**DS 01**

**1.Search a 2D Matrix**

You are given an m x n integer matrix matrix with the following two properties:

1. Each row is sorted in non-decreasing order.
2. The first integer of each row is greater than the last integer of the previous row.
3. Given an integer target, return true if the target is in matrix or false otherwise.
4. You must write a solution in O(log(m \* n)) time complexity.

#include <stdio.h>

#include <stdbool.h>

bool searchMatrix(int *rowLen*, int *colLen*,int *matrix*[][*rowLen*], int *target*);

void main(){

    int mat[][4] = {{1,3,5,7},{10,11,16,20},{23,30,34,60}};

    if(searchMatrix(3, 4, mat, 11)) printf("True");

    else printf("False");

    return;

}

bool searchMatrix(int *rowLen*, int *colLen*,int *matrix*[][*rowLen*], int *target*) {

        int rs = 0;

        int re = *rowLen* - 1;

        int cs = 0;

        int ce = *colLen* - 1;

        while (rs <= re) {

            int m1 = (rs + re) / 2;

            if (*target* >= *matrix*[m1][cs] && *target* <= *matrix*[m1][ce]) {

                int s = 0;

                int e = ce;

                while (s <= e) {

                    int m2 = (e + s) / 2;

                    if (*matrix*[m1][m2] == *target*) {

                        return true;

                    } else if (*target* < *matrix*[m1][m2]) {

                        e = m2 - 1;

                    } else if (*target* > *matrix*[m1][m2]) {

                        s = m2 + 1;

                    }

                }

                return false;

            } else if (*target* < *matrix*[m1][0]) {

                re = m1 - 1;

            } else if (*target* > *matrix*[m1][ce]) {

                rs = m1 + 1;

            }

        }

        return false;

}

**2. Finding a Specific Product in a Sorted Inventory using Binary Search**

Imagine you are working as a sales associate at a large retail store. Customers frequently ask you about the availability of specific products, and you need to be able to quickly check if the product is in stock.

**Your task is to write a function that takes the product name as input and returns the index of the product in the sorted inventory list if it is present, or -1 if the product is not found.**

#include <stdio.h>

#include <string.h>

int search(int *n*, char \**inventory*[*n*], char \**target*);

void main(){

    char \*inventory[5] = {"Bluetooth Speaker", "Laptop Bag", "Wireless Headphones", "Wired Earbuds", "Wireless Mouse"};

    char \*target = "Laptop Bag";

    int result = search(5, inventory, target);

    printf("%d",result);

    return;

}

int search(int *n*, char \**inventory*[*n*], char \**target*){

    int l = 0;

    int r = *n*;

    while(l <= r){

        int m = (l + r) / 2;

        int cmpVal = strcmp(*inventory*[m],*target*);

        if(cmpVal == 0) return m;

        else if(cmpVal > 0){

            r = m - 1;

        } else{

            l = m + 1;

        }

    }

    return -1;

}

**4. Merge Two Sorted Arrays**

Given two sorted arrays arr1 and arr2 of sizes m and n respectively, merge them into a single sorted array without using extra space. Assume arr1 has enough space (size of m + n) to hold additional elements from arr2 if needed.

#include <stdio.h>

#include <stdlib.h>

int \*sort(int \**arr1*, int \**arr2*, int *n*, int *m*);

int main(){

    int arr1[] = {1, 3, 5, 0, 0, 0};

    int arr2[] = {2, 4, 6, 8, 10};

    int \*sortedArr = sort(arr1, arr2, 3, 5);

    for(int i = 0; i < 3 + 5; i++){

        printf("%d ", sortedArr[i]);

    }

    return 0;

}

int \*sort(int \**arr1*, int \**arr2*, int *n*, int *m*){

    int \*sortedArr = malloc(*n* + *m* + 1 \* sizeof(int));

    int k = 0;

    int i = 0;

    int j = 0;

    while(i < *n* && j < *m*){

        if(*arr1*[i] < *arr2*[j]){

            sortedArr[k] = *arr1*[i];

            i++;

        } else{

            sortedArr[k] = *arr2*[j];

            j++;

        }

        k++;

    }

    while(i < *n*){

        sortedArr[k] = *arr1*[i];

        i++; k++;

    }

    while(j < *m*){

        sortedArr[k] = *arr2*[j];

        j++; k++;

    }

    return sortedArr;

}

**11. Floor in a Sorted Array**

Given a sorted array and a value x, the floor of x is the largest element in the array smaller than or equal to x. Write efficient functions to find the floor of x. Follow the given steps to solve the problem:

1. Traverse through the array from start to end.
2. If the current element is greater than x print the previous number and break out of the loop
3. If there is no number greater than x then print the last element
4. If the first number is greater than x then print that the floor of x doesn’t exist

#include <stdio.h>

int floorSearch(int *arr*[], int *n*, int *x*);

int main(){

    int arr[] = {1, 2, 8, 10, 10, 12, 19};

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

    int x = 0;

    int result = floorSearch(arr, n, x);

    printf("%d",result);

}

int floorSearch(int *arr*[], int *n*, int *x*){

    if(*arr*[*n* - 1] < *x*)

        return *arr*[*n* - 1];

    if(*arr*[0] > *x*)

        return -1;

    for(int i = 0; i < *n*; i++){

        if(*arr*[i] > *x*){

            return *arr*[i - 1];

        }

    }

    return -1;

}

**12. Check if two arrays are equal or not.**

Given two arrays, arr1 and arr2 of equal length N, the task is to determine if the given arrays are equal or not. Two arrays are considered equal if: Both arrays contain the same set of elements. The arrangements (or permutations) of elements may be different. If there are repeated elements, the counts of each element must be the same in both arrays.

#include <stdio.h>

int isEqual(int *arr1*[], int *arr2*[], int *n*);

void main(){

    int arr1[] = {1, 2, 5, 4, 0, 2, 1};

    int arr2[] = {2, 4, 5, 0, 1, 1, 2};

    int n = sizeof(arr1)/sizeof(arr1[0]);

    if(isEqual(arr1, arr2, n))

        printf("Yes");

    else

        printf("No");

}

int isEqual(int *arr1*[], int *arr2*[], int *n*){

    int cache[100] = {0};

    for(int i = 0; i < *n*; i++)

        cache[*arr1*[i]] += 1;

    for(int i = 0; i < *n*; i++){

        if(cache[*arr2*[i]] == 0)

            return 0;

        cache[*arr2*[i]] -= 1;

    }

    return 1;

}

**13. Searching in a Large Sorted Dataset Using Jump Sort.**

A software engineer tasked with optimizing the search functionality of a large, sorted dataset containing integer values. The dataset is too large for linear search to be efficient, and he needs to ensure that search operations are performed quickly. Given that the dataset is sorted, he decided to use Jump Search to achieve this. Implement the Jump Search algorithm to find the target value in a sorted array.

#include <stdio.h>

#include <math.h>

int jumpSearch(int *arr*[], int *n*, int *key*);

void main(){

    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 26};

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

    int key = 26;

    int result = jumpSearch(arr, n, key);

    printf("%d", result);

}

int jumpSearch(int *arr*[], int *n*, int *key*){

    int s = 0;

    int e = sqrt(*n*);

    while(*arr*[e] < *key* && s < *n*){

        s = e;

        e += sqrt(*n*);

        if(e > *n*){

            e = *n* - 1;

            break;

        }

    }

    for(int i = s; i <= e; i++)

        if(*arr*[i] == *key*)

            return i;

    return -1;

}