**Array 1D**

#include <stdio.h>

void bubbleSort(int arr[],int n){

int f, t;

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

{

f = 0;

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

if(arr[j]>arr[j+1]){

t = arr[j];

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

arr[j + 1] = t;

f = 1;

}

}

if(f==0){break;}

}

}

void insertionSort(int arr[],int n){

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

int key = arr[i];

int j = i - 1;

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

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

j -= 1;

}

arr[j + 1] = key;

}

}

void selectionSort(int arr[],int n){

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

int min = i;

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

if(arr[j]<arr[min]){

min = j;

}

}

int temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

}

}  
  
int binarySearch(int arr[], int size, int target) {

int left = 0, right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target)

return mid;

else if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;   
}  
  
int linearSearch(int arr[], int size, int target) {

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

if (arr[i] == target)

return i;

}

return -1;  
}

void insert(int arr[],int n,int pos,int elem){

for (int i = n; i > pos;i--){

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

}

arr[pos] = elem;

}

void delete(int arr[],int n, int pos){

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

{

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

}

}

int main()

{

int arr[11] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

int arrLen = sizeof(arr) / sizeof(int);

//bubbleSort(arr, arrLen);

//insertionSort(arr, arrLen);

//selectionSort(arr, arrLen);

insert(arr, arrLen, 2, 10);

delete (arr, arrLen, 4);

printf("New array :");

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

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

}

return 0;

}

**Array 2D**

#include <stdio.h>

#include <stdlib.h>

// Function to add two matrices

int \*add(int A[2][2], int B[2][2], int m, int n)

{

int \*C = malloc(m \* n \* sizeof(int)); // Allocate memory for result matrix

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

{

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

{

// Correct matrix addition logic

\*(C + i \* n + j) = A[i][j] + B[i][j]; // Access elements of A and B correctly

}

}

return C;

}

int \*multiply(int A[2][2], int B[2][2], int m, int n)

{

int \*C = malloc(m \* n \* sizeof(int)); // Allocate memory for result matrix

int ne = 0;

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

{

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

{

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

{

// Correct matrix addition logic

ne += A[i][j] \* B[j][k]; // Access elements of A and B correctly

}

\*(C + i \* m + k) = ne;

ne = 0;

}

}

return C;

}

int main()

{

int A[2][2] = {{1, 2}, {3, 4}};

int B[2][2] = {{6, 7}, {8, 9}};

int \*C = multiply(A, B, 2, 2); // Call add function

// Print the result matrix

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

printf("%d ", \*(C + i \* 2 + j)); // Accessing elements of the result matrix

}

printf("\n");

}

// Free the dynamically allocated memory

free(C);

return 0;

}