**DSA**

**Submitted By: Rishant Kushwaha Submitted To:**

**SAP ID:500126797 Virendra Kandyan**

**Roll No.: R2142232084**

**Q1. Explain the term arrays as a linear data structure.**

*Answer:* Arrays are a linear data structure consisting of a collection of elements, each identified by at least one array index or key. They store elements in contiguous memory locations, allowing efficient access to elements using their indices. Arrays have a fixed size, meaning the number of elements they can hold is predetermined at the time of declaration.

**Q2. What are sparse matrices? Give an example.**

*Answer:* Sparse matrices are matrices where the majority of elements are zero. They arise in various applications, such as finite element analysis and graph algorithms, where only a few entries are non-zero. An example of a sparse matrix is:

0 0 0 0 0

0 0 0 0 0

0 5 0 0 0

0 0 0 0 0

**Q3. How are two-dimensional arrays represented in memory? Also obtain the formula for calculating the address of any element stored in the array, in the case of column-major.**

*Answer:* In memory, a 2D array is represented linearly, either row-by-row or column-by-column.

For column-major representation, the formula to calculate the address of any element stored in the array is:

address = base\_address + (column\_index \* number\_of\_rows + row\_index) \* size\_of\_data\_type

**Q4. Derive the formula to find the physical address of an element of three-dimensional arrays stored in row-major order.**

*Answer:* The formula to find the physical address of an element in a 3D array stored in row-major order is:

address = base\_address + ((z \* size\_of\_y \* size\_of\_x) + (y \* size\_of\_x) + x) \* size\_of\_data\_type

**Q5. Write a program to input a matrix and to determine if it is an asymmetrical matrix.**

1. #include <stdio.h>
2. #define MAX\_SIZE 10
3. int main() {
4. int matrix[MAX\_SIZE][MAX\_SIZE];
5. int rows, cols, i, j;
6. int asymmetric = 0;
7. printf("Enter the number of rows and columns: ");
8. scanf("%d %d", &rows, &cols);
9. printf("Enter the elements of the matrix:\n");
10. for (i = 0; i < rows; i++) {
11. for (j = 0; j < cols; j++) {
12. scanf("%d", &matrix[i][j]);
13. }
14. }
15. // Check if the matrix is asymmetric
16. for (i = 0; i < rows; i++) {
17. for (j = 0; j < cols; j++) {
18. if (matrix[i][j] != matrix[j][i]) {
19. asymmetric = 1;
20. break;
21. }
22. }
23. if (asymmetric == 1) {
24. break;
25. }
26. }
27. if (asymmetric == 1) {
28. printf("The matrix is asymmetric.\n");
29. } else {
30. printf("The matrix is symmetric.\n");
31. }
32. return 0;
33. }

**Q6. Write a C program to display the content of pointers using arrays of pointers.**

1. #include <stdio.h>
2. int main() {
3. int a = 10, b = 20, c = 30;
4. int \*ptr[3] = {&a, &b, &c};
5. for (int i = 0; i < 3; i++) {
6. printf("Value of pointer %d: %d\n", i, \*ptr[i]);
7. }
8. return 0;
9. }

**Q7. Write an algorithm to count the common elements of two one-dimensional arrays.**

Algorithm to count common elements of two arrays:

1. Initialize a counter variable to 0.

2. Iterate through the elements of the first array.

3. For each element, iterate through the elements of the second array.

4. If a common element is found, increment the counter and break out of the inner loop.

5. After both loops, the counter holds the count of common elements.

6. Return or display the counter value.

**Q8. Write a short note on any two:**

a. **Sparse matrix and its representation:**

* A sparse matrix is a matrix where the majority of its elements are zero. They arise in many practical applications where the data is sparse, meaning there are fewer non-zero elements compared to zero elements.
* Sparse matrices are typically represented using one of several storage formats such as:
  + **Coordinate list (COO):** In this format, each non-zero element is stored along with its row and column indices.
  + **Compressed sparse row (CSR):** Here, the non-zero elements are stored row-wise along with additional arrays to represent the column indices and the starting positions of rows.
  + **Compressed sparse column (CSC):** Similar to CSR, but the non-zero elements are stored column-wise.
* Sparse matrix representations help in reducing storage space and computational overhead when dealing with large matrices with mostly zero values.

b. **Static Vs Dynamic memory allocation:**

* **Static memory allocation:** Memory is allocated at compile-time, and the size of the memory block is fixed throughout the program's execution. Examples include arrays declared with a fixed size.
* **Dynamic memory allocation:** Memory is allocated at runtime, and the size of the memory block can be determined dynamically based on program logic. Common functions for dynamic memory allocation include **malloc()**, **calloc()**, and **realloc()** in C.
* Static allocation is suitable when the size of the data is known at compile-time and doesn't change during program execution. Dynamic allocation is useful when the size of the data is not known until runtime or when flexibility in memory management is required.

**Q9. Explain the memory representation of two-dimensional arrays with the help of examples.**

* **Memory representation of two-dimensional arrays:**
  + In memory, a two-dimensional array is stored in a contiguous block of memory.
  + The elements are stored row by row, meaning the elements of the first row are stored first, followed by the elements of the second row, and so on.
  + The memory representation of a two-dimensional array can be visualized as a grid, where each cell represents an array element.
  + Example: Consider a 2x3 array:

| 1 | 2 | 3 |

| 4 | 5 | 6 |

In memory, it might be stored as **[1, 2, 3, 4, 5, 6]**.

If you wanted to store the array column-wise, it would be stored as **[1, 4, 2, 5, 3, 6],** but this is not the typical way two-dimensional arrays are stored in most programming languages.

**Q10. Draw a linked list and vector representation of the following sparse matrix:**

0 12 0 0

0 11 0 0

9 0 0 0

0 0 0 0

**Linked list representation:**

(0,1,12) -> (1,1,11) -> (2,0,9)

**Vector representation:**

[ (0,1,12), (1,1,11), (2,0,9)]

**Q11. Explain the application areas of the array with a suitable example.**

Arrays find applications in various fields such as:

* **Data Structures:** Arrays are fundamental data structures used to store collections of data elements. For example, in a stack data structure, an array can be used to implement the underlying storage.
* **Numerical Analysis:** Arrays are extensively used in numerical analysis and scientific computing to represent vectors, matrices, and multidimensional data. For instance, in image processing, arrays are used to store pixel values.
* **Algorithms:** Many algorithms rely on arrays for efficient data manipulation. Sorting algorithms like merge sort and searching algorithms like binary search operate on arrays.
* **Operating Systems:** Arrays are used in operating systems to manage system resources efficiently. For example, in memory management, arrays are used to represent page tables.
* **Embedded Systems:** Arrays are commonly used in embedded systems programming to store configuration data, sensor readings, and other system parameters efficiently.

**Q12. Write an algorithm for matrix operations such as the addition of two matrices, multiplication, subtraction, and transpose.**

* **Addition of two matrices:**
  + Iterate over each element of both matrices and add corresponding elements.
  + Store the result in a new matrix of the same dimensions.
* **Subtraction of two matrices:**
  + Iterate over each element of both matrices and subtract corresponding elements.
  + Store the result in a new matrix of the same dimensions.
* **Multiplication of two matrices:**
  + Use nested loops to iterate over rows and columns of both matrices.
  + Perform dot product of rows of the first matrix and columns of the second matrix.
  + Store the result in the corresponding position of the resulting matrix.
* **Transpose of a matrix:**
  + Iterate over each element of the matrix and swap its row and column indices.
  + Store the result in a new matrix where rows become columns and vice versa.

**Q13. Determine the formula to find the address location of an element in three-dimensional arrays, suppose each element takes four bytes of space and elements are stored in row-major order.**

* **Address location of an element in a 3D array:**
  + Suppose the dimensions of the 3D array are **M x N x P**.
  + If **A[i][j][k]** is the element, the address can be calculated using the formula:

Address(A[i][j][k]) = Base\_Address + ((i \* N \* P) + (j \* P) + k) \* Size\_of\_Element

* + - Where:
    - Base\_Address is the starting memory address of the array.
    - Size\_of\_Element is the size of each element in bytes.
    - N is the number of columns.
    - P is the number of planes.

Q14. Write and explain the algorithm to find the 7th smallest element in an array.

*Answer:* To find the 7th smallest element in an array, we can use various algorithms like sorting, min-heap, or quick select. Here, I'll explain the algorithm using quick select, which has an average time complexity of O(n).

**Algorithm to Find the 7th Smallest Element using Quick select:**

1. **Choose a Pivot:** Choose a pivot element from the array. This can be done randomly or by selecting the middle element.
2. **Partition the Array:** Partition the array into two parts: elements less than the pivot and elements greater than the pivot. After partitioning, the pivot is in its correct sorted position.
3. **Recursively Select a Subarray:** Check the position of the pivot. If it's at the 7th index, return it as the 7th smallest element. If the pivot's position is greater than 7, recursively select the left subarray. If the pivot's position is less than 7, recursively select the right subarray.
4. **Repeat:** Repeat steps 1 to 3 until the 7th smallest element is found.

**Explanation:**

* Quick select is a variation of the quicksort algorithm where we only focus on the partition that contains the desired element.
* By repeatedly partitioning the array around a pivot element, we can quickly find the kth smallest element.
* At each step, after partitioning, we know the exact position of the pivot in the sorted array. This allows us to determine which subarray to focus on.
* The average time complexity of quick select is O(n), making it efficient for finding the kth smallest element.

**Example:**

Consider the array **[5, 3, 9, 1, 7, 2, 8, 4, 6]**.

1. Choose a pivot, let's say **5**.
2. Partition the array: **[3, 1, 2, 4] [5] [9, 7, 8, 6]**.
3. The pivot is now at position **4**. Since **4 < 7**, we focus on the right subarray **[9, 7, 8, 6]**.
4. Choose a pivot, let's say **9**.
5. Partition the right subarray: **[7, 6] [9] [8]**.
6. The pivot is now at position **7**. Since **7 > 7**, we focus on the left subarray **[7, 6]**.
7. Choose a pivot, let's say **7**.
8. Partition the left subarray: **[6] [7]**.
9. The pivot is now at position **1 + 1 + 1 = 3**. Since **3 < 7**, we focus on the right subarray **[7]**.
10. Choose a pivot, let's say **7**.
11. Now the pivot is at position **1**. Since **1 < 7**, we focus on the right subarray **[7]**.
12. Choose a pivot, let's say **7**.
13. Now the pivot is at position **1**. Since **1 < 7**, we focus on the right subarray **[7]**.
14. The pivot is the 7th smallest element.

This process continues recursively until the 7th smallest element is found.

**Q. 15: Program to print elements of an array in reverse order:**

1. #include <stdio.h>
2. ***// Function to print elements of an array in reverse order***
3. void printReverse(int arr[], int size) {
4. for (int i = size - 1; i >= 0; i--) {
5. printf("%d ", arr[i]);
6. }
7. }
8. int main() {
9. int arr[10];
10. ***// Input 10 elements into the array***
11. printf("Enter 10 elements:\n");
12. for (int i = 0; i < 10; i++) {
13. scanf("%d", &arr[i]);
14. }
15. ***// Print elements in reverse order***
16. printf("Elements in reverse order:\n");
17. printReverse(arr, 10);
18. return 0;
19. }

**Q. 16: Program to arrange binary digits such that all 0's precede all 1's:**

1. #include <stdio.h>
2. ***// Function to swap two elements in an array***
3. void swap(char \*a, char \*b) {
4. char temp = \*a;
5. \*a = \*b;
6. \*b = temp;
7. }
8. // Function to arrange binary digits
9. void arrangeBinary(char binary[], int size) {
10. int left = 0, right = size - 1;
11. while (left < right) {
12. ***// Move left pointer to the first occurrence of '1'***
13. while (binary[left] == '0' && left < right) {
14. left++;
15. }
16. ***// Move right pointer to the first occurrence of '0' from the end***
17. while (binary[right] == '1' && left < right) {
18. right--;
19. }
20. ***// Swap the elements at left and right indices***
21. if (left < right) {
22. swap(&binary[left], &binary[right]);
23. left++;
24. right--;
25. }
26. }
27. }
28. int main() {
29. char binary[] = "11010100";
30. int size = sizeof(binary) - 1; ***// Exclude null terminator***
31. printf("Original binary sequence: %s\n", binary);
32. ***// Arrange binary digits***
33. arrangeBinary(binary, size);
34. printf("Arranged binary sequence: %s\n", binary);
35. return 0;
36. }

**Q. 17. Game of Chocolates:**

1. #include <stdio.h>
2. ***// Function to calculate the minimum cost to complete the game***
3. int minCost(int chocolates[], int n) {
4. int cost = 0;
6. ***// Iterate until only one chocolate left***
7. while (n > 1) {
8. ***// Find the index of the cheaper chocolate among adjacent pairs***
9. int minIndex = 0;
10. for (int i = 1; i < n; i++) {
11. if (chocolates[i] < chocolates[minIndex]) {
12. minIndex = i;
13. }
14. }
16. ***// Pay for the cheaper chocolate and remove the expensive one***
17. cost += chocolates[minIndex];
18. for (int i = minIndex; i < n - 1; i++) {
19. chocolates[i] = chocolates[i + 1];
20. }
22. ***// Update the number of chocolates***
23. n--;
24. }
26. return cost;
27. }
28. int main() {
29. int T;
30. printf("Enter the number of games: ");
31. scanf("%d", &T); ***// Input number of games***
33. while (T--) {
34. int N;
35. printf("Enter the number of chocolates for this game: ");
36. scanf("%d", &N); ***// Input number of chocolates***
38. int chocolates[N];
39. printf("Enter the cost of each chocolate: ");
40. for (int i = 0; i < N; i++) {
41. scanf("%d", &chocolates[i]); ***// Input cost of each chocolate***
42. }
44. ***// Calculate and print the minimum cost for each game***
45. printf("Minimum cost for this game: %d\n", minCost(chocolates, N));
46. }
48. return 0;
49. }

***Q.18 & Q19. Rotate and Delete And Minimize the sum of product***

1. #include <stdio.h>
2. ***// Function to find the last element to be deleted***
3. int lastDeletedElement(int arr[], int n) {
4. int i, lastDeleted = -1;
5. for (i = 0; i < n; i++) {
6. lastDeleted = arr[i];
7. }
8. return lastDeleted;
9. }
10. ***// Function to minimize the sum of product***
11. int minimizeSumOfProduct(int A[], int B[], int n) {
12. int i, minSum = 0;
13. ***// Sorting the arrays A and B***
14. for (i = 0; i < n - 1; i++) {
15. for (int j = i + 1; j < n; j++) {
16. if (A[i] > A[j]) {
17. int temp = A[i];
18. A[i] = A[j];
19. A[j] = temp;
20. }
21. if (B[i] > B[j]) {
22. int temp = B[i];
23. B[i] = B[j];
24. B[j] = temp;
25. }
26. }
27. }
28. ***// Finding the sum of product***
29. for (i = 0; i < n; i++) {
30. minSum += A[i] \* B[n - i - 1];
31. }
32. return minSum;
33. }
34. int main() {
35. int T;
36. printf("Enter the number of test cases: ");
37. scanf("%d", &T);
38. ***// For each test case***
39. while (T--) {
40. int N;
41. printf("Enter the size of the array: ");
42. scanf("%d", &N);
43. int arr[N];
44. printf("Enter the elements of the array: ");
45. for (int i = 0; i < N; i++) {
46. scanf("%d", &arr[i]);
47. }
48. ***// Finding the last deleted element***
49. int lastDeleted = lastDeletedElement(arr, N);
50. printf("Last deleted element: %d\n", lastDeleted);
51. ***// Input arrays A and B for minimizing sum of product***
52. int A[N], B[N];
53. printf("Enter the elements of array A: ");
54. for (int i = 0; i < N; i++) {
55. scanf("%d", &A[i]);
56. }
57. printf("Enter the elements of array B: ");
58. for (int i = 0; i < N; i++) {
59. scanf("%d", &B[i]);
60. }
61. ***// Finding the minimum sum of product***
62. int minSum = minimizeSumOfProduct(A, B, N);
63. printf("Minimum sum of product: %d\n\n", minSum);
64. }
65. return 0;
66. }

**Q. 20. Let A[1 : : : n] be an array of n distinct numbers. If i < j and A[i] > A[j], then the pair (i, j) is called an inversion of A. List the five inversions of the array < 2; 3; 8; 6; 1 >. What array with elements from the set {1, 2,..........., n} has the most inversions? How many does it have? Give an algorithm that determines the number of inversions in any permutation of n elements in O(n log n) time. (Hint: Modify a known sorting algorithm).**

1. #include <stdio.h>
2. long long merge(int arr[], int temp[], int left, int mid, int right) {
3. int i = left, j = mid, k = left;
4. long long inv\_count = 0;
5. while ((i <= mid - 1) && (j <= right)) {
6. if (arr[i] <= arr[j])
7. temp[k++] = arr[i++];
8. else {
9. temp[k++] = arr[j++];
10. inv\_count += (mid - i);
11. }
12. }
13. while (i <= mid - 1)
14. temp[k++] = arr[i++];
15. while (j <= right)
16. temp[k++] = arr[j++];
17. for (i = left; i <= right; i++)
18. arr[i] = temp[i];
19. return inv\_count;
20. }
21. long long mergeSort(int arr[], int temp[], int left, int right) {
22. long long inv\_count = 0;
23. if (right > left) {
24. int mid = (right + left) / 2;
25. inv\_count += mergeSort(arr, temp, left, mid);
26. inv\_count += mergeSort(arr, temp, mid + 1, right);
27. inv\_count += merge(arr, temp, left, mid + 1, right);
28. }
29. return inv\_count;
30. }
31. int main() {
32. int arr[] = {1, 20, 6, 4, 5};
33. int n = sizeof(arr) / sizeof(arr[0]);
34. int temp[n];
35. long long inversion\_count = mergeSort(arr, temp, 0, n - 1);
36. printf("Number of inversions are %lld\n", inversion\_count);
37. return 0;
38. }

**Q. 21. One spring day on his way to university Radha found an array A. Radha likes to split arrays into several parts. This time Radha decided to split the array A into several, possibly one, new arrays so that the sum of elements in each of the new arrays is not zero. One more condition is that if we place the new arrays one after another they will form the old array A. Radha is tired now so he asked you to split the array. Help Radha! Input The first line contains single integer n (1 ≤ n ≤ 100) — the number of elements in the array A. The next line contains n integers a1, a2, ..., an ( - 103 ≤ ai ≤ 103) — the elements of the array A.**

1. #include <stdio.h>
2. int main() {
3. int n;
4. scanf("%d", &n);
5. int arr[n];
6. **// Input array elements**
7. for (int i = 0; i < n; i++)
8. scanf("%d", &arr[i]);
9. **// Find the index to split the array**
10. int split\_index = -1;
11. for (int i = 0; i < n; i++) {
12. if (arr[i] != 0) {
13. split\_index = i;
14. break;
15. }
16. }
17. **// If no non-zero element found, print "NO"**
18. if (split\_index == -1) {
19. printf("NO\n");
20. return 0;
21. }
22. **// Print "YES" and split the array**
23. printf("YES\n");
24. printf("2\n");
25. printf("1 %d\n", split\_index);
26. printf("%d %d\n", split\_index + 1, n);
27. return 0;
28. }

**Q. 22. Tom and Balls: Tom has N boxes of balls. In one minute, he can either add or remove a ball from any box. Can you tell the minimum amount of time in which he can make the gcd of all the balls divisible by K? Input The first line of the input contains T, denoting the total number of test cases. The first line of each test case contains N denoting the total number of boxes. Followed by N space-separated positive integers where ith number denotes the number of balls in the ith box.**

1. #include <stdio.h>
2. int main() {
3. int n;
4. scanf("%d", &n);
5. int arr[n];
6. **// Input array elements**
7. for (int i = 0; i < n; i++)
8. scanf("%d", &arr[i]);
9. **// Find the index to split the array**
10. int split\_index = -1;
11. for (int i = 0; i < n; i++) {
12. if (arr[i] != 0) {
13. split\_index = i;
14. break;
15. }
16. }
17. **// If no non-zero element found, print "NO"**
18. if (split\_index == -1) {
19. printf("NO\n");
20. return 0;
21. }
22. **// Print "YES" and split the array**
23. printf("YES\n");
24. printf("2\n");
25. printf("1 %d\n", split\_index);
26. printf("%d %d\n", split\_index + 1, n);
27. return 0;
28. }

**Q. 23. Minimum Sum: You are given an array of N positive elements, find the minimum value of A[i] + A[j] such that i < j. Input The first line contains N denoting the total number of elements. The next line contains N integers denoting the array elements**.

1. #include <stdio.h>
2. #include <limits.h>
3. int main() {
4. int N;
5. printf("Enter the total number of elements: ");
6. scanf("%d", &N);
8. int arr[N];
9. printf("Enter the array elements: ");
10. for (int i = 0; i < N; i++)
11. scanf("%d", &arr[i]);
13. int min\_sum = INT\_MAX;
14. for (int i = 0; i < N - 1; i++) {
15. for (int j = i + 1; j < N; j++) {
16. int sum = arr[i] + arr[j];
17. if (sum < min\_sum)
18. min\_sum = sum;
19. }
20. }
22. printf("Minimum sum: %d\n", min\_sum);
24. return 0;
25. }

**Q. 24. The sum in a Range: You are given an array of N positive integers and Q number of queries. In each query you are given two integers L and R, find the sum of elements between L and R both inclusive. Input The first line contains two integers N and Q denoting the number of elements and number of queries respectively. The next line contains N positive integers denoting the array elements. Each of the next Q lines contains two space-separated positive integers L and R.**

1. #include <stdio.h>
2. int main() {
3. int N, Q;
4. printf("Enter the number of elements and number of queries: ");
5. scanf("%d %d", &N, &Q);
7. int arr[N];
8. printf("Enter the array elements: ");
9. for (int i = 0; i < N; i++)
10. scanf("%d", &arr[i]);
12. int prefix\_sum[N + 1];
13. prefix\_sum[0] = 0;
14. for (int i = 1; i <= N; i++)
15. prefix\_sum[i] = prefix\_sum[i - 1] + arr[i - 1];
17. printf("Enter the queries (L R): \n");
18. for (int i = 0; i < Q; i++) {
19. int L, R;
20. scanf("%d %d", &L, &R);
21. int sum = prefix\_sum[R] - prefix\_sum[L - 1];
22. printf("Sum of elements between %d and %d inclusive: %d\n", L, R, sum);
23. }
25. return 0;
26. }

**Q. 25. Buy Maximum Items: You are given N items where the cost of the ith item is Ci, and you have a total budget B, find the maximum number of items you can buy within that budget. Input The first line of the input contains N denoting the total number of items and B denoting the budget. The next line contains N space-separated positive integers denoting the cost of the ith item.**

1. #include <stdio.h>
2. #include <stdlib.h>
3. ***// Function to compare two integers (for qsort)***
4. int compare(const void \*a, const void \*b) {
5. return (\*(int\*)a - \*(int\*)b);
6. }
7. ***// Function to find the maximum number of items that can be bought within the budget***
8. int max\_items(int N, long long B, int A[]) {
9. // Sort the array of item costs
10. qsort(A, N, sizeof(int), compare);
11. ***// Initialize variables***
12. long long total\_cost = 0;
13. int items\_bought = 0;
14. ***// Iterate through the sorted array***
15. for (int i = 0; i < N; i++) {
16. // If adding the current item cost doesn't exceed the budget
17. if (total\_cost + A[i] <= B) {
18. total\_cost += A[i];
19. items\_bought++;
20. } else {
21. break;
22. }
23. }
24. return items\_bought;
25. }
26. int main() {
27. int N;
28. long long B;
30. ***// Reading input***
31. printf("Enter the total number of items and the budget: ");
32. scanf("%d %lld", &N, &B);
33. int A[N];
34. printf("Enter the costs of the items: ");
35. for (int i = 0; i < N; i++) {
36. scanf("%d", &A[i]);
37. }
38. ***// Calling the function and printing the result***
39. printf("Maximum number of items you can buy: %d\n", max\_items(N, B, A));
40. return 0;
41. }