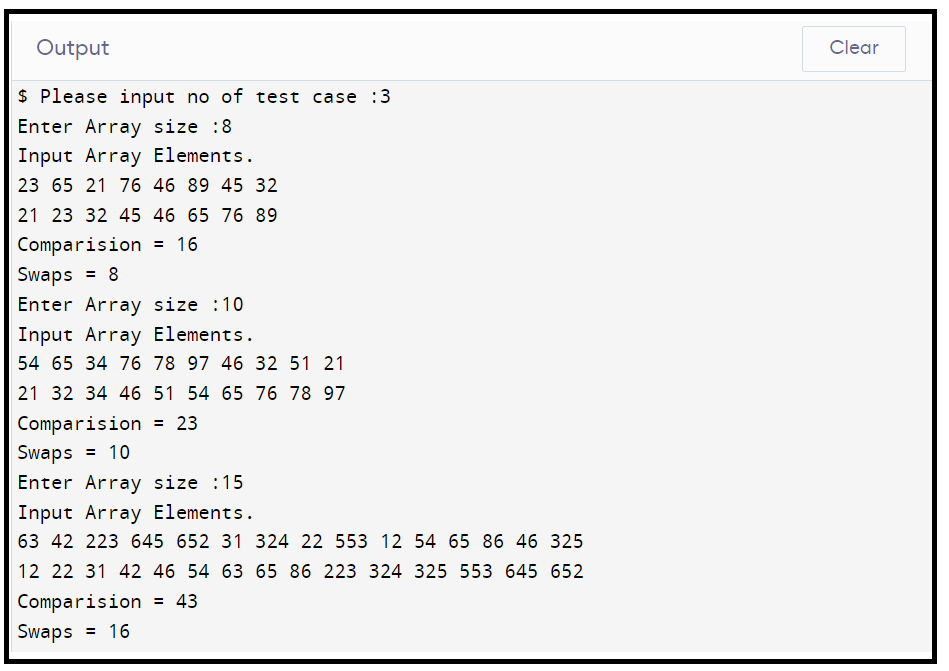
void PrintArray(int \*arr, int size)

{

int i;

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

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

}

**/\*Ques3(Week4). Given an unsorted array of integers, design an**

**algorithm and implement it using a program to find Kth smallest or**

**largest element in the array. (Worst case Time Complexity = O(n))\*/**

#include <stdio.h>

#include <stdlib.h>

int Partition(int \*arr, int low, int high);

int QuickSelect(int \*arr, int low, int high, int K);

int main()

{

int test;

printf("Please input no of Test :");

scanf("%d", &test);

while (test--)

{

int size, K, i;

printf("Input the size of Array :");

scanf("%d", &size);

int \*arr = (int \*)malloc(size \* sizeof(int));

printf("Input the Array Elements.\n");

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

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

printf("Enter the number 'K' to find 'Kth' Largest and Smallest Elements :");

scanf("%d", &K);

printf("%d Smallest Elemenets is :%d\n", K, QuickSelect(arr, 0, size - 1, K));

printf("%d Largest Elemenets is :%d\n", K, QuickSelect(arr, 0, size - 1, size - K + 1));

}

return 0;

}

int QuickSelect(int \*arr, int low, int high, int K)

{

if (low == high)

return arr[low];

int PivotIndex = Partition(arr, low, high);

int SizeofArray = PivotIndex - low + 1;

if (SizeofArray > K)

return QuickSelect(arr, low, PivotIndex - 1, K);

else if (SizeofArray < K)

return QuickSelect(arr, PivotIndex + 1, high, K - SizeofArray);

else

return arr[PivotIndex];

}

int Partition(int \*arr, int low, int high)

{

int pivot = arr[low];

int i = low + 1;

int j = high;

while (i <= j)

{

while (arr[i] <= pivot)

i++;

while (arr[j] > pivot)

j--;

if (i < j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

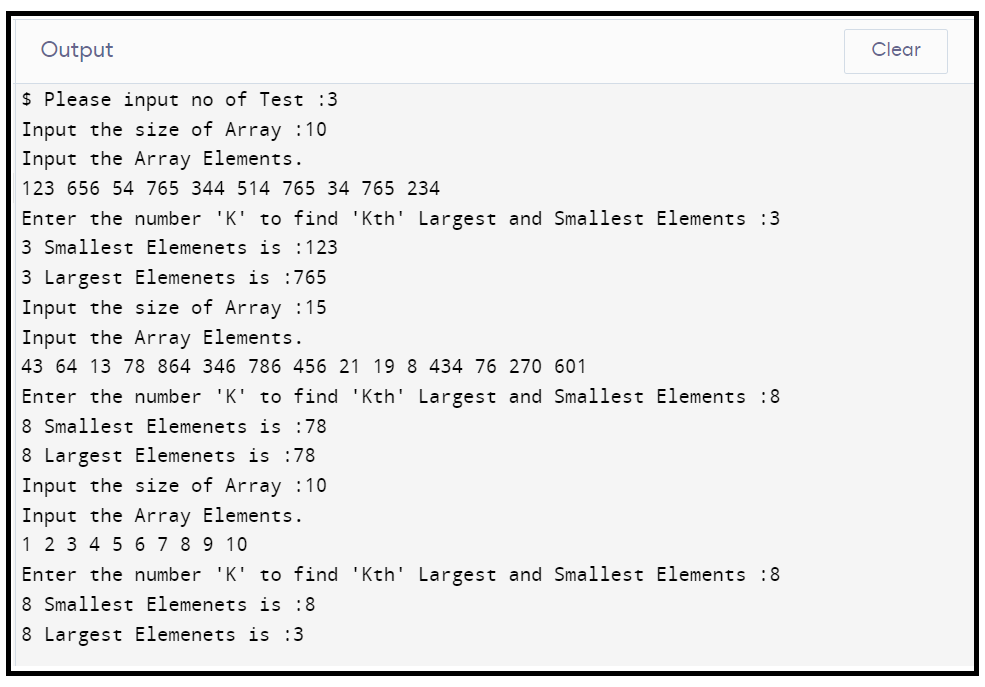
}

arr[low] = arr[j];

arr[j] = pivot;

return j;

}



**WEEK 5**

**/\*Ques1(Week5).Given an unsorted array of alphabets containing duplicate**

**elements. Design an algorithm and implement it using a program to find**

**which alphabet has maximum number of occurrences and print it.**

**(Time Complexity = O(n)) (Hint: Use counting sort)\*/**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int test;

printf("Test Cases :");

scanf("%d", &test);

while (test--)

{

int size, i, key, max = INT\_MIN, pos = 0;

char ch;

printf("Please Input Array size :");

scanf("%d", &size);

int \*arr = (int \*)malloc(size \* sizeof(int));

int \*temp = (int \*)malloc(26 \* sizeof(int));

for (i = 0; i < 26; i++)

temp[i] = 0;

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

{

scanf(" %c", &ch);

key = ch - 'a';

arr[i] = key;

if (max < arr[i])

max = arr[i];

}

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

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

int max1 = INT\_MIN;

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

{

if (max1 < temp[i])

{

max1 = temp[i];

pos = i;

}

}

if (max1 >= 2)

{

ch = 'a' + pos;

printf("%c -> %d\n", ch, max1);

}

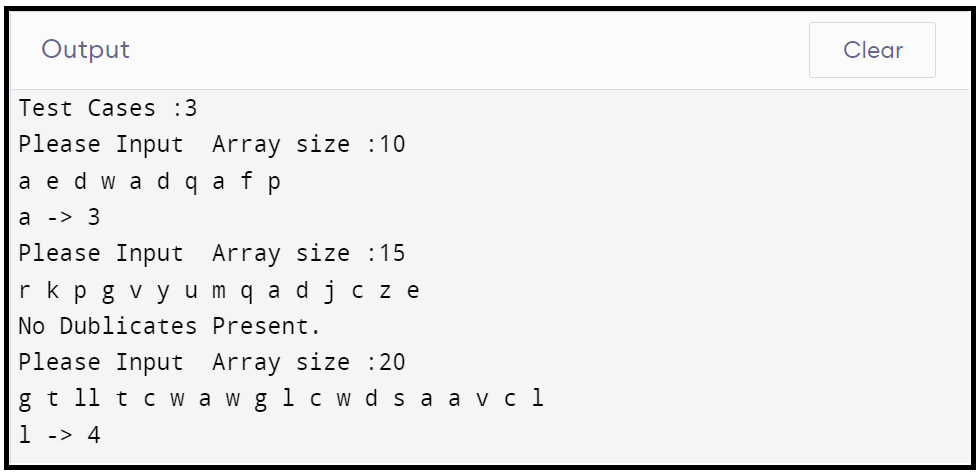
else

printf("No Dublicates Present.\n");

}

return 0;

}



**/\*Ques2(Week5). Given an unsorted array of integers, design an**

**algorithm and implement it using a program to find whether two**

**elements exist such that their sum is equal to the given key**

**element. (Time Complexity = O(n log n))\*/**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void MergeSort(int \*arr, int \*temp, int low, int high);

void MergeElements(int \*arr, int \*temp, int low, int mid, int high);

bool SumCheck(int \*arr, int low, int high);

int main()

{

int test;

printf("Test Cases :");

scanf("%d", &test);

while (test--)

{

int size;

printf("Enter Array size :");

scanf("%d", &size);

printf("Input Array Elements.\n");

int \*arr = (int \*)malloc(size \* sizeof(int));

int \*temp = (int \*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

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

MergeSort(arr, temp, 0, size - 1);

if (!(SumCheck(arr, 0, size - 1)))

printf("No Such Pairs Exist.\n");

}

return 0;

}

bool SumCheck(int \*arr, int low, int high)

{

int sum, temp = 0;

printf("input sum :");

scanf("%d", &sum);

while (low < high)

{

if ((arr[low] + arr[high]) == sum)

{

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

low++;

high--;

temp = 1;

}

else if ((arr[low] + arr[high]) > sum)

high--;

else

low++;

}

if (!temp)

return false;

else

return true;

}

void MergeSort(int \*arr, int \*temp, int low, int high)

{

int mid;

if (low < high)

{

mid = (low + high) / 2;

MergeSort(arr, temp, low, mid);

MergeSort(arr, temp, mid + 1, high);

MergeElements(arr, temp, low, mid, high);

}

}

void MergeElements(int \*arr, int \*temp, int low, int mid, int high)

{

int i = low, j = mid + 1, k = low;

while (i <= mid && j <= high)

{

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

{

temp[k] = arr[i];

i++;

k++;

}

else

{

temp[k] = arr[j];

j++;

k++;

}

}

while (i <= mid)

{

temp[k] = arr[i];

k++;

i++;

}

while (j <= high)

{

temp[k] = arr[j];

j++;

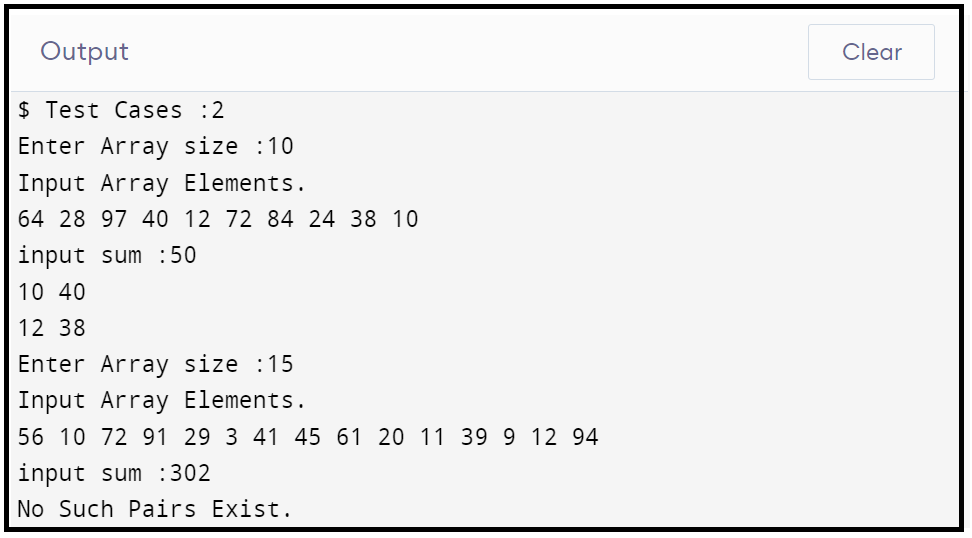
k++;

}

for (i = low; i <= high; i++)

arr[i] = temp[i];

}



**/\*Ques3(Week5).You have been given two sorted integer arrays of size**

**m and n.Design an algorithm and implement it using a program to find**

**list of elements which are common to both. (Time Complexity = O(m+n))\*/**

#include <stdio.h>

#include <stdlib.h>

void SameElements(int \*arr1, int \*arr2, int size1, int size2);

int main()

{

int test;

printf("Test Cases :");

scanf("%d", &test);

while (test--)

{

int size1, size2, i;

printf("Enter array1 size :");

scanf("%d", &size1);

printf("Enter array2 size :");

scanf("%d", &size2);

int \*arr1 = (int \*)malloc(size1 \* sizeof(int));

int \*arr2 = (int \*)malloc(size2 \* sizeof(int));

printf("Input array1 Elemenst.\n");

for (i = 0; i < size1; i++)

scanf("%d", &arr1[i]);

printf("Input array2 Elemenst.\n");

for (i = 0; i < size2; i++)

scanf("%d", &arr2[i]);

SameElements(arr1, arr2, size1, size2);

printf("\n");

}

return 0;

}

void SameElements(int \*arr1, int \*arr2, int size1, int size2)

{

int i = 0, j = 0;

while (i <= size1 && j <= size2)

{

if (arr1[i] < arr2[j])

i++;

else if (arr1[i] > arr2[j])

j++;

else

{

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

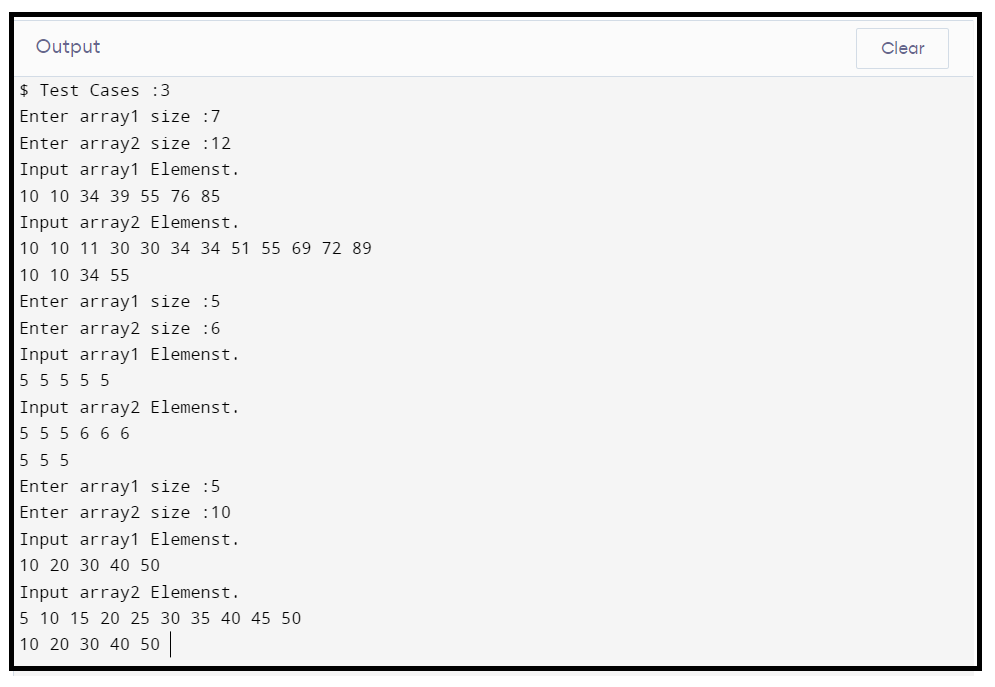
i++;

j++;

}

}

}



**DAA LAB PROGRAM**

**Name: - Mohd Nasir University Roll No: -2016855**

**Section: - ‘F’ Student Id: -20021595**

**Roll No: - 14 Enrolment No: -GE202016855**

**Week 6.**

**/\*Ques1. Given a (directed/undirected) graph, design an algorithm**

**and implement it using a program to find if a path exists between**

**two given vertices or not. (Hint: use DFS)\*/**

#include <iostream>

#include <vector>

#include <stdbool.h>

using namespace std;

void DFS(int source, vector<vector<int>> matrix, vector<bool> &visited);

int main()

{

int ch;

cout << "1. For Adjacency Matrix as a Input." << endl;

cout << "2. For Edges and Vertex as a Input." << endl;

cin >> ch;

if (ch == 1)

{

int source, destination, size, t;

cout << "Input size :";

cin >> size;

vector<bool> visited(size, false);

vector<vector<int>> matrix;

vector<int> temp;

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

cin >> t;

temp.push\_back(t);

}

matrix.push\_back(temp);

temp.clear();

}

cout << "Input Source and Destination :";

cin >> source >> destination;

DFS(source - 1, matrix, visited);

if (visited[destination - 1] == true)

cout << "\nYes Path Exists.";

else

cout << "\nNo Such Path Exists.";

}

else

{

int source, destination;

int edges, vertex;

cout << "Input no of Vertex and Edges :";

cin >> vertex >> edges;

vector<bool> visited(vertex);

vector<vector<int>> matrix;

vector<int> temp;

for (int i = 0; i < vertex; i++)

{

for (int j = 0; j < vertex; j++)

{

temp.push\_back(0);

}

matrix.push\_back(temp);

temp.clear();

}

for (int i = 0; i < edges; i++)

{

int u, v;

cin >> u >> v;

matrix[u - 1][v - 1] = 1;

}

cout << "Input Source and Destination :";

cin >> source >> destination;

DFS(source - 1, matrix, visited);

if (visited[destination - 1] == true)

cout << "\nYes Path Exists.";

else

cout << "\nNo Such Path Exists.";

}

return 0;

}

void DFS(int source, vector<vector<int>> matrix, vector<bool> &visited)

{

printf("%d ", source);

visited[source] = true;

for (int neigh = 0; neigh < matrix.size(); neigh++)

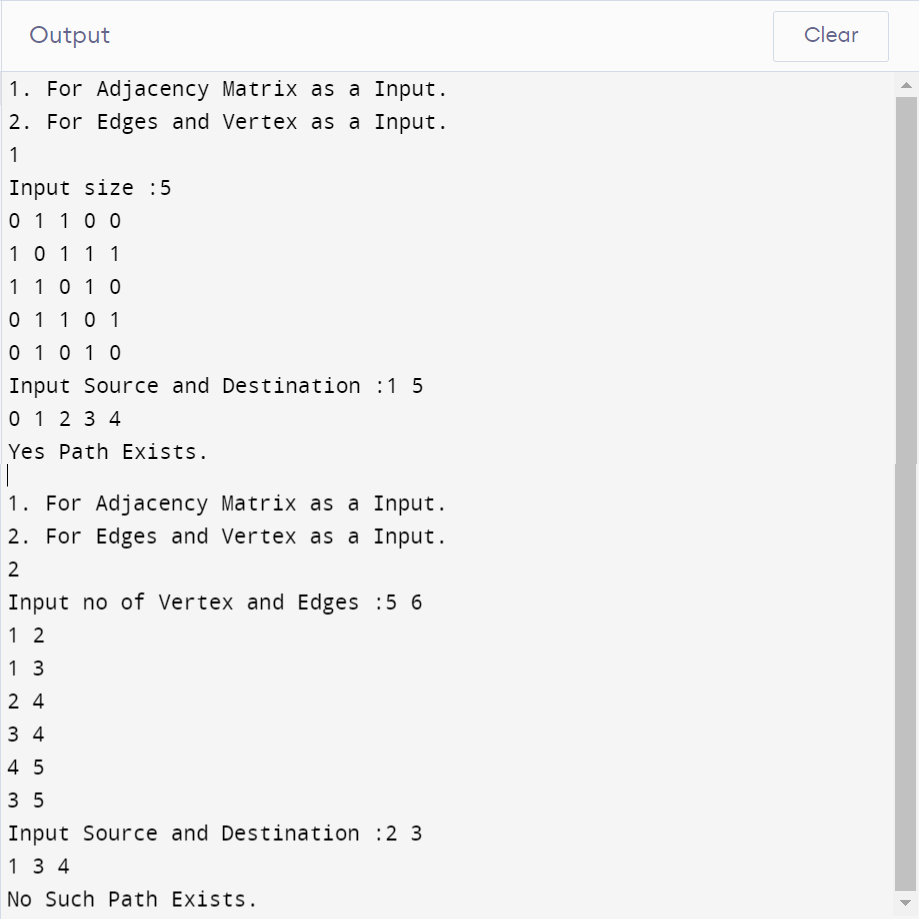
{

if (matrix[source][neigh] == 1 && visited[neigh] == false)

DFS(neigh, matrix, visited);

}

}



**/\*Ques2.Given a graph, design an algorithm and implement it using**

**a program to find if a graph is bipartite or not. (Hint: use BFS)\*/**

#include <iostream>

#include <queue>

using namespace std;

bool bipar(int \*\*AdjM, int V)

{

queue<int> Q;

int \*color = new int[V]();

int curr = 1;

color[0] = curr;

Q.push(0);

while (!Q.empty())

{

int u = Q.front();

Q.pop();

if (AdjM[u][u])

return false;

curr \*= -1;

for (int v = 0; v < V; v++)

{

if (AdjM[u][v] && color[v] == 0)

{

color[v] = curr;

Q.push(v);

}

else if (AdjM[u][v] && color[u] == color[v])

return false;

}

}

return true;

}

int main()

{

int v, s, d;

cin >> v; // Size of adjacency matrix

int \*\*AdjM; // Adjacency Matrix

AdjM = (int \*\*)malloc(v \* sizeof(int \*));

for (int i = 0; i < v; i++)

AdjM[i] = (int \*)malloc(v \* sizeof(int));

for (int i = 0; i < v; i++)

{

for (int j = 0; j < v; j++)

{

cin >> AdjM[i][j];

}

}

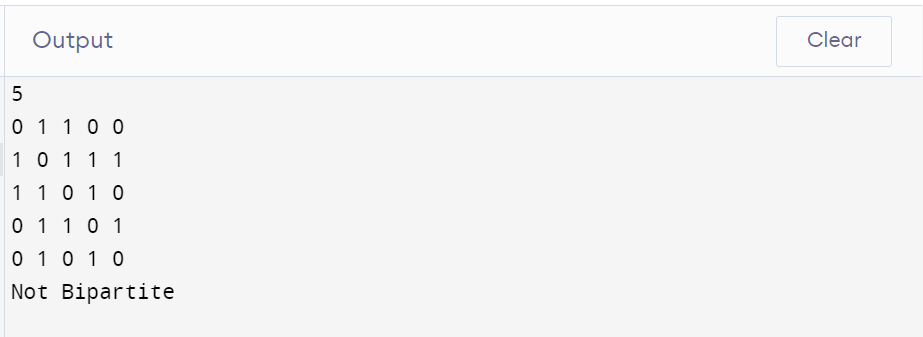
if (bipar(AdjM, v) == true)

cout << "Yes Bipartite";

else

cout << "Not Bipartite";

}



**/\*Ques3.Given a directed graph, design an algorithm and implement**

**it using a program to find whether cycle exists in the graph or not \*/**

#include <iostream>

#include <vector>

#include <stdbool.h>

using namespace std;

bool IsCycle(int src, vector<vector<int>> &adj, vector<bool> &stack, vector<bool> &visited)

{

stack[src] = true;

if (visited[src] == false)

{

for (auto j : adj[src])

{

if (visited[j] == false && IsCycle(j, adj, stack, visited))

return true;

if (stack[j] == true)

return true;

}

}

stack[src] = false;

return false;

}

int main()

{

int vertex = 0, edges;

cin >> edges;

vector<vector<int>> matrix;

vector<int> temp;

for (int i = 0; i < edges; i++)

{

for (int j = 0; j < edges; j++)

{

int t;

cin >> t;

temp.push\_back(t);

if (t == 1)

vertex++;

}

matrix.push\_back(temp);

temp.clear();

}

vector<vector<int>> adj(vertex);

for (int i = 0; i < edges; i++)

for (int j = 0; j < edges; j++)

if (matrix[i][j] == 1)

adj[i].push\_back(j);

bool cycle = false;

vector<bool> stack(edges, false);

vector<bool> visited(edges, false);

for (int i = 0; i < edges; i++)

{

if (visited[i] == false && IsCycle(i, adj, visited, stack))

{

cycle = true;

break;

}

}

if (cycle)

cout << "Yes Cycle Exixts.";

else

cout << "No Cycle Exits";

return 0;

}



**Week7.**

**/\*Ques1. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend’s location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra’s algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay’s results? (Hint: Print shortest path and distance from friends’ location to Akshay’s house)\*/**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int minDisIndex(int \*dis, bool \*vis, int v)

{

int i;

int minDis = INT\_MAX;

int minIndex = -1;

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

{

if (vis[i] == false && dis[i] <= minDis)

{

minDis = dis[i];

minIndex = i;

}

}

return minIndex;

}

void dijkstra(vector<vector<int>> mat, int v, int s)

{

int dis[v];

bool vis[v];

int parent[v];

int i, j;

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

{

dis[i] = INT\_MAX;

vis[i] = false;

parent[i] = -1;

}

dis[s] = 0;

parent[s] = s;

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

{

int m = minDisIndex(dis, vis, v);

vis[m] = true;

for (j = 0; j < v; j++)

{

if (dis[m] != INT\_MAX && !vis[j] && mat[m][j])

{

if (dis[j] > dis[m] + mat[m][j])

{

dis[j] = dis[m] + mat[m][j];

parent[j] = m;

}

}

}

}

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

{

if (i == s)

{

cout << i + 1 << " : " << dis[i] << endl;

continue;

}

cout << i + 1;

j = i;

while (parent[j] != s)

{

cout << " " << parent[j] + 1;

j = parent[j];

}

cout << " " << s + 1 << " : " << dis[i] << endl;

}

}

int main()

{

int i, j;

int v;

cin >> v;

vector<vector<int>> mat(v, vector<int>(v));

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

for (j = 0; j < v; j++)

cin >> mat[i][j];

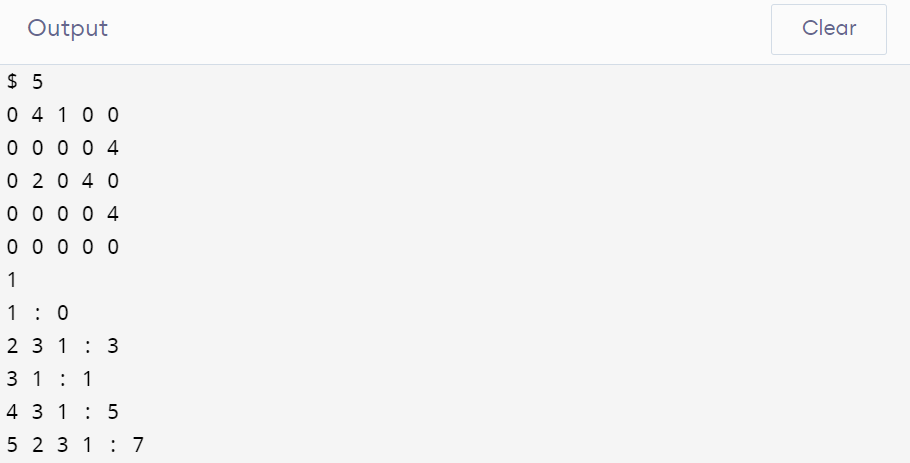
int s;

cin >> s;

dijkstra(mat, v, s - 1);

return 0;

}

****

**/\*Ques2.Design an algorithm and implement it using a program to solve previous question's problem using Bellman-Ford's shortest path algorithm.\*/**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

void bellmanFord(vector<vector<int>> mat, int v, int s)

{

int dis[v];

int parent[v];

int i, j;

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

{

dis[i] = INT\_MAX;

}

dis[s] = 0;

parent[s] = s;

for (int k = 0; k < v - 1; k++)

for (i = 0; i < v - 1; i++)

{

for (j = 0; j < v; j++)

{

if (dis[i] != INT\_MAX && mat[i][j])

{

if (dis[j] > dis[i] + mat[i][j])

{

dis[j] = dis[i] + mat[i][j];

parent[j] = i;

}

}

}

}

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

{

if (i == s)

{

cout << i + 1 << " : " << dis[i] << endl;

continue;

}

cout << i + 1;

j = i;

while (parent[j] != s)

{

cout << " " << parent[j] + 1;

j = parent[j];

}

cout << " " << s + 1 << " : " << dis[i] << endl;

}

}

int main()

{

int i, j;

int v;

cin >> v;

vector<vector<int>> mat(v, vector<int>(v));

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

for (j = 0; j < v; j++)

cin >> mat[i][j];

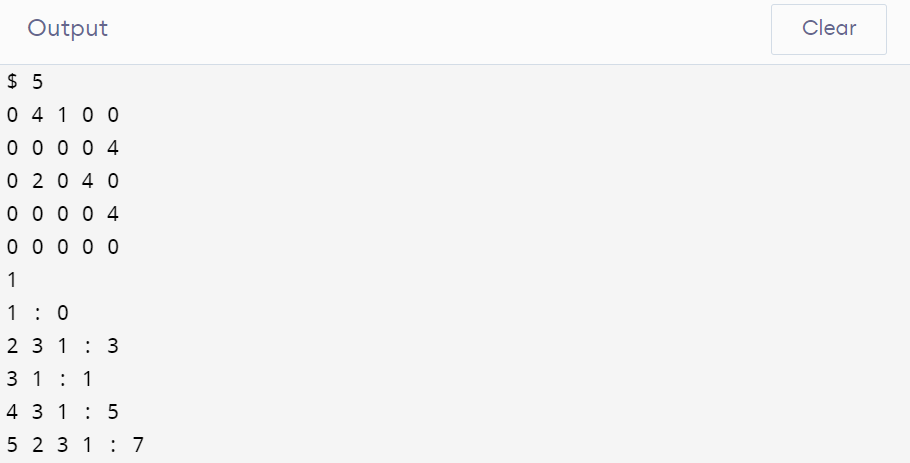
int s;

cin >> s;

bellmanFord(mat, v, s - 1);

return 0;

}



**/\*Ques3. Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.\*/**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

int min(int a, int b)

{

return a < b ? a : b;

}

int shortestPath(int V, int graph[][V], int u, int v, int k)

{

if (k == 0 && u == v)

return 0;

if (k == 1 && graph[u][v] != INT\_MAX)

return graph[u][v];

if (k <= 0)

return INT\_MAX;

int res = INT\_MAX;

for (int i = 0; i < V; i++)

{

if (graph[u][i] != INT\_MAX && u != i && v != i)

{

int rec\_res = shortestPath(V, graph, i, v, k - 1);

if (rec\_res != INT\_MAX)

res = min(res, graph[u][i] + rec\_res);

}

}

return res;

}

int main()

{

int V, src, des, k;

scanf("%d", &V);

int graph[V][V];

for (int i = 0; i < V; i++)

{

for (int j = 0; j < V; j++)

{

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

}

}

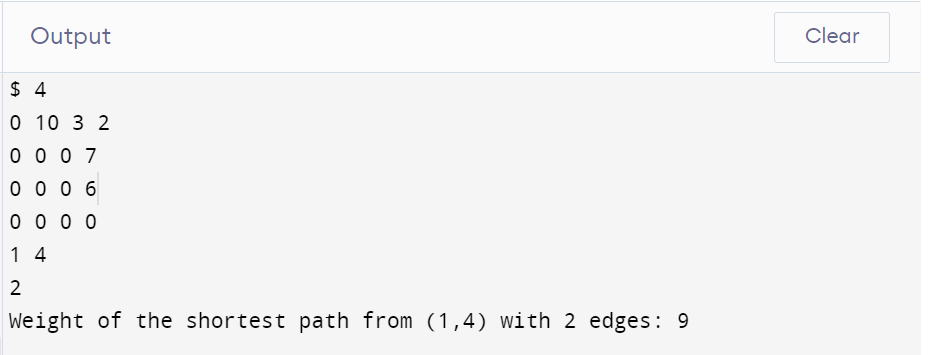
scanf("%d%d", &src, &des);

scanf("%d", &k);

printf("Weight of the shortest path from (%d,%d) with %d edges: %d", src, des, k, shortestPath(V, graph, src - 1, des - 1, k));

return 0;

}

****

**Week 8.**

**/\*Ques1. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)\*/**

#include <bits/stdc++.h>

using namespace std;

#define V 7

int minimumnodevertex(vector<int> weight, vector<bool> process)

{

int vertex;

int minimum = INT\_MAX;

for (int i = 0; i < V; i++)

{

if (minimum > weight[i] && process[i] == false)

{

minimum = weight[i];

vertex = i;

}

}

return vertex;

}

void findMST(int graph[V][V])

{

vector<int> parent(V, -1);

vector<int> weight(V, INT\_MAX);

vector<bool> process(V, false);

int MinimumWight = 0;

weight[0] = 0;

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

{

int minvertex = minimumnodevertex(weight, process);

process[minvertex] = true;

for (int j = 0; j < V; j++)

{

if (graph[minvertex][j] != 0 && process[j] != true && graph[minvertex][j] < weight[j])

{

weight[j] = graph[minvertex][j];

parent[j] = minvertex;

}

}

}

for (int i = 1; i < V; ++i)

MinimumWight += graph[parent[i]][i];

cout << "Minimum Spanning Weight :" << MinimumWight;

}

int main()

{

int graph[V][V];

for (int i = 0; i < V; i++)

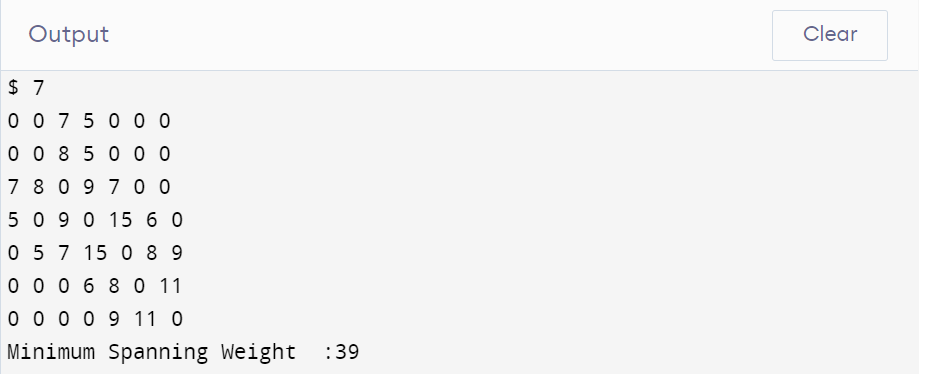
for (int j = 0; j < V; j++)

cin >> graph[i][j];

findMST(graph);

return 0;

}



**/\*Ques2.Implement the previous problem using Kruskal's algorithm.\*/**

#include <iostream>

#include <algorithm>

#include <vector>

#include <limits.h>

using namespace std;

vector<pair<int, pair<int, int>>> graph;

vector<pair<int, pair<int, int>>> result;

int parent[10000];

using namespace std;

void make(int i)

{

parent[i] = i;

}

int find(int V)

{

if (V == parent[V])

return V;

return find(parent[V]);

}

void Union(int a, int b, int i)

{

a = find(a);

b = find(b);

if (a != b)

{

parent[b] = a;

result.push\_back(graph[i]);

}

}

void Kruskal(int V)

{

sort(graph.begin(), graph.end());

int E = graph.size();

int s, d, w;

for (int i = 0; i < E; i++)

{

s = graph[i].second.first;

d = graph[i].second.second;

Union(s, d, i);

}

}

int main()

{

int V, E = 0;

cin >> V;

vector<vector<int>> matrix;

vector<int> temp;

for (int i = 0; i < V; i++)

{

for (int j = 0; j < V; j++)

{

int t;

cin >> t;

temp.push\_back(t);

if (t != 0)

E++;

}

matrix.push\_back(temp);

temp.clear();

}

for (int i = 0; i < V; i++)

make(i);

for (int i = 0; i < V; i++)

for (int j = 0; j < V; j++)

if (matrix[i][j] != 0)

graph.push\_back(make\_pair(matrix[i][j], make\_pair(i, j)));

int sum = 0;

Kruskal(V);

for (int i = 0; i < result.size(); i++)

{

int w = result[i].first;

sum = sum + w;

}

cout << "Minimum Spanning weight :" << sum;

return 0;

}



**/\*Ques3.Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.\*/**

#include <iostream>

#include <algorithm>

#include <vector>

#include <limits.h>

using namespace std;

vector<pair<int, pair<int, int>>> graph;

vector<pair<int, pair<int, int>>> result;

int parent[10000];

using namespace std;

void make(int i)

{

parent[i] = i;

}

int find(int V)

{

if (V == parent[V])

return V;

return find(parent[V]);

}

void Union(int a, int b, int i)

{

a = find(a);

b = find(b);

if (a != b)

{

parent[b] = a;

result.push\_back(graph[i]);

}

}

void Kruskal(int V)

{

sort(graph.rbegin(), graph.rend());

int E = graph.size();

int s, d, w;

for (int i = 0; i < E; i++)

{

s = graph[i].second.first;

d = graph[i].second.second;

Union(s, d, i);

}

}

int main()

{

int V, E = 0;

cin >> V;

vector<vector<int>> matrix;

vector<int> temp;

for (int i = 0; i < V; i++)

{

for (int j = 0; j < V; j++)

{

int t;

cin >> t;

temp.push\_back(t);

if (t != 0)

E++;

}

matrix.push\_back(temp);

temp.clear();

}

for (int i = 0; i < V; i++)

make(i);

for (int i = 0; i < V; i++)

for (int j = 0; j < V; j++)

if (matrix[i][j] != 0)

graph.push\_back(make\_pair(matrix[i][j], make\_pair(i, j)));

int sum = 0;

Kruskal(V);

for (int i = 0; i < result.size(); i++)

{

int w = result[i].first;

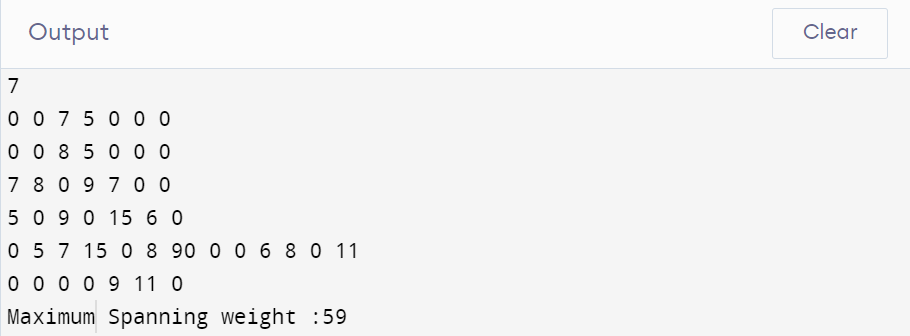
sum = sum + w;

}

cout << "Maximum Spanning weight :" << sum;

return 0;

}



**Week 9.**

**/\*Ques1. Given a graph, Design an algorithm and implement it using a program to implement Floyed-Warshall all pair shortest algorithm.\*/**

#include <bits/stdc++.h>

using namespace std;

#define V 5

#define INF 99999

void printSolution(int dist[][V])

{

cout << "Shortest Distance Matrix: \n";

for (int i = 0; i < V; i++)

{

for (int j = 0; j < V; j++)

{

if (dist[i][j] == INF)

cout << "INF"

<< " ";

else

cout << dist[i][j] << " ";

}

cout << endl;

}

}

void floydWarshall(int graph[][V])

{

int dist[V][V], i, j, k;

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

for (j = 0; j < V; j++)

dist[i][j] = graph[i][j];

for (k = 0; k < V; k++)

{

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

{

for (j = 0; j < V; j++)

{

if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF && dist[i][k] != INF))

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

printSolution(dist);

}

int main()

{

int graph[V][V] = {{0, 10, 5, 5, INF},

{INF, 0, 5, 5, 5},

{INF, INF, 0, INF, 10},

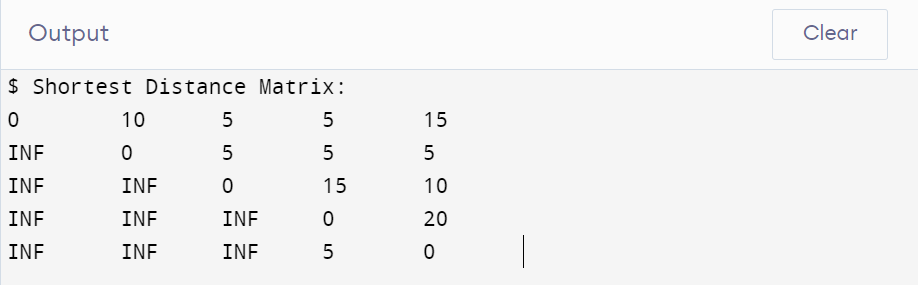
{INF, INF, INF, 0, 20},

{INF, INF, INF, 5, 0}};

floydWarshall(graph);

return 0;

}



**/\*Ques2.Given a knapsack of maximum capacity w . N items are provided , each having its own value and weight . You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value . You can take fractions of items , i.e . the items can be broken into smaller pieces so that you have to carry only a fraction x of item i , where 0<=x<=1\*/**

#include <iostream>

#include <vector>

#include <map>

using namespace std;

int main()

{

float weight, volume, capacity, PerUnit;

map<float, pair<float, int>, greater<float>> mymap;

int n;

cin >> n;

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

{

cin >> weight >> volume;

PerUnit = volume / weight;

mymap.insert({PerUnit, make\_pair(weight, i + 1)});

}

cin >> capacity;

float MaxProfit = 0, we = 0;

vector<pair<int, float>> item\_weight;

for (auto it = mymap.begin(); it != mymap.end(); it++)

{

MaxProfit += it->first \* it->second.first;

we += it->second.first;

if (we < capacity)

item\_weight.push\_back(make\_pair(it->second.second, it->second.first));

if (we > capacity)

{

float temp = 0;

MaxProfit -= it->first \* it->second.first;

we -= it->second.first;

temp = capacity - we;

MaxProfit += it->first \* temp;

we += temp;

item\_weight.push\_back(make\_pair(it->second.second, temp));

}

if (we == capacity)

break;

}

cout << "Maximum Value:" << MaxProfit << endl;

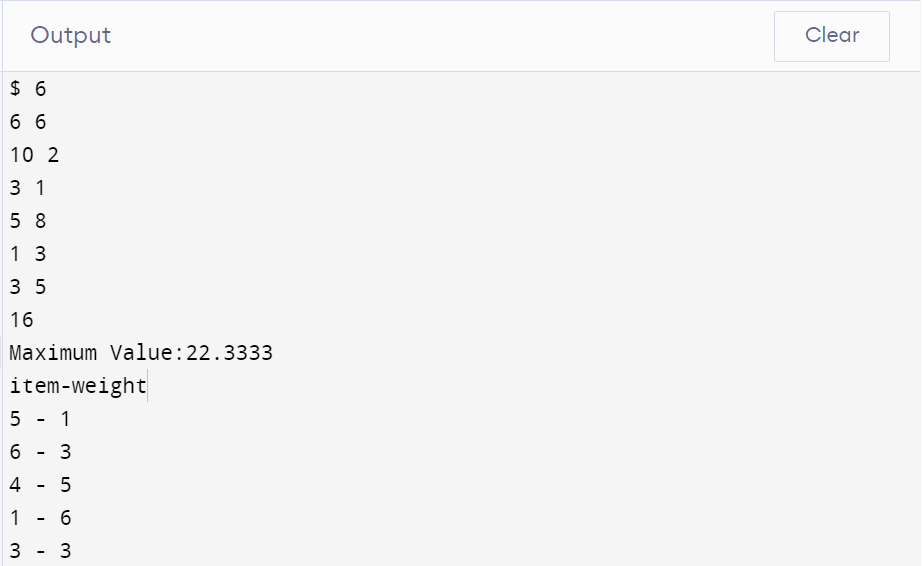
cout << "item-weight" << endl;

for (int i = 0; i < item\_weight.size(); i++)

cout << item\_weight[i].first << " - " << item\_weight[i].second << endl;

return 0;

}

****

**/\*Ques3. Given an array of elements . Assume arr [ i ] represents the size of file i . Write an algorithm and a program to merge all these files into single file with minimum computation . For given two files A and B with sizes m and n , computation cost of merging them is O ( m + n ) . ( Hint : use greedy approach )\*/**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

int n, sum = 0, MinimumCost = 0;

cin >> n;

vector<int> vrr(n);

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

cin >> vrr[i];

sort(vrr.begin(), vrr.end());

for (int i = 1; i < vrr.size(); i++)

{

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

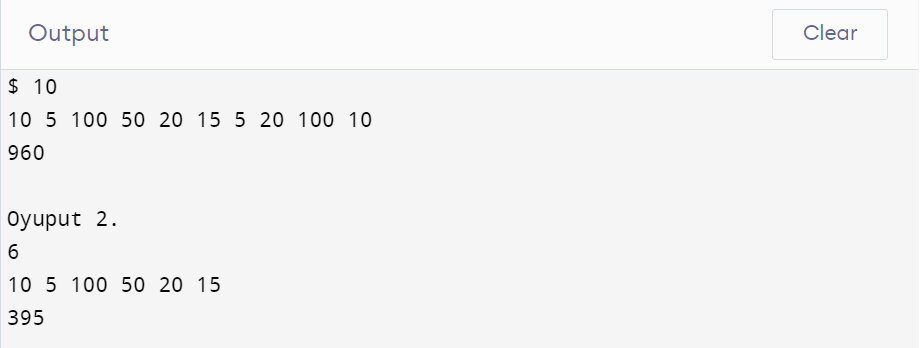
MinimumCost += vrr[i];

}

cout << MinimumCost;

return 0;

}



**Week 10.**

**/\*Ques1. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.\*/**

#include <iostream>

#include <vector>

#include <map>

#include <algorithm>

using namespace std;

int main()

{

int size;

cin >> size;

vector<int> result;

multimap<int, pair<int, int>> activity;

for (int i = 0; i < size; i++)

{

int s, e;

cin >> s >> e;

activity.insert({e, make\_pair(s, i + 1)});

}

int count = 0, next = -1;

for (auto it = activity.begin(); it != activity.end(); it++)

{

if (it->second.first > next)

{

count++;

next = it->first;

result.push\_back(it->second.second);

}

}

cout << "No. of non-conflicting activities: " << count << endl;

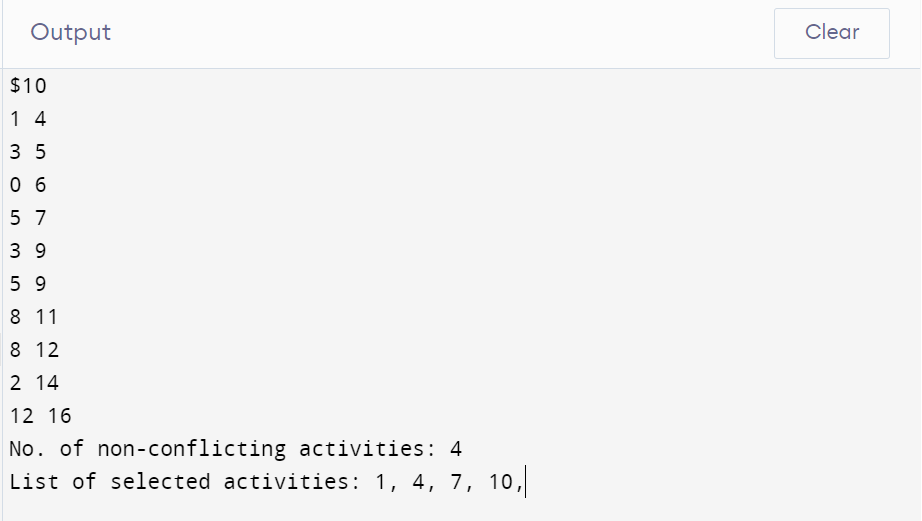
cout << "List of selected activities: ";

for (int i = 0; i < result.size(); i++)

cout << result[i] << ", ";

return 0;

}



**/\*Ques3. Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.\*/**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

int n;

cin >> n;

vector<int> vrr(n);

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

cin >> vrr[i];

int median = vrr[n / 2];

sort(vrr.begin(), vrr.end());

int count = 1, temp = 0;

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

{

if (vrr[i] == vrr[i + 1])

count++;

else

{

if (count >= (n / 2))

{

cout << "Yes";

temp = 1;

}

count = 1;

}

}

if (!temp)

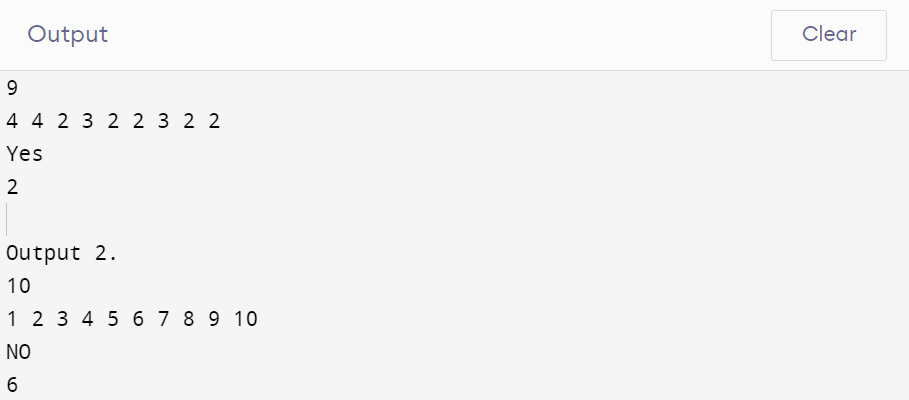
cout << "NO";

cout << "\n"

<< median;

return 0;

}



**Week 11.**

**/\*Ques1. Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied\*/**

#include <bits/stdc++.h>

using namespace std;

int MatrixChainOrder(int p[], int i, int j)

{

if (i == j)

return 0;

int k;

int min = INT\_MAX;

int count;

for (k = i; k < j; k++)

{

count = MatrixChainOrder(p, i, k) + MatrixChainOrder(p, k + 1, j) + p[i - 1] \* p[k] \* p[j];

if (count < min)

min = count;

}

return min;

}

int main()

{

int n;

cin >> n;

int arr[n];

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

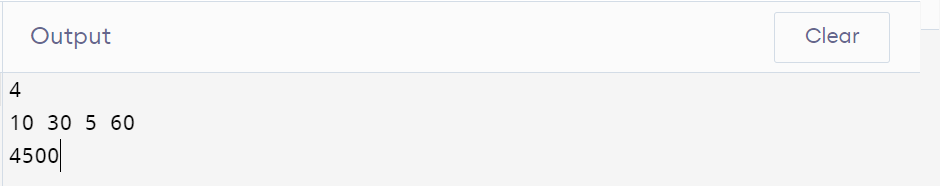
{

cin >> arr[i];

}

cout << "Minimum number of multiplications is " << MatrixChainOrder(arr, 1, n);

}



**/\*Ques2. Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.\*/**

#include <bits/stdc++.h>

using namespace std;

int count(int S[], int m, int n)

{

if (n == 0)

return 1;

if (n < 0)

return 0;

if (m <= 0 && n >= 1)

return 0;

return count(S, m - 1, n) +

count(S, m, n - S[m - 1]);

}

int main()

{

int n, m;

cin >> n;

int arr[n];

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

{

cin >> arr[i];

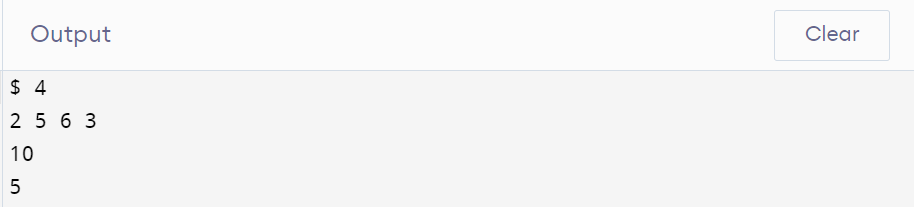
}

cin >> m;

cout << " " << count(arr, n, m);

return 0;

}



**/\*Ques3. Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.\*/**

#include <bits/stdc++.h>

using namespace std;

bool isSubsetSum(int arr[], int n, int sum)

{

if (sum == 0)

return true;

if (n == 0 && sum != 0)

return false;

if (arr[n - 1] > sum)

return isSubsetSum(arr, n - 1, sum);

return isSubsetSum(arr, n - 1, sum) || isSubsetSum(arr, n - 1, sum - arr[n - 1]);

}

bool findPartiion(int arr[], int n)

{

int sum = 0;

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

sum += arr[i];

if (sum % 2 != 0)

return false;

return isSubsetSum(arr, n, sum / 2);

}

int main()

{

int n;

cin >> n;

int arr[n];

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

{

cin >> arr[i];

}

if (findPartiion(arr, n) == true)

cout << "yes";

else

cout << "no";

return 0;

}

