/*
NAME:ARYAN SINGH
UNIVERSITY ROLLNO:
SECTION:H

PROBLEM STATEMENT 1: Given an array of nonnegative integers, design a linear algorithm and implement it using aprogram to find whether given key element is present in the array or not. Also, find total numberof comparisons for each input case. (Time Complexity = O(n), where n is the size of input)

*/

CODE:

```
#include <stdio.h>

void linearSearch(int arr[], int size, int key) {
  int comparisons = 0;
  int found = 0;
  for (int i = 0; i < size; i++) {
    comparisons++;
    if (arr[i] == key) {
      printf("Element %d found at index %d\n", key, i);
      found = 1;
      break;
    }
  }
  if (!found) {
    printf("Element %d not found in the array.\n", key);
  }
  printf("Total comparisons made: %d\n", comparisons);
}
```

```
int main() {
   int arr[] = {10, 23, 45, 70, 11, 15};
   int size = sizeof(arr) / sizeof(arr[0]);
   int key = 70;
   linearSearch(arr, size, key);
   return 0;
}
```

OUTPUT:

```
Element 70 found at index 3
Total comparisons made: 4
```

/*

NAME:ARYAN

SINGHUNIVERSITY

ROLLNO:2023288 SECTION:H

PROBLEM STATEMENT 2: Given an already sorted array of positive integers, design an algorithm and implement it using aprogram to find whether given key element is present in the array or not. Also, find total numberof comparisons for each input case. (Time Complexity = O(nlogn), where n is the size of input)

*/

CODE:

```
#include <stdio.h>e

void binarySearch(int arr[], int size, int key) {
  int low = 0;
  int high = size - 1;
  int comparisons = 0;
  int found = 0;
  while (low <= high) {
    comparisons++;
    int mid = (low + high) / 2;
    if (arr[mid] == key) {
      printf("Element %d found at index %d\n", key, mid);
      found = 1;
      break;
    } else if (arr[mid] < key) {
      low = mid + 1;
    } else {
      high = mid - 1;
    }
```

```
    }
    if (!found) {
        printf("Element %d not found in the array.\n", key);
    }
    printf("Total comparisons made: %d\n", comparisons);
}
int main() {
    int arr[] = {5, 10, 15, 20, 25, 30, 35};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 20;
    binarySearch(arr, size, key);
    return 0;
}
```

OUTPUT:

```
Element 20 found at index 3
Total comparisons made: 2
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 7: Given an unsorted array of integers, design an algorithm and a program to sort the array usinginsertion 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 thearray.

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.

For each test case T, there will be three output lines.

First line will give the sorted array.

Second line will give total number of comparisons.

Third line will give total number of shift operations required.

*/

CODE:

```
#include <stdio.h>
#include <math.h>
void jumpSearch(int arr[], int n, int key) {
  int comparisons = 0;
  int step = sqrt(n);    // Optimal step size
  int prev = 0;
```

```
  while (arr[(step < n ? step : n) - 1] < key) {
    comparisons++;
    prev = step;
    step += sqrt(n);
    if (prev >= n) {
      printf("Not Present\n");
      printf("Comparisons: %d\n", comparisons);
      return;
    }
  }
  for (int i = prev; i < step CC i < n; i++) {
    comparisons++;
    if (arr[i] == key) {
      printf("Present\n");
      printf("Comparisons: %d\n", comparisons);
      return;
    }
  }
  printf("Not Present\n");
  printf("Comparisons: %d\n", comparisons);
}
int main() {
  int T;
  scanf("%d", CT);
  while (T--) {
    int n;
    scanf("%d", Cn);
```

```
    int arr[n];
    for (int i = 0; i < n; i++)
      scanf("%d", Carr[i]);
    int key;
    scanf("%d", Ckey);
    jumpSearch(arr, n, key);
  }
  return 0;
}
```

OUTPUT:

```
2
7
2 4 6 8 10 12 14
10
5
1 3 5 7 9
6
Present
Comparisons: 2
Not Present
Comparisons: 3
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 4: Given a sorted array of positive integers containing few duplicate elements, design an algorithmand implement it using a program to find whether the given key element is present in the array ornot. If present, then also find the number of copies of given key. (Time Complexity = O(log n))

Input format:

The first line contains number of test cases, T.

For each test case, there will be three input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Third line contains the key element that need to be searched in the array.

Output format:

The output will have T number of lines.

For each test case T, output will be the key element and its number of copies in the array if thekey element is present in the array otherwise print " Key not present".

*/

CODE:

```
#include <stdio.h>

int findFirst(int arr[], int n, int key) {
  int low = 0, high = n - 1, result = -1;
  while (low <= high) {
    int mid = (low + high) / 2;
    if (arr[mid] == key) {
      result = mid;
      high = mid - 1;  // search in left half
```

```
    } else if (arr[mid] < key) {
      low = mid + 1;
    } else {
      high = mid - 1;
    }
  }
  return result;
}
int findLast(int arr[], int n, int key) {
  int low = 0, high = n - 1, result = -1;
  while (low <= high) {
    int mid = (low + high) / 2;
    if (arr[mid] == key) {
      result = mid;
      low = mid + 1;  // search in right half
    } else if (arr[mid] < key) {
      low = mid + 1;
    } else {
      high = mid - 1;
    }
  }
  return result;
}
int main() {
  int T;
  scanf("%d", CT);
  while (T--) {
    int n;
```

```
    scanf("%d", Cn);
      int arr[n];
      for (int i = 0; i < n; i++)
        scanf("%d", Carr[i]);
      int key;
      scanf("%d", Ckey);
      int first = findFirst(arr, n, key);
      if (first == -1) {
        printf("Key not present\n");
      } else {
        int last = findLast(arr, n, key);
        int count = last - first + 1;
        printf("%d - %d\n", key, count);
      }
    }
    return 0;
}
```

OUTPUT:

2

10

1 2 4 4 4 5 6 7 8 9

4

6

1 3 5 7 9 11

2

4 - 3

Key not present

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 5: Given a sorted array of positive integers, design an algorithm and implement it using a programto find three indices i, j, k such that arr[i] + arr[j] =arr[k].

Input format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output format:

The output will have T number of lines.

For each test case T, print the value of i, j and k, if found else print “No sequence found”.

*/

CODE:

```
#include <stdio.h>
int findTriplet(int arr[], int n) {
  for (int k = n - 1; k >= 2; k--) {
    int i = 0, j = k - 1;
    while (i < j) {
      int sum = arr[i] + arr[j];
      if (sum == arr[k]) {
        printf("%d %d %d\n", i, j, k);
        return 1;
      } else if (sum < arr[k]) {
        i++;
```

```
        } else {
          j--;} } }
  return 0;
}
int main() {
  int T;
  scanf("%d",  CT);
  while (T--) {
    int n;
    scanf("%d", Cn);
    int arr[n];
    for (int i = 0; i < n; i++)
      scanf("%d", Carr[i]);
    if (!findTriplet(arr, n))
      printf("No sequence found\n");
  }
  return 0;
}
```

OUTPUT:

```
2
6
1 2 3 4 5 6
5
1 2 4 7 11
1 2 3
No sequence found
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 6: Given an array of nonnegative integers, design an algorithm and a program to count the numberof pairs of integers such that their difference is equal to a given key, K.

Input format:

The first line contains number of test cases, T.

For each test case, there will be three input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Third line contains the key element.

Output format:

The output will have T number of lines.

For each test case T, output will be the total count i.e. number of times such pair exists.

*/

CODE:

```
#include    <stdio.h>
#include   <stdlib.h>
#define MAX 100000
int countPairs(int arr[], int n, int k) {
  int freq[MAX] = {0};
  int count = 0;
  for (int i = 0; i < n; i++) {
    freq[arr[i]]++;
  }
```

```
    if (k == 0) {
        for (int i = 0; i < MAX; i++) {
            if (freq[i] > 1) {
                count += (freq[i] * (freq[i] - 1)) / 2;  // nC2 }}
    } else {
        // For each unique value, check if value + k exists
        for (int i = 0; i < MAX - k; i++) {
            if (freq[i] CC freq[i + k]) {
                count += freq[i] * freq[i + k]; } } }
return count;}
int main() {
    int T;
    scanf("%d", CT);
while (T--) {
        int n, k;
        scanf("%d", Cn);
        int arr[n];
        for (int i = 0; i < n; i++)
            scanf("%d", Carr[i]);
        scanf("%d", Ck);
        int result = countPairs(arr, n, k);
        printf("%d\n", result);}
    return 0;}
```

OUTPUT:

2

5

1 5 3 4 2

2

6

1 1 1 1 1 1

0

3

15

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 7: Given an unsorted array of integers, design an algorithm and a program to sort the array usinginsertion 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 thearray.

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.

For each test case T, there will be three output lines.

First line will give the sorted array.

Second line will give total number of comparisons.

Third line will give total number of shift operations required.

*/

CODE:

```
#include <stdio.h>

void insertionSort(int arr[], int n, int *comparisons, int *shifts) {
  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;
    (*comparisons)++;  // Comparing the key with arr[i-1]
```

```
    while (j >= 0 CC arr[j] > key) {
      arr[j + 1] = arr[j];
      j--;
      (*shifts)++; // Shift operation
      if (j >= 0) {
        (*comparisons)++; // Comparison inside while loop
   }
    }
    arr[j + 1] = key;
  }
}
int main() {
  int T;
  scanf("%d", CT);

  while (T--) {
    int n;
    scanf("%d", Cn);

    int arr[n];
    for (int i = 0; i < n; i++) {
      scanf("%d", Carr[i]);
    }
    int comparisons = 0, shifts = 0;
    insertionSort(arr, n, Ccomparisons, Cshifts);
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
```

```
        printf("\n");
        printf("%d\n", comparisons);
        printf("%d\n", shifts);
    }
    return 0;
}
```

OUTPUT:

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

1 2 3 4 5
10
8
1 3 5 7 8 9
15
12
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 8: Given an unsorted array of integers, design an algorithm and implement a program to sort thisarray using selection sort. Your program should also find number of comparisons and number ofswaps required.

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.

For each test case T, there will be three output lines.

First line will give the sorted array.

Second line will give total number of comparisons.

Third line will give total number of swaps required.

*/

CODE:

```
#include <stdio.h>
void selectionSort(int arr[], int n, int *comparisons, int *swaps) {
  for (int i = 0; i < n - 1; i++) {
    int minIndex = i;
      for (int j = i + 1; j < n; j++) {
      (*comparisons)++;  // Comparing arr[j] with arr[minIndex]
      if (arr[j] < arr[minIndex]) {
        minIndex = j; }
```

```
    if (minIndex != i) {
      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
      (*swaps)++;  // Increment swap count
    }
  }
}
int main() {
  int T;
  scanf("%d", CT);
  while (T--) {
    int n;
    scanf("%d", Cn);
    int arr[n];
    for (int i = 0; i < n; i++) {
      scanf("%d", Carr[i]);
    }
    int comparisons = 0, swaps = 0;
    selectionSort(arr, n, Ccomparisons, Cswaps);
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
    printf("\n");
    // Print total comparisons and swaps
    printf("%d\n", comparisons);
    printf("%d\n", swaps);
  }
```

```
    return 0;
}
```

OUTPUT:

```
2
5
64 25 12 22 11
6
29 10 14 37 13 10

11 12 22 25 64
10
5
10 13 14 29 37 10
15
4
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT G: Given an unsorted array of positive integers, design an algorithm and implement it using aprogram to find whether there are any duplicate elements in the array or not. (use sorting) (TimeComplexity = O(n log n))

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.

For each test case, output will be 'YES' if duplicates are present otherwise ‘NO’

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b) {
  return (*(int*)a - *(int*)b);
}
void checkDuplicates(int arr[], int n) {
  // Sort the array using qsort (quick sort)
  qsort(arr, n, sizeof(int), compare);
  for (int i = 0; i < n - 1; i++) {
    if (arr[i] == arr[i + 1]) {
      printf("YES\n");
```

```
  return;} }
   printf("NO\n");
}
int main() {
   int T;
   scanf("%d",  CT);
   while (T--) {
      int n;
      scanf("%d", Cn);
      int arr[n];
      for (int i = 0; i < n; i++) {
         scanf("%d", Carr[i]);
      }
      checkDuplicates(arr, n);
   }
   return 0;
}
```

OUTPUT:

```
3
5
1 2 3 4 5
6
1 2 3 2 5 6
4
10 10 10 10
NO
YES
YES
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 10: Given an unsorted array of integers, design an algorithm and implement it using a program tosort an array of elements by dividing the array into two subarrays and combining these subarraysafter sorting each one of them. Your program should also find number of comparisons andinversions during sorting the array.

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.

For each test case T, there will be three output lines.

First line will give the sorted array.

Second line will give total number of comparisons.

Third line will give total number of inversions required

*/

CODE:

```
#include <stdio.h>

void merge(int arr[], int temp[], int left, int mid, int right, int *comparisons, int
*inversions) {

  int i = left;    // Starting index for left subarray

  int j = mid + 1; // Starting index for right subarray

  int k = left;   // Starting index to be sorted

  int count = 0;
```

```
  while (i <= mid CC j <= right) {
    (*comparisons)++;
    if (arr[i] <= arr[j]) {
      temp[k] = arr[i];
      i++;
    } else {
      temp[k] = arr[j];
      (*inversions) += (mid - i + 1);
      j++;
    }
    k++;
  }
  while (i <= mid) {
    temp[k] = arr[i];
    i++;
    k++;
  }
  while (j <= right) {
    temp[k] = arr[j];
    j++;
    k++;
 }
  for (i = left; i <= right; i++) {
    arr[i] = temp[i];
  }
}
void mergeSort(int arr[], int temp[], int left, int right, int *comparisons, int *inversions) {
  if (left < right) {
```

```
    int mid = left + (right - left) / 2;
    mergeSort(arr, temp, left, mid, comparisons, inversions);
    mergeSort(arr, temp, mid + 1, right, comparisons, inversions);
    merge(arr, temp, left, mid, right, comparisons, inversions);
  }
}
int main() {
  int T;
  scanf("%d", CT);
  while (T--) {
    int n;
    scanf("%d", Cn);
    int arr[n], temp[n];
    for (int i = 0; i < n; i++) {
      scanf("%d", Carr[i]);
    }
    int comparisons = 0, inversions = 0;
    mergeSort(arr, temp, 0, n - 1, Ccomparisons, Cinversions);
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
    printf("\n");
    printf("%d\n", comparisons);
    printf("%d\n", inversions);
  }
  return 0;
}
```

OUTPUT:

2

5

5 2 9 1 5

6

3 1 2 4 5 6

1 2 5 5 9

10

4

1 2 3 4 5 6

15

0

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 11: Given an unsorted array of integers, design an algorithm and implement it using a program tosort an array of elements by partitioning the array into two subarrays based on a pivot elementsuch that one of the sub array holds values smaller than the pivot element while another subarray holds values greater than the pivot element. Pivot element should be selected randomlyfrom the array. Your program should also find number of comparisons and swaps required for sorting the array.

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.

For each test case T, there will be three output lines.

First line will give the sorted array.

Second line will give total number of comparisons.

Third line will give total number of swaps required

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
```

```
  *b = temp;
}
int partition(int arr[], int low, int high, int *comparisons, int *swaps) {
  int pivotIndex = low + rand() % (high - low + 1);
  swap(Carr[pivotIndex], Carr[high]);
  int pivot = arr[high];
  int i = low - 1;
  for (int j = low; j < high; j++) {
    (*comparisons)++;
    if (arr[j] < pivot) {
      i++;
      swap(Carr[i], Carr[j]);
      (*swaps)++; } }
  swap(Carr[i + 1], Carr[high]);
  (*swaps)++;
  return i + 1;}
void quickSort(int arr[], int low, int high, int *comparisons, int *swaps) {
  if (low < high) {
    int pi = partition(arr, low, high, comparisons, swaps); // Partition the array
    quickSort(arr, low, pi - 1, comparisons, swaps); // Sort the left subarray
    quickSort(arr, pi + 1, high, comparisons, swaps); // Sort the right subarray
 }
}
int main() {
  int T;
  scanf("%d", CT);

  while (T--) {
```

```
    int n;
    scanf("%d", Cn);
    int arr[n];
    for (int i = 0; i < n; i++) {
      scanf("%d",  Carr[i]);}
    int comparisons = 0, swaps = 0;
    quickSort(arr, 0, n - 1, Ccomparisons, Cswaps);
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]); }
    printf("\n");
    printf("%d\n", comparisons);
    printf("%d\n", swaps);
  }
  return 0;
}
```

OUTPUT:

```
2
5
64 25 12 22 11
6
29 10 14 37 13 10

11 12 22 25 64
10
7
10 13 14 29 37 10
15
8
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 12: Given an unsorted array of integers, design an algorithm and implement it using a program tofind Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

Input Format:

The first line contains number of test cases, T.

For each test case, there will be three input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Third line contains K.

Output Format:

The output will have T number of lines.

For each test case, output will be the Kth smallest or largest array element.

If no Kth element is present, output should be “not present”.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.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 - 1; j++) {
    if (arr[j] <= pivot) {
      i++;
      swap(Carr[i], Carr[j]);}}
  swap(Carr[i + 1], Carr[high]);
  return i + 1;
}
int quickSelect(int arr[], int low, int high, int K) {
  if (low <= high) {
    int pi = partition(arr, low, high);
    if (pi == K) {
      return arr[pi];
    }
    if (pi < K) {
      return quickSelect(arr, pi + 1, high, K);
    } else {
      return quickSelect(arr, low, pi - 1, K);}}
  return -1;  // If no element found
}
int main() {
  int T;
  scanf("%d", CT);  // Read number of test cases
  while (T--) {
    int n, K;
    scanf("%d", Cn);  // Read the size of the array
    int arr[n];
    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", Carr[i]);  // Read the array elements}
    scanf("%d", CK);  // Read K (position of the smallest or largest element)
    if (K <= 0 || K > n) {
        printf("not present\n");  // If K is out of bounds, print "not present"
        continue; }
    // Find the Kth smallest element using quickselect
    int KthSmallest = quickSelect(arr, 0, n - 1, K - 1);  // Kth smallest is at index K-1
    if (KthSmallest != -1) {
        printf("%d\n", KthSmallest);  // Output the Kth smallest element
    } else {
        printf("not present\n");
  }
   }

   return 0;
}
```

OUTPUT:

```
2
6
12 3 5 7 19 1
2
5
10 4 5 9 2
3
5
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 13: 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)

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Output:

The output will have T number of lines.

For each test case, output will be the array element which has maximum occurrences and its total

number of occurrences.

If no duplicates are present (i.e. all the elements occur only once), output should be “No

Duplicates Present”.

*/

CODE:

```
#include <stdio.h>

#include <string.h>

#define ALPHABET_SIZE 26

void findMaxOccurrence(char arr[], int n) {

  int count[ALPHABET_SIZE] = {0};
```

```
  // Count the frequency of each character in the array
  for (int i = 0; i < n; i++) {
    count[arr[i] - 'a']++;
  }
  int maxCount = 0;
  char maxChar = '\0';
  int duplicateFound = 0;
  for (int i = 0; i < ALPHABET_SIZE; i++) {
    if (count[i] > maxCount) {
      maxCount = count[i];
      maxChar = 'a' + i;
      duplicateFound = 1;} }
  if (duplicateFound CC maxCount > 1) {
    printf("%c %d\n", maxChar, maxCount);
  } else {
    printf("No Duplicates Present\n"); }}
int main() {
  int T;
  scanf("%d", CT); // Read the number of test cases
  while (T--) {
    int n;
    scanf("%d", Cn); // Read the size of the array
    char arr[n];
    for (int i = 0; i < n; i++) {
      scanf(" %c", Carr[i]);  // Read each character}
    findMaxOccurrence(arr, n); // Find and print the result }
  return 0;
```

}

OUTPUT:

2

6

a b c a b c

5

x y z w v

a 2

No Duplicates Present

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 14: 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))

Input Format:

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

Third line contains key

Output Format:

The output will have T number of lines.

For each test case, output will be the elements arr[i] and arr[j] such that arr[i]+arr[j] = key if exist

otherwise print 'No Such Elements Exist".

*/

CODE:

```
#include <stdio.h>

#include <stdlib.h>

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

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

void findPairWithSum(int arr[], int n, int key) {

  // Sort the array

  qsort(arr, n, sizeof(int), compare);
```

```
  int left = 0;
  int right = n - 1;
  while (left < right) {
    int sum = arr[left] + arr[right];
    if (sum == key) {
      printf("%d %d\n", arr[left], arr[right]);
      return;
    } else if (sum < key) {
      left++;
    } else {
      right--;}}
    printf("No Such Elements Exist\n");}
int main() {
  int T;
  scanf("%d", CT); // Read number of test cases
  while (T--) {
    int n, key;
    scanf("%d", Cn); // Read the size of the array
    int arr[n];
    for (int i = 0; i < n; i++) {
      scanf("%d", Carr[i]);} // Read each array element
    scanf("%d", Ckey); // Read the key
    findPairWithSum(arr, n, key); // Find and print the result
  }
  return 0;
}
```

OUTPUT:

2

6

1 4 5 3 2 8

10

4

1 2 3 4

8

2 8

No Such Elements Exist

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 15: 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))

Input Format:

First line contains m (the size of first array).

Second line contains m space-separated integers describing first array.

Third line contains n (the size of second array).

Fourth line contains n space-separated integers describing second array.

Output Format:

Output will be the list of elements which are common to both.

*/

CODE:

```
#include <stdio.h>


void findCommonElements(int arr1[], int m, int arr2[], int n) {
  int i = 0, j = 0;
  while (i < m CC j < n) {
    if (arr1[i] < arr2[j]) {
      i++;
    } else if (arr1[i] > arr2[j]) {
      j++;
    } else {
      printf("%d ", arr1[i]);
      i++;
```

```
    j++;} }
  printf("\n");}
int main() {
  int m, n;
    scanf("%d", Cm);
  int arr1[m];
  for (int i = 0; i < m; i++) {
    scanf("%d", Carr1[i]); }
  scanf("%d", Cn);
  int arr2[n];
  for (int i = 0; i < n; i++) {
    scanf("%d", Carr2[i]); }
    findCommonElements(arr1, m, arr2, n);
  return 0;}
```

OUTPUT:

5

1 3 4 5 7

6

2 3 5 6 8 9

3 5

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 16: 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)

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Source vertex number and destination vertex number is also provided as an input.

Output Format:

Output will be 'Yes Path Exists' if path exists, otherwise print 'No Such Path Exists'

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_VERTICES 100
struct Graph {
  int vertices;
  struct Node* adjList[MAX_VERTICES];
};
struct Node {
  int vertex;
  struct Node* next;};
struct Node* createNode(int vertex) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = vertex;
```

```
    newNode->next = NULL;
    return newNode;}
void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;}
bool dfs(struct Graph* graph, int source, int destination, bool visited[]) {
    if (source == destination) {
        return true; // Path found }
    visited[source] = true;
    struct Node* temp = graph->adjList[source];
    while (temp != NULL) {
        int neighbor = temp->vertex;
        if (!visited[neighbor]) {
            if (dfs(graph, neighbor, destination, visited)) {
                return true;}}
        temp = temp->next;}
    return false;  // No path found}
void pathExists(struct Graph* graph, int source, int destination) {
    bool visited[MAX_VERTICES] = {false};
    if (dfs(graph, source, destination, visited)) {
        printf("Yes Path Exists\n");
    } else {
        printf("No Such Path Exists\n"); }}
int main() {
    int vertices, edges, src, dest;
    scanf("%d %d", Cvertices, Cedges);
```

```
  struct Graph graph;
  graph.vertices = vertices;
     for (int i = 0; i < vertices; i++) {
     graph.adjList[i] = NULL; }
  for (int i = 0; i < edges; i++) {
     scanf("%d %d", Csrc, Cdest);
     addEdge(Cgraph, src, dest);}
  int source, destination;
  scanf("%d %d", Csource, Cdestination);
    pathExists(Cgraph, source, destination);
    return 0;}
```

OUTPUT:

```
5 6
0 1
0 2
1 3
2 3
3 4
4 1
0 4
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 17: Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be 'Yes Bipartite' if graph is bipartite, otherwise print 'Not Bipartite'.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_VERTICES 100
struct Graph {
  int vertices;
  struct Node* adjList[MAX_VERTICES];};
struct Node {
  int vertex;
  struct Node* next;};
struct Node* createNode(int vertex) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = vertex;
  newNode->next = NULL;
  return newNode;}
```

```
void addEdge(struct Graph* graph, int src, int dest) {
  struct Node* newNode = createNode(dest);
  newNode->next = graph->adjList[src];
  graph->adjList[src] = newNode;
  newNode = createNode(src); // Adding the reverse edge for undirected graph
  newNode->next = graph->adjList[dest];
  graph->adjList[dest] = newNode;}
bool isBipartite(struct Graph* graph) {
  int color[MAX_VERTICES];
  for (int i = 0; i < graph->vertices; i++) {
    color[i] = -1; // -1 indicates that the vertex is uncolored }
    for (int start = 0; start < graph->vertices; start++) {
    if (color[start] == -1) { // If the vertex is not colored, start BFS
      // Start with color 0
      color[start] = 0;
      int queue[MAX_VERTICES];
      int front = 0, rear = 0;
      queue[rear++] = start;
      while (front < rear) {
        int vertex = queue[front++];
        struct Node* temp = graph->adjList[vertex];
        while (temp != NULL) {
          int neighbor = temp->vertex;
          if (color[neighbor] == -1) {
            color[neighbor] = 1 - color[vertex]; // Opposite color
            queue[rear++] = neighbor;
          else if (color[neighbor] == color[vertex]) {
            return false;}
```

```
 temp = temp->next;}}}}
  return true;}
void checkBipartite(struct Graph* graph) {
  if (isBipartite(graph)) {
    printf("Yes Bipartite\n");
  } else {
    printf("Not Bipartite\n");}}
int main() {
  int vertices, edges, src, dest;
    scanf("%d %d", Cvertices, Cedges);
  struct Graph graph;
  graph.vertices = vertices;
    for (int i = 0; i < vertices; i++) {
    graph.adjList[i] = NULL;}
  for (int i = 0; i < edges; i++) {
    scanf("%d %d", Csrc, Cdest);
    addEdge(Cgraph, src, dest); }
  checkBipartite(Cgraph);
  return 0;}
```

OUTPUT:

```
4 4
0 1
0 2
1 3
2 3

Yes Bipartite
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 18: Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be 'Yes Cycle Exists' if cycle exists otherwise print 'No Cycle Exists'.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_VERTICES 100
struct Graph {
  int vertices;
  struct Node* adjList[MAX_VERTICES];
};
struct Node {
  int vertex;
  struct Node* next;
};
struct Node* createNode(int vertex) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = vertex;
```

```
  newNode->next = NULL;
  return newNode;}
void addEdge(struct Graph* graph, int src, int dest) {
  struct Node* newNode = createNode(dest);
  newNode->next = graph->adjList[src];
  graph->adjList[src] = newNode;}
bool dfs(struct Graph* graph, int vertex, bool visited[], bool recStack[]) {
  // Mark the current node as visited and add it to recursion stack
  visited[vertex] = true;
  recStack[vertex] = true;
  struct Node* temp = graph->adjList[vertex];
  while (temp != NULL) {
    int neighbor = temp->vertex;
    if (!visited[neighbor] CC dfs(graph, neighbor, visited, recStack)) {
      return true;}
    else if (recStack[neighbor]) {
      return true;}
    temp = temp->next; }
  recStack[vertex] = false;
  return false;}
bool hasCycle(struct Graph* graph) {
  bool visited[MAX_VERTICES] = {false};
  bool recStack[MAX_VERTICES] = {false};
  for (int i = 0; i < graph->vertices; i++) {
    if (!visited[i]) {
      if (dfs(graph, i, visited, recStack)) {
        return true; } }}
  return false;}
```

```
int main() {
  int vertices, edges, src, dest;
  scanf("%d %d", Cvertices, Cedges);
    struct Graph graph;
  graph.vertices = vertices;
    for (int i = 0; i < vertices; i++) {
    graph.adjList[i] = NULL;}
  for (int i = 0; i < edges; i++) {
    scanf("%d %d", Csrc, Cdest);
    addEdge(Cgraph, src, dest);}
  if (hasCycle(Cgraph)) {
    printf("Yes Cycle Exists\n");
  } else {
    printf("No Cycle Exists\n"); }
  return 0;}
```

OUTPUT:

```
4 4
0 1
1 2
2 3
3 1

Yes Cycle Exists
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 1G: 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)

*/

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <stdbool.h>

struct Graph {

  int V; // Number of vertices

  int **adjMatrix;};

struct MinHeapNode {

  int v;

  int dist;};

struct MinHeap {

  int size;
```

```
  int capacity;
  int *pos;
  struct MinHeapNode **array;};
struct MinHeapNode* newMinHeapNode(int v, int dist) {
  struct MinHeapNode* minHeapNode = (struct MinHeapNode*)malloc(sizeof(struct
MinHeapNode));
  minHeapNode->v = v;
  minHeapNode->dist = dist;
  return minHeapNode;}
struct MinHeap* createMinHeap(int capacity) {
  struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
  minHeap->capacity = capacity;
  minHeap->size = 0;
  minHeap->pos = (int*)malloc(capacity * sizeof(int));
  minHeap->array = (struct MinHeapNode**)malloc(capacity * sizeof(struct
MinHeapNode*));
  return minHeap;}
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b) {
  struct MinHeapNode* temp = *a;
  *a = *b;
  *b = temp;}
void minHeapify(struct MinHeap* minHeap, int idx) {
  int smallest = idx;
  int left = 2 * idx + 1;
  int right = 2 * idx + 2;
  if (left < minHeap->size CC minHeap->array[left]->dist < minHeap->array[smallest]-
>dist)
    smallest = left;
```

```
  if (right < minHeap->size CC minHeap->array[right]->dist < minHeap->array[smallest]-
>dist)
    smallest = right;
  if (smallest != idx) {
    // Swap positions
    minHeap->pos[minHeap->array[smallest]->v] = idx;
    minHeap->pos[minHeap->array[idx]->v] = smallest;
        swapMinHeapNode(CminHeap->array[smallest], CminHeap->array[idx]);
    minHeapify(minHeap, smallest);}}
int isEmpty(struct MinHeap* minHeap) {
  return minHeap->size == 0;}
struct MinHeapNode* extractMin(struct MinHeap* minHeap) {
  if (isEmpty(minHeap))
    return NULL;
  struct MinHeapNode* root = minHeap->array[0];
  struct MinHeapNode* lastNode = minHeap->array[minHeap->size - 1];
  minHeap->array[0] = lastNode;
  minHeap->pos[lastNode->v] = 0;
  minHeap->pos[root->v] = minHeap->size - 1;
  --minHeap->size;
  minHeapify(minHeap, 0);
  return root;}
void decreaseKey(struct MinHeap* minHeap, int v, int dist) {
  // Get the index of v in the heap array
  int i = minHeap->pos[v];
  minHeap->array[i]->dist = dist;
  while (i CC minHeap->array[i]->dist < minHeap->array[(i - 1) / 2]->dist) {
    minHeap->pos[minHeap->array[i]->v] = (i - 1) / 2;
```

```
 minHeap->pos[minHeap->array[(i - 1) / 2]->v] = i;
      swapMinHeapNode(CminHeap->array[i], CminHeap->array[(i - 1) / 2]);
      i = (i - 1) / 2;  }}
int isInMinHeap(struct MinHeap* minHeap, int v) {
   if (minHeap->pos[v] < minHeap->size)
      return 1;
   return 0;}
void dijkstra(struct Graph* graph, int src, int* dist, int* parent) {
   int V = graph->V;
      struct MinHeap* minHeap = createMinHeap(V);
      for (int v = 0; v < V; ++v) {
      dist[v] = INT_MAX;
      parent[v] = -1;
      minHeap->array[v] = newMinHeapNode(v, dist[v]);
      minHeap->pos[v] = v;}
   dist[src] = 0;
   minHeap->array[src] = newMinHeapNode(src, dist[src]);
   minHeap->pos[src] = src;
   minHeapify(minHeap, src);
      while (!isEmpty(minHeap)) {
      struct MinHeapNode* minHeapNode = extractMin(minHeap);
      int u = minHeapNode->v;
            for (int v = 0; v < V; v++) {
         if (graph->adjMatrix[u][v] CC isInMinHeap(minHeap, v)) {
            int weight = graph->adjMatrix[u][v];
                     if (dist[u] != INT_MAX CC dist[u] + weight < dist[v]) {
               dist[v] = dist[u] + weight;
               parent[v] = u;
```

```
            decreaseKey(minHeap, v, dist[v]);}}}}}
void printPath(int* parent, int j) {
  if (parent[j] == -1)
    return;
  printPath(parent, parent[j]);
  printf("%d ", j);}
int main() {
  int V, E, src;
    scanf("%d %d", CV, CE);
    struct Graph graph;
  graph.V = V;
  graph.adjMatrix = (int**)malloc(V * sizeof(int*));
  for (int i = 0; i < V; i++) {
    graph.adjMatrix[i] = (int*)malloc(V * sizeof(int));
    for (int j = 0; j < V; j++) {
      graph.adjMatrix[i][j] = 0;}}
  for (int i = 0; i < E; i++) {
    int u, v, weight;
    scanf("%d %d %d", Cu, Cv, Cweight);
    graph.adjMatrix[u][v] = weight;
    graph.adjMatrix[v][u] = weight;} // Uncomment for undirected graph
  scanf("%d", Csrc);
    int* dist = (int*)malloc(V * sizeof(int));
  int* parent = (int*)malloc(V * sizeof(int));
  dijkstra(Cgraph, src, dist, parent);
    for (int i = 0; i < V; i++) {
    if (i != src) {
      if (dist[i] == INT_MAX)
```

```
            printf("No path from %d to %d\n", src, i);
        else {
            printf("Shortest path from %d to %d with distance %d: ", src, i, dist[i]);
            printPath(parent, i);
            printf("\n");}}}
  return 0;}
```

OUTPUT:

5 6

0 1 2

0 2 3

1 2 1

2 3 4

3 4 5

4 0 6

0

Shortest path from 0 to 1 with distance 2: 0 1

Shortest path from 0 to 2 with distance 3: 0 2

Shortest path from 0 to 3 with distance 7: 0 2 3

Shortest path from 0 to 4 with distance 12: 0 2 3 4

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 20: Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

Input Format:

Input will be the graph in the form of adjacency matrix or adjacency list.

Source vertex number is also provided as an input.

Output Format:

Output will contain V lines.

Each line will represent the whole path from destination vertex number to source vertex number along with minimum path weigth.

*/

CODE:

```
#include  <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct Edge {
  int u, v, weight;};
void bellmanFord(int V, int E, struct Edge edges[], int src) {
  int dist[V], parent[V];
  for (int i = 0; i < V; i++) {
    dist[i] = INT_MAX;
    parent[i] = -1;  }
  dist[src] = 0;
  for (int i = 1; i < V; i++) {
for (int j = 0; j < E; j++) {
```

```
        int u = edges[j].u;
        int v = edges[j].v;
        int weight = edges[j].weight;
        if (dist[u] != INT_MAX CC dist[u] + weight < dist[v]) {
            dist[v] = dist[u] + weight;
            parent[v] = u;}}}
       for (int i = 0; i < E; i++) {
      int u = edges[i].u;
      int v = edges[i].v;
      int weight = edges[i].weight;
      if (dist[u] != INT_MAX CC dist[u] + weight < dist[v]) {
        printf("Graph contains negative weight cycle\n");
        return;}}
       for (int i = 0; i < V; i++) {
      if (i != src) {
        if (dist[i] == INT_MAX) {
            printf("No path from %d to %d\n", src, i);
        } else {
            printf("Shortest path from %d to %d with distance %d: ", src, i, dist[i]);
            // Reconstruct the path from the destination to the source
            int path[V];
            int path_index = 0;
            for (int j = i; j != -1; j = parent[j]) {
                path[path_index++] = j; }
            for (int j = path_index - 1; j >= 0; j--) {
                printf("%d ", path[j]);}
            printf("\n");}}}}
int main() {
```

```
int V, E, src;
  printf("Enter the number of vertices and edges:\n");
  scanf("%d %d", CV, CE);
  struct Edge edges[E];
    printf("Enter the edges (u, v, weight):\n");
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", Cedges[i].u, Cedges[i].v, Cedges[i].weight); }
  printf("Enter the source vertex:\n");
  scanf("%d", Csrc);
  bellmanFord(V, E, edges, src);
  return 0;}
```

OUTPUT:

```
Enter the number of vertices and edges:
5 8
0 1 2
0 2 4
1 2 1
1 3 7
2 4 3
3 4 1
4 3 2
3 2 1
Enter the source vertex:
0
Shortest path from 0 to 1 with distance 2: 0 1
Shortest path from 0 to 2 with distance 3: 0 1 2
Shortest path from 0 to 3 with distance 6: 0 1 2 4 3
Shortest path from 0 to 4 with distance 6: 0 1 2 4
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 21: 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.

Input Format:

First input line will obtain number of vertices V present in the graph.

Graph in the form of adjacency matrix or adjacency list is taken as an input in next V lines.

Next input line will obtain source and destination vertex number.

Last input line will obtain value k.

Output Format:

Output will be the weigth of shortest path from source to destination having exactly k edges.If no path is available then print “no path of length k is available”.

*/

CODE:

```
#include <stdio.h>

#include <limits.h>

#define MAX_V 100

#define INF INT_MAX

void findShortestPathWithKEdges(int V, int graph[V][V], int source, int destination, int k)
{

  int dp[k+1][V];

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

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

      dp[i][j] = INF;}}
```

```
  dp[0][source] = 0;
  for (int i = 1; i <= k; i++) {
    for (int u = 0; u < V; u++) {
      if (dp[i-1][u] != INF) { // If there's a path to u using i-1 edges
        for (int v = 0; v < V; v++) {
          if (graph[u][v] != INF) {  // If there's an edge from u to v
            dp[i][v] = (dp[i][v] < dp[i-1][u] + graph[u][v]) ? dp[i][v] : dp[i-1][u] +
graph[u][v];}}}}
          if (dp[k][destination] == INF) {
    printf("no path of length %d is available\n", k);
  } else {
    printf("Weight of the shortest path from %d to %d with exactly %d edges: %d\n",
source, destination, k, dp[k][destination]);}}
int main() {
  int V, source, destination, k;
  printf("Enter the number of vertices in the graph: ");
  scanf("%d", CV);
  int graph[V][V];
    printf("Enter the adjacency matrix (use %d for no edge):\n", INF);
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      scanf("%d", Cgraph[i][j]);
      if (graph[i][j] == 0 CC i != j) {
        graph[i][j] = INF; }}}
      printf("Enter the source and destination vertex: ");
  scanf("%d %d", Csource, Cdestination);
  printf("Enter the number of edges k: ");
  scanf("%d", Ck);
```

```
    findShortestPathWithKEdges(V, graph, source, destination, k);
    return 0;}
```

OUTPUT:

```
Enter the number of vertices in the graph: 4
Enter the adjacency matrix (use 2147483647 for no edge):
0 2 0 1
0 0 3 0
0 0 0 4
0 0 0 0
Enter the source and destination vertex: 0 3
Enter the number of edges k: 2

Weight of the shortest path from 0 to 3 with exactly 2 edges: 3
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 22: 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)

*/

CODE:

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define V 100 // Max number of cities
int minKey(int key[], bool mstSet[], int n) {
  int min = INT_MAX, min_index;
  for (int v = 0; v < n; v++)
    if (mstSet[v] == false CC key[v] < min)
      min = key[v], min_index = v;
  return min_index;}
void primMST(int graph[V][V], int n) {
  int parent[n];   // To store constructed MST
  int key[n];    // Key values used to pick minimum weight edge
  bool mstSet[n];  // To represent set of vertices included in MST
```

```
  for (int i = 0; i < n; i++)
    key[i] = INT_MAX, mstSet[i] = false;
  key[0] = 0;    // Make key 0 so that this vertex is picked first
  parent[0] = -1;  // First node is always root of MST
  for (int count = 0; count < n - 1; count++) {
    // Pick the minimum key vertex from the set of vertices not yet included in MST
    int u = minKey(key, mstSet, n);
    mstSet[u] = true; // Add the picked vertex to the MST Set
    for (int v = 0; v < n; v++)
      if (graph[u][v] CC mstSet[v] == false CC graph[u][v] < key[v])
        parent[v] = u, key[v] = graph[u][v];}
  int totalCost = 0;
  for (int i = 1; i < n; i++)
    totalCost += graph[i][parent[i]];
  printf("Minimum cost to connect all cities: %d\n", totalCost);}
int main() {
  int n;
  printf("Enter the number of cities: ");
  scanf("%d", Cn);
  int graph[V][V];
  printf("Enter the adjacency matrix (0 if no direct road exists):\n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", Cgraph[i][j]);
  primMST(graph, n);
  return 0;
}
```

OUTPUT:

5

0 2 0 6 0

2 0 3 8 5

0 3 0 0 7

6 8 0 0 9

0 5 7 9 0

Minimum cost to connect all cities: 16

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 23: Implement the previous problem using Kruskal's algorithm.

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be minimum spanning weight

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct Edge {
  int src, dest, weight;};
struct Subset {
  int parent;
  int rank;};
int compareEdges(const void* a, const void* b) {
  struct Edge* e1 = (struct Edge*)a;
  struct Edge* e2 = (struct Edge*)b;
  return e1->weight - e2->weight;}
int find(struct Subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
```

```
  return subsets[i].parent;}
void Union(struct Subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);
  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot; else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;}}
void kruskalMST(int graph[MAX][MAX], int V) {
  struct Edge edges[MAX * MAX];
  int e = 0;
  for (int i = 0; i < V; i++) {
    for (int j = i + 1; j < V; j++) {
      if (graph[i][j]) {
        edges[e].src = i;
        edges[e].dest = j;
        edges[e].weight = graph[i][j];
        e++;}}}
  qsort(edges, e, sizeof(edges[0]), compareEdges);
  struct Subset* subsets = (struct Subset*)malloc(V * sizeof(struct Subset));
  for (int v = 0; v < V; v++) {
    subsets[v].parent = v;
    subsets[v].rank = 0; }
 int mst_weight = 0;
int edge_count = 0;
  for (int i = 0; edge_count < V - 1 CC i < e; i++) {
```

```
 struct Edge next_edge = edges[i];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);
    if (x != y) {
      mst_weight += next_edge.weight;
      Union(subsets, x, y);
      edge_count++;}}
  printf("Minimum cost to connect all cities: %d\n", mst_weight);
  free(subsets);}
int main() {
  int V;
  int graph[MAX][MAX];
  printf("Enter number of cities: ");
  scanf("%d", CV);
  printf("Enter the adjacency matrix (0 if no road exists):\n");
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      scanf("%d", Cgraph[i][j]);
  kruskalMST(graph, V);
  return 0;}
```

OUTPUT:

```
5
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Minimum cost to connect all cities: 16
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 24: 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.

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Out will be maximum spanning weight.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct Edge {
  int src, dest, weight;};
struct Subset {
  int parent, rank;};
int compareDesc(const void *a, const void *b) {
  return ((struct Edge *)b)->weight - ((struct Edge *)a)->weight;}
int find(struct Subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
```

```
return subsets[i].parent;}
void Union(struct Subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);
  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;
  else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;}}
void maxSpanningTree(int graph[MAX][MAX], int V) {
  struct Edge edges[MAX * MAX];
  int e = 0;
  for (int i = 0; i < V; i++) {
    for (int j = i + 1; j < V; j++) {
      if (graph[i][j]) {
        edges[e].src = i;
        edges[e].dest = j;
        edges[e].weight = graph[i][j];
        e++;}}}
  qsort(edges, e, sizeof(edges[0]), compareDesc);
  struct Subset *subsets = malloc(V * sizeof(struct Subset));
  for (int v = 0; v < V; v++) {
    subsets[v].parent = v;
    subsets[v].rank = 0;}
  int maxWeight = 0, edgeCount = 0;
```

```
 for (int i = 0; edgeCount < V - 1 CC i < e; i++) {
     int x = find(subsets, edges[i].src);
     int y = find(subsets, edges[i].dest);
     if (x != y) {
       maxWeight += edges[i].weight;
       Union(subsets, x, y);
       edgeCount++;}}
  printf("Maximum cost to connect all cities: %d\n", maxWeight);
  free(subsets);}
int main() {
  int V;
  int graph[MAX][MAX];
  printf("Enter number of cities: ");
  scanf("%d", CV);
  printf("Enter adjacency matrix (0 for no connection):\n");
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      scanf("%d", Cgraph[i][j]);
  maxSpanningTree(graph, V);
  return 0;}
```

OUTPUT:

First line: number of vertices V

Next V lines: adjacency matrix (0 = no edge)

Maximum cost to connect all cities: <total_weight>

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 25: Given a graph, Design an algorithm and implement it using a program to implement FloydWarshall all pair shortest path algorithm.

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list. If a direct edge is notpresent between any pair of vertex (u,v), then this entry is shown as AdjM[u,v] = INF.

Output Format:

Output will be shortest distance matrix in the form of V X V matrix, where each entry (u,v)represents shortest distance between vertex u and vertex v.

*/

CODE:

```
#include <stdio.h>

#define INF 99999 // Define a large value as infinity

#define MAX 100

void floydWarshall(int graph[MAX][MAX], int V) {
  int dist[MAX][MAX];
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      dist[i][j] = graph[i][j];
  for (int k = 0; k < V; k++) {
    // Pick all vertices as source one by one
    for (int i = 0; i < V; i++) {
      // Pick all vertices as destination for the above source
```

```
 for (int j = 0; j < V; j++) {
            if (dist[i][k] != INF CC dist[k][j] != INF CC
              dist[i][j] > dist[i][k] + dist[k][j])
              dist[i][j] = dist[i][k] + dist[k][j];}}}
   printf("Shortest distance matrix:\n");
   for (int i = 0; i < V; i++) {
     for (int j = 0; j < V; j++) {
       if (dist[i][j] == INF)
         printf("INF ");
       else
         printf("%3d ", dist[i][j]); }
     printf("\n");}}
int main() {
   int V;
   int graph[MAX][MAX];
   printf("Enter number of vertices: ");
   scanf("%d", CV);
   printf("Enter the adjacency matrix (use 99999 for INF):\n");
   for (int i = 0; i < V; i++)
     for (int j = 0; j < V; j++)
       scanf("%d", Cgraph[i][j]);
   floydWarshall(graph, V);
   return 0;}
```

OUTPUT:

4

0 5 99999 10

99999 0 3 99999

99999 99999 0 1

99999 99999 99999 0

Shortest distance matrix:

0 5 8 9

INF 0 3 4

INF INF 0 1

INF INF INF 0

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 26: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 xi of item i, where 0 ≤xi≤ 1.

Input Format:

First input line will take number of items N which are provided.Second input line will contain N space-separated array containing weights of all N items.Third input line will contain N space-separated array containing values of all N items.Last line of the input will take the maximum capacity w of knapsack.

Output Format:

First output line will give maximum value that can be achieved.Next Line of output will give list of items selected along with their fraction of amount which has been taken.

*/

CODE:

```
#include  <stdio.h>
#include <stdlib.h>
typedef struct {
   int index;
   float weight;
   float value;
 float ratio;
} Item;
int compare(const void *a, const void *b) {
   Item *item1 = (Item *)a;
```

```
    Item *item2 = (Item *)b;
    if (item2->ratio > item1->ratio) return 1;
    else if (item2->ratio < item1->ratio) return -1;
    else return 0;}
int main() {
    int n;
    float W;
    printf("Enter number of items: ");
    scanf("%d", Cn);
    float weights[n], values[n];
    Item items[n];
    printf("Enter weights of items:\n");
    for (int i = 0; i < n; i++) {
      scanf("%f", Cweights[i]); }
    printf("Enter values of items:\n");
    for (int i = 0; i < n; i++) {
      scanf("%f", Cvalues[i]);}
    printf("Enter maximum capacity of knapsack: ");
    scanf("%f", CW);
    for (int i = 0; i < n; i++) {
      items[i].index = i + 1;
      items[i].weight = weights[i];
      items[i].value = values[i];
      items[i].ratio = values[i] / weights[i]; }
qsort(items, n, sizeof(Item), compare);
    float totalValue = 0.0;
    float remainingWeight = W;
    printf("\nItems selected (index and fraction taken):\n");
```

```
  for (int i = 0; i < n CC remainingWeight > 0; i++) {
    if (items[i].weight <= remainingWeight) {
      // Take full item
      totalValue += items[i].value;
      remainingWeight -= items[i].weight;
      printf("Item %d: 1.00\n", items[i].index);
    } else {
      // Take fraction of item
      float fraction = remainingWeight / items[i].weight;
      totalValue += items[i].value * fraction;
      printf("Item %d: %.2f\n", items[i].index, fraction);
      break;}} // Knapsack is full
  printf("\nMaximum value that can be achieved: %.2f\n", totalValue);
  return 0;}
```

OUTPUT:

```
Enter number of items: 3
Enter weights of items:
10 20 30
Enter values of items:
60 100 120
Enter maximum capacity of knapsack: 50
Items selected (index and fraction taken):
Item 1: 1.00
Item 2: 1.00
Item 3: 0.33
Maximum value that can be achieved: 240.00
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 27: 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)

Input Format:

First line will take the size n of the array.

Second line will take array s an input.

Output Format:

Output will be the minimum computation cost required to merge all the elements of the array.

*/

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_SIZE 1000

void heapify(int heap[], int n, int i) {

  int smallest = i;

  int left = 2*i + 1;

  int right = 2*i + 2;

  if (left < n CC heap[left] < heap[smallest])

    smallest = left;

  if (right < n CC heap[right] < heap[smallest])

    smallest = right;

 if (smallest != i) {
```

```
 int temp = heap[i];
      heap[i] = heap[smallest];
      heap[smallest] = temp;
      heapify(heap, n, smallest);}}
void buildMinHeap(int heap[], int n) {
   for (int i = n/2 - 1; i >= 0; i--)
      heapify(heap, n, i);}
int extractMin(int heap[], int *n) {
   int min = heap[0];
   heap[0] = heap[--(*n)];
   heapify(heap, *n, 0);
   return min;}
void insertHeap(int heap[], int *n, int value) {
   int i = (*n)++;
   heap[i] = value;
   while (i > 0 CC heap[(i-1)/2] > heap[i]) {
      int temp = heap[i];
      heap[i] = heap[(i-1)/2];
      heap[(i-1)/2] = temp;
      i = (i-1)/2;}}
int main() {
   int n, arr[MAX_SIZE];
   printf("Enter number of files: ");
   scanf("%d", Cn);
   printf("Enter sizes of the files:\n");
   for (int i = 0; i < n; i++)
      scanf("%d", Carr[i]);
buildMinHeap(arr, n);
```

```
int totalCost = 0;
  while (n > 1) {
    int x = extractMin(arr, Cn);
    int y = extractMin(arr, Cn);
    int cost = x + y;
    totalCost      +=      cost;
    insertHeap(arr, Cn, cost);}
  printf("Minimum computation cost: %d\n", totalCost);
  return 0;}
```

OUTPUT:

```
Enter number of files: 4
Enter sizes of the files:
20 30 10 5

Minimum computation cost: 115
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 28: 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.

Input Format:

First line of input will take number of activities N.

Second line will take N space-separated values defining starting time for all the N activities.

Third line of input will take N space-separated values defining finishing time for all the N

activities.

Output Format:

Output will be the number of non-conflicting activities and the list of selected activities.

*/

CODE:

```
#include  <stdio.h>
#include <stdlib.h>
typedef struct {
  int index;
  int start;
  int end;
} Activity;
int compare(const void *a, const void *b) {
  Activity *actA = (Activity *)a;
```

```
  Activity *actB = (Activity *)b;
   return actA->end - actB->end;}
int main() {
   int n;
   printf("Enter number of activities: ");
   scanf("%d", Cn);
   int start[n], end[n];
   printf("Enter start times:\n");
   for (int i = 0; i < n; i++)
      scanf("%d", Cstart[i]);
   printf("Enter end times:\n");
   for (int i = 0; i < n; i++)
      scanf("%d", Cend[i]);
   Activity activities[n];
   for (int i = 0; i < n; i++) {
      activities[i].index = i;
      activities[i].start = start[i];
      activities[i].end = end[i];}
   qsort(activities, n, sizeof(Activity), compare);
   printf("\nSelected activities: ");
   int count = 1;
   int lastEnd = activities[0].end;
   printf("%d ", activities[0].index);  // selecting first activity
   for (int i = 1; i < n; i++) {
      if (activities[i].start >= lastEnd) {
         printf("%d ", activities[i].index);
         lastEnd = activities[i].end;
         count++;}}
```

```
 printf("\nMaximum number of non-conflicting activities: %d\n", count);
   return 0;}
```

OUTPUT:

```
Enter number of activities: 6
Enter start times:
1 3 0 5 8 5
Enter end times:
2 4 6 7 9 9

Selected activities: 0 1 3 4
Maximum number of non-conflicting activities: 4
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 2G: Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

Input Format:

First line will give total number of tasks n.

Second line of input will give n space-separated elements of array representing time taken by each task.

Third line of input will give n space-separated elements of array representing deadline associated with each task.

Output Format:

Output will be the total number of maximum tasks that can be completed.

*/

CODE:

```
#include  <stdio.h>

#include <stdlib.h>

typedef struct {

  int time;

  int deadline;

  int index;  // to identify task

} Task;

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

  Task* t1 = (Task*)a;

  Task* t2 = (Task*)b;
```

```
  return t1->deadline - t2->deadline;}
int findMaxIndex(Task selected[], int count) {
  int maxIndex = 0;
  for (int i = 1; i < count; i++) {
    if (selected[i].time > selected[maxIndex].time) {
      maxIndex = i;}}
  return maxIndex;}
int main() {
  int n;
  printf("Enter number of tasks: ");
  scanf("%d", Cn);
  Task tasks[n];
  int timeArr[n], deadlineArr[n];
  printf("Enter time for each task: ");
  for (int i = 0; i < n; i++) {
    scanf("%d", CtimeArr[i]); }
  printf("Enter deadline for each task: ");
  for (int i = 0; i < n; i++) {
    scanf("%d", CdeadlineArr[i]);}
  for (int i = 0; i < n; i++) {
    tasks[i].time = timeArr[i];
    tasks[i].deadline = deadlineArr[i];
    tasks[i].index = i + 1;  // for identification}
  qsort(tasks, n, sizeof(Task), compare);
  Task selected[n];  // to store selected tasks
  int totalTime = 0, selectedCount = 0;
  for (int i = 0; i < n; i++) {
    totalTime += tasks[i].time;
```

```
selected[selectedCount++] = tasks[i];
      if (totalTime > tasks[i].deadline) {
        int maxIndex = findMaxIndex(selected, selectedCount);
        totalTime -= selected[maxIndex].time;
        // Remove the max time task
        for (int j = maxIndex; j < selectedCount - 1; j++) {
          selected[j] = selected[j + 1];}
        selectedCount--;}}
  printf("\nMaximum number of tasks that can be completed: %d\n", selectedCount);
  printf("Selected tasks (time, deadline):\n");
  for (int i = 0; i < selectedCount; i++) {
    printf("Task %d: Time = %d, Deadline = %d\n", selected[i].index, selected[i].time,
selected[i].deadline);}
  return 0;}
```

OUTPUT:

```
Enter number of tasks: 5
Enter time for each task: 2 1 2 1 3
Enter deadline for each task: 2 2 3 3 6

Maximum number of tasks that can be completed: 4
Selected tasks (time, deadline):
Task 1: Time = 2, Deadline = 2
Task 2: Time = 1, Deadline = 2
Task 4: Time = 1, Deadline = 3
Task 5: Time = 3, Deadline = 6
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 30: 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.

Input Format:

First line of input will give size n of array.

Second line of input will take n space-separated elements of array.

Output Format:

First line of output will be 'yes' if majority element exists, otherwise print 'no'.

Second line of output will print median of the array.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b) {
  return (*(int *)a - *(int *)b);}
int findCandidate(int arr[], int n) {
  int count = 1, candidate = arr[0];
  for (int i = 1; i < n; i++) {
    if (arr[i] == candidate)
      count++;
    else
      count--;
     if (count == 0) {
```

```
            candidate = arr[i];
            count = 1;}}
    return candidate;}
int isMajority(int arr[], int n, int candidate) {
    int count = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] == candidate)
            count++;
    return count > n / 2;}
float findMedian(int arr[], int n) {
    qsort(arr, n, sizeof(int), compare);
    if (n % 2 == 0)
        return (arr[n/2 - 1] + arr[n/2]) / 2.0;
    else
        return arr[n/2];}
int main() {
    int n;
    printf("Enter size of array: ");
    scanf("%d", Cn);
    int arr[n];
    printf("Enter array elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", Carr[i]);
    int candidate = findCandidate(arr, n);
    if (isMajority(arr, n, candidate))
        printf("yes\n");
    else
        printf("no\n");
```

```
float median = findMedian(arr, n);
   printf("%.2f\n", median);
   return 0;}
```

OUTPUT:

```
Enter size of array: 7
Enter array elements: 3 3 4 2 4 4 4

no
4.00
```

/*Z

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 31: 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.

Input Format:

First line of input will take number of matrices n that you need to multiply.

For each line i in n, take two inputs which will represent dimensions aXb of matrix i.

Output Format:

Output will be the minimum number of operations that are required to multiply the list of matrices.

*/

CODE:

```
#include <stdio.h>
#include <limits.h>
void matrixChainOrder(int p[], int n) {
  int m[n][n];
  int s[n][n];
    for (int i = 1; i < n; i++)
    m[i][i] = 0;
    for (int length = 2; length < n; length++) {
    for (int i = 1; i < n - length + 1; i++) {
      int j = i + length - 1;
      m[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
        int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
```

```
            if (q < m[i][j]) {
              m[i][j] = q;
              s[i][j] = k;}}}}
           printf("Minimum number of scalar multiplications: %d\n", m[1][n - 1]);
      printf("Optimal parenthesization: ");
   printOptimalParenthesization(s, 1, n - 1);
   printf("\n");}
void printOptimalParenthesization(int s[], int i, int j) {
   if (i == j)
      printf("A%d", i);
   else {
      printf("(");
      printOptimalParenthesization(s, i, s[i][j]);
      printOptimalParenthesization(s, s[i][j] + 1, j);
      printf(")");}}
int main() {
   int n;
   printf("Enter number of matrices: ");
   scanf("%d", Cn);
      int p[n + 1];
      printf("Enter dimensions of matrices:\n");
   for (int i = 0; i < n; i++) {
      int rows, cols;
      scanf("%d %d", Crows, Ccols);
      p[i] = rows;
      if (i == n - 1) {
         p[i + 1] = cols; } }
      matrixChainOrder(p, n + 1);  return 0;}
```

OUTPUT:

Enter number of matrices: 3

Enter dimensions of matrices:

10 20

20 30

30 40

Minimum number of scalar multiplications: 18000

Optimal parenthesization: (A1(A2A3))

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 32: 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.

Input Format:

First line of input will take number of coins that are available.

Second line of input will take the value of each coin.

Third line of input will take the value N for which you need to find sum.

Output Format:

Output will be the number of ways.

*/

CODE:

```
#include <stdio.h>
void countWays(int coins[], int m, int N) {
  int dp[N + 1];
  for (int i = 0; i <= N; i++) {
    dp[i] = 0; }
    dp[0] = 1;
    for (int i = 0; i < m; i++) {
    for (int j = coins[i]; j <= N; j++) {
      dp[j] += dp[j - coins[i]];}}
  printf("Number of ways to make sum %d: %d\n", N, dp[N]);}
int main() {
  int m, N;
```

```
  printf("Enter number of coins: ");
  scanf("%d", Cm);
    int coins[m];
  printf("Enter coin denominations: ");
  for (int i = 0; i < m; i++) {
    scanf("%d", Ccoins[i]);}
  printf("Enter the value N: ");
  scanf("%d", CN);
    countWays(coins, m, N);
  return 0;}
```

OUTPUT:

```
Enter number of coins: 3
Enter coin denominations: 1 2 3
Enter the value N: 4
Number of ways to make sum 4: 4
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 33: 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.

Input Format:

First line of input will take number of elements n present in the set.

Second line of input will take n space-separated elements of the set.

Output Format:

Output will be 'yes' if two such subsets found otherwise print 'no'.

*/

CODE:

```
#include <stdio.h>
#include <stdbool.h>
bool canPartition(int arr[], int n) {
  int total_sum = 0;
  for (int i = 0; i < n; i++) {
    total_sum += arr[i];}
  if (total_sum % 2 != 0) {
    return false;}
  int subset_sum = total_sum / 2;
  bool dp[subset_sum + 1];
  dp[0] = true; // A sum of 0 can always be achieved (by not taking any element)
  for (int i = 1; i <= subset_sum; i++) {
    dp[i] = false;}
  for (int i = 0; i < n; i++) {
```

```
        for (int j = subset_sum; j >= arr[i]; j--) {
            dp[j] = dp[j] || dp[j - arr[i]];}}
            return dp[subset_sum];}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", Cn);
    int arr[n];
    printf("Enter the elements of the set: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", Carr[i]); }
    if (canPartition(arr, n)) {
        printf("yes\n");
    } else {
        printf("no\n");}
return 0;}
```

OUTPUT:

```
Enter number of elements: 4
Enter the elements of the set: 1 5 11 5
yes
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 34: Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the

same relative order, but not necessarily contiguous.

Input Format:

First input line will take character sequence 1.

Second input line will take character sequence 2.

Output Format:

Output will be the longest common subsequence along with its length.

*/

CODE:

```
#include <stdio.h>
#include <string.h>
void printLCS(char *X, char *Y, int m, int n, int dp[m+1][n+1]) {
  char lcs[m+n];
  int index = dp[m][n];
  int i = m, j = n;
  while (i > 0 CC j > 0) {
    if (X[i-1] == Y[j-1]) {
      lcs[index-1] = X[i-1]; // Add the character to the LCS
      i--; j--; index--;  // Move diagonally up-left
    } else if (dp[i-1][j] > dp[i][j-1]) {
      i--;  // Move up
    } else {
```

```
 j--; }} // Move left
   lcs[dp[m][n]] = '\0';  // Null-terminate the string
   printf("Longest Common Subsequence: %s\n", lcs);}
int main() {
   char X[100], Y[100];
   printf("Enter first sequence: ");
   scanf("%s", X);
   printf("Enter second sequence: ");
   scanf("%s", Y);
   int m = strlen(X);
   int n = strlen(Y);
   int dp[m+1][n+1];
   for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
         if (i == 0 || j == 0) {
            dp[i][j] = 0;  // Base case: LCS of an empty string is 0
         } else if (X[i-1] == Y[j-1]) {
            dp[i][j] = dp[i-1][j-1] + 1;  // If characters match
         } else {
            dp[i][j] = (dp[i-1][j] > dp[i][j-1]) ? dp[i-1][j] : dp[i][j-1]; }}} // Take the max
       printf("Length of Longest Common Subsequence: %d\n", dp[m][n]);
   printLCS(X, Y, m, n, dp);
   return 0;}
```

OUTPUT:

```
Enter first sequence: ABCBDAB
Enter second sequence: BDCABB
Length of Longest Common Subsequence: 4
Longest Common Subsequence: BCAB
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 35: Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. 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. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

Input Format:

First line of input will provide number of items n.

Second line of input will take n space-separated integers describing weights for all items.

Third line of input will take n space-separated integers describing value for each item.

Last line of input will give the knapsack capacity.

Output Format:

Output will be maximum value that can be achieved and list of items selected along with their

weight and value

*/

CODE:

```
#include <stdio.h>

#include <stdlib.h>

void knapsack(int n, int w, int weight[], int value[]) {

  // Create a DP table

  int dp[n + 1][w + 1];

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

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

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

```
            dp[i][j] = 0;  // Base case: no items or no capacity
        } else if (weight[i - 1] <= j) {
            // Take the maximum value between including or excluding the current item
            dp[i][j] = (dp[i - 1][j] > (value[i - 1] + dp[i - 1][j - weight[i - 1]])) ?
                    dp[i - 1][j] : (value[i - 1] + dp[i - 1][j - weight[i - 1]]);
        } else {
            dp[i][j] = dp[i - 1][j];}}}  // If the item cannot be included
    printf("Maximum value that can be achieved: %d\n", dp[n][w]);
    printf("Selected items (index, weight, value):\n");
    int i = n, j = w;
    while (i > 0 CC j > 0) {
      if (dp[i][j] != dp[i - 1][j]) {
        // The item is included
        printf("Item %d: Weight = %d, Value = %d\n", i, weight[i - 1], value[i - 1]);
        j -= weight[i - 1]; } // Reduce the remaining capacity
       i--;}}
int main() {
    int n;
    printf("Enter number of items: ");
    scanf("%d", Cn);
    int weight[n], value[n];
      printf("Enter weights of the items: ");
    for (int i = 0; i < n; i++) {
      scanf("%d", Cweight[i]);}
    printf("Enter values of the items: ");
    for (int i = 0; i < n; i++) {
      scanf("%d", Cvalue[i]); }
```

```
int W;
printf("Enter the knapsack capacity: ");
scanf("%d", CW);
knapsack(n, W, weight, value);
return 0;}
```

OUTPUT:

```
Enter number of items: 4
Enter weights of the items: 2 3 4 5
Enter values of the items: 3 4 5 6
Enter the knapsack capacity: 5

Maximum value that can be achieved: 7
Selected items (index, weight, value):
Item 3: Weight = 4, Value = 5
Item 1: Weight = 2, Value = 3
```

/*

NAME:ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 36: Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.

Input Format:

String of characters is provided as an input.

Output Format:

Output will be the list of all possible permutations in lexicographic order.

*/

CODE:

```
#include <stdio.h>
#include <string.h>
void swap(char* a, char* b) {
  char temp = *a;
  *a = *b;
  *b = temp;}
void permute(char* str, int l, int r) {
  if (l == r) {
    printf("%s\n", str);
  } else {
    for (int i = l; i <= r; i++) {
      // Skip duplicate characters to avoid generating same permutation again
      if (i != l CC str[i] == str[l]) {
        continue; }
      swap(Cstr[l], Cstr[i]); // Swap characters at position l and i
```

```
        permute(str, l + 1, r); // Recurse for the remaining characters
        swap(Cstr[l], Cstr[i]);}}} // Backtrack: undo the swap
int main() {
  char str[100];
  printf("Enter a string: ");
  scanf("%s", str);
  int n = strlen(str);
  qsort(str, n, sizeof(char), (int (*)(const void*, const void*)) strcmp);
  printf("All permutations in lexicographic order:\n");
  permute(str, 0, n - 1);
  return 0;}
```

OUTPUT:

```
Enter a string: abc
All permutations in lexicographic order:
abc
acb
bac
bca
cab
cba
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 37: Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = O(n))

Input Format:

First line of input will give the size n of the character array.

Second line of input will give n space-separated elements to character array.

Output Format:

Output will be the list of characters present in the array in alphabetical order

and frequency of each character in the array.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct CharacterFreq {
  char character;
  int frequency;};
int compare(const void *a, const void *b) {
  struct CharacterFreq *char1 = (struct CharacterFreq *)a;
  struct CharacterFreq *char2 = (struct CharacterFreq *)b;
  return char1->character - char2->character;}
int main() {
  int n;
  printf("Enter the size of the array: ");
```

```
scanf("%d", Cn);
  char arr[n];
  printf("Enter the characters in the array: ");
  for (int i = 0; i < n; i++) {
    scanf(" %c", Carr[i]);}  // Read characters from input
    struct CharacterFreq freqMap[256];  // Assuming ASCII characters, 256 possible
characters
  int distinctCount = 0;
  for (int i = 0; i < 256; i++) {
    freqMap[i].character = 0;  // Empty character
   freqMap[i].frequency = 0;}  // No frequency initially
   for (int i = 0; i < n; i++) {
    char currentChar = arr[i];
    if (freqMap[currentChar].frequency == 0) {
      freqMap[currentChar].character = currentChar;  // Set character if it's the first
occurrence
      distinctCount++; } // Increment the distinct character count
    freqMap[currentChar].frequency++;  }// Increment the frequency of the character
    struct CharacterFreq distinctChars[distinctCount];
  int j = 0;
  for (int i = 0; i < 256; i++) {
    if (freqMap[i].frequency > 0) {
      distinctChars[j] = freqMap[i];
      j++;}}
  qsort(distinctChars, distinctCount, sizeof(struct CharacterFreq), compare);
  printf("Distinct characters and their frequencies:\n");
  for (int i = 0; i < distinctCount; i++) {
    printf("%c: %d\n", distinctChars[i].character, distinctChars[i].frequency); }
return 0;}
```

OUTPUT:

Enter the size of the array: 8

Enter the characters in the array: b a c a b c c a

Distinct characters and their frequencies:

a: 3

b: 2

c: 3

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 38: Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicate within a small window of size k < n.

Input Format:

First input line contains number of test cases T.

For each test case T, there will be three input lines.

First line contains size n of array.

Second input line contains n space-separated array elements.

Third input line contains value k.

Output Format:

Output will have T number of lines.For each test case, output will be “Duplicate present in window k” if the duplicate element is found in the array, otherwise “Duplicate not present in window k”.

*/

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX_SIZE 10000

bool hasDuplicate(int arr[], int n, int k) {

  bool hashSet[MAX_SIZE] = {false};

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

    if (hashSet[arr[i]]) {

return true;} // Duplicate found
```

```
    hashSet[arr[i]] = true;
    if (i >= k) {
      hashSet[arr[i - k]] = false;}}
  return false;}
int main() {
  int T;
  scanf("%d", CT); // Number of test cases
  while (T--) {
    int n, k;
    scanf("%d", Cn); // Size of the array
    int arr[n];
    for (int i = 0; i < n; i++) {
      scanf("%d", Carr[i]); } // Input array elements
    scanf("%d", Ck); // Window size k
    if (hasDuplicate(arr, n, k)) {
      printf("Duplicate present in window %d\n", k);
    } else {
      printf("Duplicate not present in window %d\n", k);}}
  return 0;}
```

OUTPUT:

2

5

1 2 3 4 5

3

6

1 2 3 1 5 6

2

Duplicate not present in window 3.Duplicate present in window 2

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 3G: Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a*b = c*d, where a, b, c and d are distinct elements of array.

Input Format:

First line of input will give size of array n.

Second line of input will give n space-separated array elements.

Output Format:

First line of output will give pair (a,b)

Second line of output will give pair (c,d).

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 1000
typedef struct {
  int first;
  int second;
} Pair;
void findPairs(int arr[], int n) {
  int productMap[MAX_SIZE] = {0};
  Pair pairs[MAX_SIZE];
    for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      int product = arr[i] * arr[j];
```

```
        if (productMap[product] > 0) {
          for (int k = 0; k < productMap[product]; k++) {
            Pair firstPair = pairs[k];
            // Ensure pairs have distinct elements
            if (firstPair.first != arr[i] CC firstPair.first != arr[j] CC
              firstPair.second != arr[i] CC firstPair.second != arr[j]) {
              printf("Pair 1: (%d, %d)\n", firstPair.first, firstPair.second);
              printf("Pair 2: (%d, %d)\n", arr[i], arr[j]);
              return;}}}
            pairs[productMap[product]] = (Pair){arr[i], arr[j]};
        productMap[product]++;}}
      printf("No such pairs found\n");}
int main() {
  int n;
  scanf("%d", Cn);  // Number of elements in array
  int arr[n];
  for (int i = 0; i < n; i++) {
    scanf("%d", Carr[i]); }
  findPairs(arr, n);
  return 0;}
```

OUTPUT:

```
6
1 2 3 6 4 5
Pair 1: (1, 6)
Pair 2: (2, 3)
```

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 40: Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12,15, 16, 18, 20, 24, ……is sequence of ugly numbers.

Input:

First line of input will give number of test cases T.

For each test case T, enter a number n.

Output:

There will be T output lines.

For each test case T, Output will be nth ugly number.

*/

CODE:

```
#include    <stdio.h>
#include   <stdlib.h>
#define MAX 100000
int getNthUglyNumber(int n) {
  int ugly[n];
  ugly[0] = 1; // First ugly number is 1
     int i2 = 0, i3 = 0, i5 = 0;
     int next2 = 2, next3 = 3, next5 = 5;
  for (int i = 1; i < n; i++) {
     int nextUgly = (next2 < next3) ? next2 : next3;
     nextUgly = (nextUgly < next5) ? nextUgly : next5;
          ugly[i] = nextUgly;
```

```
    if (nextUgly == next2) {
      i2++;
      next2 = ugly[i2] * 2; }
    if (nextUgly == next3) {
      i3++;
      next3 = ugly[i3] * 3;}
    if (nextUgly == next5) {
      i5++;
      next5 = ugly[i5] * 5;}}
     return ugly[n - 1];}
int main() {
  int T;
  scanf("%d", CT);  // Number of test cases
  while (T--) {
    int n;
    scanf("%d", Cn);  // Read nth ugly number to find
    printf("%d\n", getNthUglyNumber(n)); } // Output the nth ugly number
  return 0;}
```

OUTPUT:

2

7

10

8

12

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 41: Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.

Input:

Graph in the form of adjacency matrix or adjacency list is provided as an input.

Output:

Output will be the mother vertex number.

Solved Example: Consider a directed graph:

In this graph, vertex 0 is mother vertex.

*/

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int v, int visited[], int n) {
  visited[v] = 1;
  for (int i = 0; i < n; i++) {
    if (graph[v][i] CC !visited[i]) {
      dfs(graph, i, visited, n);}}}
int findMotherVertex(int graph[MAX_VERTICES][MAX_VERTICES], int n) {
  int visited[MAX_VERTICES] = {0};
  int lastV = -1;
  for (int i = 0; i < n; i++) {
    if (!visited[i]) {
```

```
            dfs(graph, i, visited, n);
            lastV = i;}}
        for (int i = 0; i < n; i++) {
        visited[i] = 0;}
    dfs(graph, lastV, visited, n);
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            return -1; }}// No mother vertex
    return lastV; }// Mother vertex found
int main() {
    int n, m;
    printf("Enter number of vertices in the graph: ");
    scanf("%d", Cn);
    int graph[MAX_VERTICES][MAX_VERTICES] = {0};
    printf("Enter number of edges in the graph: ");
    scanf("%d", Cm);
    printf("Enter the edges (u v) in the form of 0-based indices:\n");
    for (int i = 0; i < m; i++) {
        int u, v;
        scanf("%d %d", Cu, Cv);
        graph[u][v] = 1;} // Directed edge from u to v
    int motherVertex = findMotherVertex(graph, n);
    if (motherVertex == -1) {
        printf("No mother vertex found.\n");
    } else {
        printf("The mother vertex is: %d\n", motherVertex); }
    return 0;}
```

OUTPUT:

Enter number of vertices in the graph: 5

Enter number of edges in the graph: 5

Enter the edges (u v) in the form of 0-based indices:

0 1

0 2

1 3

4 1

4 2

The mother vertex is: 4

/*

NAME: ARYAN SINGH

UNIVERSITY ROLLNO:2023288

SECTION:H

PROBLEM STATEMENT 41:Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A

mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.

Input:

Graph in the form of adjacency matrix or adjacency list is provided as an input.

Output:

Output will be the mother vertex number.

Solved Example: Consider a directed graph:

In this graph, vertex 0 is mother vertex.

*/

CODE:

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

#define MAX_VERTICES 100

// DFS function to mark the visited vertices
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int v, int visited[], int n) {
  visited[v] = 1;
  for (int i = 0; i < n; i++) {
    if (graph[v][i] CC !visited[i]) {
      dfs(graph, i, visited, n);
    }
```

```
  }
}

// Function to find the mother vertex
int findMotherVertex(int graph[MAX_VERTICES][MAX_VERTICES], int n) {
  int visited[MAX_VERTICES] = {0};
  int lastV = -1;

  // Step 1: Do a DFS and find the last finished vertex
  for (int i = 0; i < n; i++) {
    if (!visited[i]) {
      dfs(graph, i, visited, n);
      lastV = i;
    }
  }

  // Step 2: Perform DFS from the last finished vertex and check if all vertices are
reachable
  for (int i = 0; i < n; i++) {
    visited[i] = 0;
  }

  dfs(graph, lastV, visited, n);

  // Step 3: If all vertices are visited, then the last finished vertex is the mother vertex
  for (int i = 0; i < n; i++) {
    if (!visited[i]) {
      return -1; // No mother vertex
```

```
    }
  }
  return lastV; // Mother vertex found
}

int main() {
  int n, m;

  printf("Enter number of vertices in the graph: ");
  scanf("%d", Cn);

  int graph[MAX_VERTICES][MAX_VERTICES] = {0};

  printf("Enter number of edges in the graph: ");
  scanf("%d", Cm);

  printf("Enter the edges (u v) in the form of 0-based indices:\n");
  for (int i = 0; i < m; i++) {
    int u, v;
    scanf("%d %d", Cu, Cv);
    graph[u][v] = 1; // Directed edge from u to v
  }

  int motherVertex = findMotherVertex(graph, n);

  if (motherVertex == -1) {
    printf("No mother vertex found.\n");
  } else {
```

```
    printf("The mother vertex is: %d\n", motherVertex);
  }

  return 0;
}
```

OUTPUT:

```
Enter number of vertices in the graph: 5
Enter number of edges in the graph: 5
Enter the edges (u v) in the form of 0-based indices:
0 1
0 2
1 3
4 1
4 2

The mother vertex is: 4
```