۱

الف)

```
int calculateAndPrintDiagonalSums(int N, int matrix[N][N]){

    int Sum = 0;

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

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

            if ( ( i == j ) || ( i + j == N-1 ) )

                Sum += *( matrix[0] +  i * N  +  j );


    return Sum;
}
```

ب)

```
typedef int bool;
#define true 1
#define false 0

bool IsDiagonalMatrix(int N, int matrix[N][N]){

    int count = 0 , elements = 0 ;

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

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

            if ( i != j ){

                elements ++;

                if ( *(matrix[0] + i * N + j) == 0 )

                    count ++;
            }


    if ( count == elements ) return true;

    else return false;
}
```

ابتدا درون تابع main یک آرایه ای از رشته ها تعریف می شود . سپس تابع با آدرس آرایه فراخوانی می شود.

تابع function به عنوان ورودی یک اشاره گر (pointer to pointer) به ابتدای رشته های درون آرایه می گیرد (ptr) . درون تابع یک اشاره گر به یک کاراکتر تعریف می شود (ptr1) .

در خط بعدی داریم :

- ptr += sizeof(int) :

در این قسمت به محتوای اشاره گر (که اشاره گر به رشته است) به اندازه تعداد بایت های عدد صحیح (۴ بایت) اضافه می شود . پس اگر ما به اندازه ۴ بایت در آدرس پیشروی کنیم ، اشاره گر ما به عنصر ۴ ام آرایه یعنی '6' اشاره می کند .

- (ptr += sizeof(int))[-2] :

حالا با استفاده از اندیس [-2] اشاره گر را دو عنصر به عقب بر می گردانیم.

- ptr1 = (ptr += sizeof(int)) [-2] :

یعنی در نهایت ptr1 به عنصر دوم آرایه یعنی رشته "1403" اشاره می کند.

در آخر هم رشته ای که ptr1 به ابتدای آن اشاره می کند یعنی "1403" نمایش داده می شود .

```c
#include <stdlib.h>
#define SO sizeof(int)

void insert( int* addr , int* new_num , int* size){

    int temp1 , temp2;

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

        if ( ( *new_num >= *( addr + i * SO ) ) && ( *new_num <= *( addr + (i+1) * SO ) )  ){

            temp1 = *( addr + (i+1) * SO ) ;

            *( addr + (i+1) * SO ) = *new_num ;

            (*size) ++ ;

            int* addr =(int*)realloc(addr ,(*size)*SO );

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

                temp2 = *( addr + (j+1) * SO ) ;

                *( addr + (j+1) * SO ) = temp1 ;

                temp1 = temp2 ;
            }
        }
    }
}
```

الف) pointer to pointer در زبان C متغیری است که محتوای آن آدرس یک اشاره گر دیگر است . در واقع در ابتدا ما یک اشاره گر داریم که محتوای آن آدرس یک متغیر (مثلا int a = 5) است . حال اشاره گر دیگری داریم که محتوای آن آدرس اشاره گر اولی است .

این قابلیت به ما این امکان را می دهد که به طور غیر مستقیم به داده ها دسترسی داشته باشیم و آنها را تغییر دهیم . این نوع متغیر کاربرد زیادی در آرایه های چندبعدی (که آرایه ای از آرایه ها هستند) دارد .

مثال :

```c
#include <stdio.h>

int main(){

    int a = 5 ;
    int* p1 = &a ;   // p1 contains address of a
    int** p2 = &p1 ;

//p2 contains address of p1 (p2 is a pointer to pointer)

    printf("%d\n" , a);
    printf("%d\n", *p1);
    printf("%d\n", **p2);

    // Output for each one is the same (5)

    return 0 ;
}
```

ب)

مزایا :

۱) **امکان تغییر مستقیم داده :** گاهی لازم است که داخل تابعی ، مقدار متغیری را تغییر دهیم و این تغییر در متغیر اصلی که فراخوانده شده است هم اعمال شود .

۲) **کارایی بالاتر :** به جای کپی کردن کل داده ها ، تنها آدرس آن منتقل می شود و در نتیجه مصرف حافظه کاهش می یابد.

معایب :

۱) **کاهش خوانایی کد :** استفاده بیش از حد از pass by reference در کد می تواند باعث کاهش خوانایی کد شود ؛ زیرا درک تغییراتی که در متغیرها ایجاد می شود برای ما دشوارتر می شود و در نتیجه درک منطق آن برنامه سخت تر است .

۲) **خطایابی دشوارتر :** در این نوع کد ها معمولا debugging سخت تراست ؛ زیرا تغییراتی که روی یک متغیر اعمال می شود ممکن است به طور غیرمنتظره ای بر روی متغیرهای دیگر هم اثر بگذارد و منجر به بروزخطایی می شود که تشخیص آن معمولا دشوار است .

```c
#include <stdlib.h>
#define SO sizeof(int)

int* mergeAndSortArrays(int* array1, int N1, int* array2, int N2){

    int N = N1 + N2 ;

    int* result = (int*) malloc(N * SO);

    int min1 , min2;

    int status = 0;

    int counter1 = 0 , counter2 = 0;


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

        if (status){

            status = 0;
            continue;
        }

        if ( counter1 < N1)

            min1 = *(array1 + counter1 * SO);

        if ( counter2 < N2)

            min2 = *(array2 + counter2 * SO);

       // Be continuing in the next page
```

```
        if ( min1 < min2 ) {

            *(result + i * SO) = min1;
            counter1 ++;
        }

        else if ( min1 > min2 ) {

            *(result + i * SO) = min2;
            counter2 ++;
        }

        else {

            status = 1;

            *(result + i * SO) = min1;
            *(result + (i+1) * SO) = min1;

            counter1 ++;
            counter2 ++;
        }


    } // Endfor

    return result;
}
```