

## **SUBJECT**

**Internet of things (IOT)**

## **TITLE**

**Report assignment\_1**

### **By:**

<b>Loubid Emad Elboghdady</b>	<b>23012047</b>
<b>Mohamed Akram Ali</b>	<b>23011462</b>
<b>Marwan Mohamed Zain</b>	<b>23011523</b>
<b>Yahya Mahmoud Zakaria</b>	<b>23011619</b>
<b>Fares El Den Mohamed Saber</b>	<b>23012122</b>

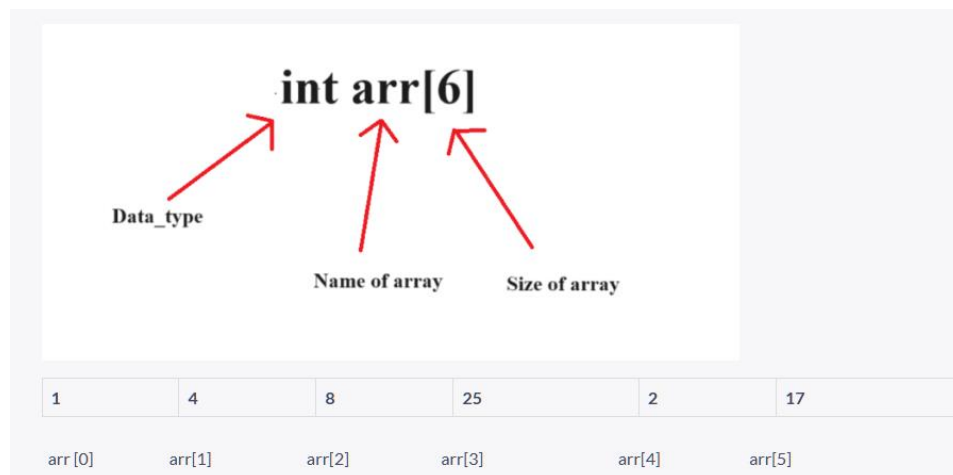
### **Supervision:**

**Eng. Mohamed Hatem**  
**Eng. Mahmoud Essam**  
**Eng. Ahmed Salah**  
**Eng. Fares**

## Question 1

### 1. Define an array in C. What are its main characteristics and uses in programming?

An array is a collection of similar types of data items at a contiguous memory location. However, the size of the array is fixed, and you need to declare its size at the time of declaration. To create an array, define the data type (like int)



```
#include <stdio.h>

int main()
{
    int myNumbers[] = {1, 4, 8, 25, 2};
    printf("%d", myNumbers[3]);

    // Outputs 25
}
```

### Why do we need Arrays?

Array stores an extensive collection of similar data types. We have only three variables, and then we can easily store them using separate names like `int var1`, `int var2`, and `int var3`, but what if we have an extensive collection of these variables? Then, we cannot assign separate names to each of them, as it would be tedious and time-consuming. Here comes the use of arrays in C, where we can store multiple data type values in a contiguous memory block. Hence, we can use an array in C when we are working with a large number of similar items.

## Main Characteristics of Arrays in C:

Characteristic	Description
Same data type	All elements in the array must be of the same type (e.g., all int or all float).
Zero-based indexing	The first element is at index 0, the second at 1, and so on.
Contiguous memory	Array elements are stored next to each other in memory.
Fixed size	Once declared, the size of the array cannot change during runtime.
Fast access	You can access any element quickly using its index (e.g., arr[2]).

## 2. Explain how memory is allocated for a one-dimensional array in C. Why is it said that arrays are stored in contiguous memory?

When you declare a one-dimensional array in C like this:

You're telling the compiler to **allocate a block of memory**

that can hold 5 integers, each integer takes 4 Bytes

```
#include <stdio.h>

int main()
{
    int arr[5];
}
```

$$4 \times 5 = 20 \text{ bytes}$$

These 20 bytes are reserved in **one continuous block** in memory — no gaps in between.

Why??

Because when the array is stored in memory, each element is placed right after the previous one, in sequential (continuous) memory addresses, this makes it easy to access elements by index.

### 3. Describe how you can initialize an array in C. Give examples for full and partial initialization. What values do uninitialized elements take?

#### 1. Full Initialization:

You provide values for every element in the array.

```
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
}
```

#### 2. Partial Initialization:

You provide values for some elements only.

```
int main()
{
    int arr[5] = {10, 20}; // only first two elements initialized
}
```

**Note:** In partial initialization, any elements

you don't specify are automatically set to 0 (zero) only if the array is initialized during declaration.

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

```
int main()
{
    int arr[5];
}
```

This is **uninitialized**, the values of the elements are **garbage values**

### 4. How do you calculate the number of elements in a statically declared array in C? Why is this technique not applicable to dynamically allocated arrays?

How do you calculate the number of elements in a statically declared array in C:

```
int main()
{
    int arr[5] = {10, 20}; // only first two elements initialized
    int length = sizeof(arr) / sizeof(arr[0]);
    printf("%d\n", length);
    // output is 5
}
```

Why doesn't it work with dynamic arrays:

Because the compiler has no idea how much memory malloc allocated — it only sees a pointer.

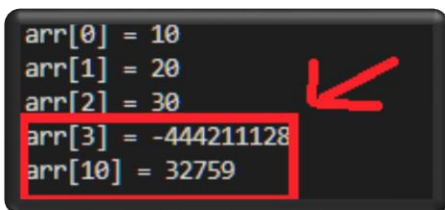
**We can create a dynamic array in C by using the following methods:**

1. Using malloc() Function
2. Using calloc() Function
3. Resizing Array Using realloc() Function
4. Using Variable Length Arrays(VLAs)
5. Using Flexible Array Members

## **5.What are the consequences of accessing an array index outside its bounds? Why doesn't the C compiler always detect this error?**

Accessing an array index outside its valid range in C results in undefined behavior. This means the program may behave in unpredictable ways, depending on what memory is accessed. C gives the programmer direct access to memory without automatic safety checks, So, the programmer must be careful to stay within valid array limits.

```
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = -444211128
arr[10] = 32759
```



```
int main()
{
    int arr[3] = {10, 20, 30};

    printf("arr[0] = %d\n", arr[0]);
    printf("arr[1] = %d\n", arr[1]);
    printf("arr[2] = %d\n", arr[2]);
    // Trying to reach elements outside size
    printf("arr[3] = %d\n", arr[3]);
    printf("arr[10] = %d\n", arr[10]);
    // give you Random value OR undefined behavior.
}
```

## 6. Differentiate between a one-dimensional and a two-dimensional array with examples. What are typical use cases for each?

### Difference Between One-Dimensional and Two-Dimensional Array

Parameters	One-Dimensional Array	Two-Dimensional Array
Basics	A one-dimensional array stores a single list of various elements having a similar data type.	A two-dimensional array stores an <i>array of various arrays</i> , or a <i>list of various lists</i> , or an <i>array of various one-dimensional arrays</i> .
Representation	It represents multiple data items in the form of a list.	It represents multiple data items in the form of a table that contains columns and rows.
Dimensions	It has only one dimension.	It has a total of two dimensions.
Parameters of Receiving	One can easily receive it in a pointer, an unsized array, or a sized array.	The parameters that receive it must define an array's rightmost dimension.
Total Size (in terms of Bytes)	Total number of Bytes = The size of array x the size of array variable or datatype.	Total number of Bytes = The size of array visible or datatype x the size of second index x the size of the first index.

#### Example of 1D Array:

```
int main()
{
    int numbers[5] = {10, 20, 30, 40, 50};
}
```

```
int main()
{
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
}
```

#### Example of 2D Array:

The array has 2 rows and 3 columns

Accessing element at row 1, column 2 would be matrix [1][2], which is 6

Elements are stored row by row in memory (row-major order)

How to Print a 2D Array: Using nested loops:

In C, a two-dimensional array is essentially an array of one-dimensional arrays.

## 7. How can you pass an array to a function in C? Explain what has passed and how this affects the original array.

In C, you can pass an array to a function by using its name only

What is passed to the function is not the entire array,

but only a pointer to the first element

**Does this affect the original array?**

**YES**

Since the function works on the same memory (because of the pointer), any change inside the function will affect the original array.

```
void printArray(int arr[], int size);

int main()
{
    int arr[3] = {1, 2, 3};
    printArray(arr, 3);
}
```

## 8. Describe the relationship between pointers and arrays in C. How can pointer arithmetic be used to access array elements?

In C, arrays and pointers are closely related.

When using pointer arithmetic with arrays in C, you're working directly with memory addresses instead of traditional indexing. Since the name of an array acts like a pointer to its first element, you can use arithmetic to move through the array by incrementing or offsetting the pointer

When you declare an array, the array name acts as a pointer to the first element of the array ex.

```
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr = arr;
    printf("Using array notation: arr[2] = %d\n", arr[2]);
    printf("Using pointer notation: *(ptr + 2) = %d\n", *(ptr + 2));
    return 0;
}
```

## **9. What limitations do arrays have in C, and how can these be addressed using structures like dynamic arrays, linked lists, or vectors in other languages?**

### **1. Fixed Size:**

- The size of an array must be known at compile time.
- You can't change its size while the program is running.

### **2. No Bounds Checking:**

- C does not check if you access an element outside the array's range.
- This may lead to undefined behavior or memory corruption.

### **3. Wasted or Insufficient Memory:**

- If you overestimate the size → you waste memory.
- If you underestimate it → you might run out of space.

### **4. No Built-in Functions:**

- No automatic resizing, inserting, or deleting elements.
- You must manage everything manually.

### **5. Hard to Insert/Delete Elements:**

- Inserting or deleting an element requires shifting others.
- It's inefficient in terms of performance.

## **How to Address These Limitations:**

### **1. Dynamic Arrays (Using malloc / realloc):**

- Allows arrays to be resized during runtime.
- Allocated using malloc() and grown with realloc().
- Solves the fixed size problem.



## 2. Linked Lists:

- Memory is allocated as needed for each element.
- Easily insert/delete items without shifting.
- Slower for direct access, but flexible for growing.

## 3. Vectors (e.g., in C++):

- Dynamic arrays with built-in functions.
- Automatically resize as needed.
- Provide functions for adding/removing elements.
- Safer and easier to use than raw arrays.

## 10. What is the enum data type used for?

In C, an enumeration (or enum) is a user defined data type that contains a set of named integer constants. It is used to assign meaningful names to integer values, which makes a program easy to read and maintain.

### Definition

An enum must be defined before we can use it in program.

```
enum enum_name {  
    n1, n2, n3, ...  
};
```



where, **n1, n2, n3, ...** are names assigned to an integer value. By default, first name **n1** is assigned **0**, **n2** is assigned **1** and the subsequent ones are **incremented by 1**.

Why use enum?

1. Gives meaningful names to integer values.
2. Improves code clarity.
3. Helps avoid using random numbers (magic numbers) in code.

## Common Uses of enum:

1. Representing fixed choices (like days, directions, states).
2. Improving code readability instead of using plain numbers.
3. Making decisions in if or switch statements.
4. Avoiding “magic numbers” in the code.

```
#include <stdio.h>
enum TrafficLight
{
    RED,
    YELLOW,
    GREEN
};

int main()
{
    enum TrafficLight light = RED;

    if (light == RED)
    {
        printf("Stop!");
    }
    else if (light == YELLOW)
    {
        printf("Wait...");
    }
    else if (light == GREEN)
    {
        printf("Go!");
    }
    return 0;
    // Output stop!
}
```

## References:

- 1- [https://www.w3schools.com/c/c\\_arrays.php](https://www.w3schools.com/c/c_arrays.php)
- 2- <https://pwskills.com/blog/array-in-c/>
- 3- <https://www.simplilearn.com/tutorials/c-tutorial/array-in-c>
- 4- <https://www.geeksforgeeks.org/c/dynamic-array-in-c/>
- 5- <https://byjus.com/gate/difference-between-one-dimensional-and-two-dimensional-array/>
- 6- <https://www.naukri.com/code360/library/passing-arrays-to-functions-in-c-cpp>
- 7- <https://www.wscubetech.com/resources/c-programming/pointer-arithmetic>
- 8- <https://www.geeksforgeeks.org/c/enumeration-enum-c/>

## Question 2:

```
#include <stdio.h>

int main()
{
    // Question2
    float balance, interest, Total_amount_due, minPayment;
    char choice;

    do
    {
        printf("Enter your balance please");
        scanf("%f", &balance);

        if (balance <= 1000)
        {
            interest = balance * 0.015;
        }
        else
        {
            interest = 1000 * 0.015 + (balance - 1000) * 0.01;
        }

        Total_amount_due = balance + interest;

        if (Total_amount_due <= 10)
        {
            minPayment = Total_amount_due;
        }

        else
        {
            minPayment = 10;
        }
    }
}
```

```

else
{
    interest = 1000 * 0.015 + (balance - 1000) * 0.01;
}

Total_amount_due = balance + interest;

if (Total_amount_due <= 10)

{
    minPayment = Total_amount_due;
}

else
{
    minPayment = Total_amount_due * 0.1;
    minPayment = (minPayment < 10) ? 10 : minPayment;
}

printf("Account Balance: %.5f\n", balance);
printf("Interest: %.5f\n", interest);
printf("Total Due: %.5f\n", Total_amount_due);
printf("Minimum Payment: %.5f\n", minPayment);
printf("Do you want to enter another balance? (y/n) ");
scanf(" %c", &choice);
} while (choice == 'y' || choice == 'Y');

return 0;
}

```

## Output:

```

Enter your balance please1900
Account Balance: 1900.00000
Interest: 24.00000
Total Due: 1924.00000
Minimum Payment: 192.39999
Do you want to enter another balance? (y/n) y
Enter your balance please500
Account Balance: 500.00000
Interest: 7.50000
Total Due: 507.50000
Minimum Payment: 50.75000
Do you want to enter another balance? (y/n) y
Enter your balance please20
Account Balance: 20.00000
Interest: 0.30000
Total Due: 20.30000
Minimum Payment: 10.00000
Do you want to enter another balance? (y/n) n

```

### Question3:

```
✓ #include <stdio.h>
#include <math.h>
// fn Prototype
int isPerfectSquare(int num);
int reverseDigits(int num);
int calculateSum(int num);
✓ int main()
{
    int number;
    do
    {
        printf("Please enter a positive number: ");
        scanf("%d", &number);

        if (number <= 0)
        {
            printf("Invalid input. Number must be positive.\n");
        }

    } while (number <= 0);

    printf("Is perfect square? %s\n", isPerfectSquare(number) ? "Yes" : "No");
    printf("Reversed number: %d\n", reverseDigits(number));
    printf("Sum of digits: %d\n", calculateSum(number));

    return 0;
}
// Fn Declration
✓ int isPerfectSquare(int num)
{
    int root = sqrt(num);
    return num == root * root;
}
```

```

    return 0;
}
// Fn Declration
int isPerfectSquare(int num)
{
    int root = sqrt(num);
    return num == root * root;
}
int reverseDigits(int num)
{
    int rev = 0;
    while (num > 0)
    {
        rev = rev * 10 + num % 10;
        num = num / 10;
    }
    return rev;
}
int calculateSum(int num)
{
    int sum = 0;
    while (num > 0)
    {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

```

## Output:

```

Please enter a positive number: -999
Invalid input. Number must be positive.
Please enter a positive number: 0
Invalid input. Number must be positive.
Please enter a positive number: 100
Is perfect square? Yes
Reversed number: 1
Sum of digits: 1

```

```

Please enter a positive number: 9805
Is perfect square? No
Reversed number: 5089
Sum of digits: 22

```

## Question4:

```

} else {

```

```

    printf("Equal price and size");
}
}
}

```

```

int main() {
    int diam1, diam2, price1, price2;
    printf("Enter price for first pizza");
    scanf("%d", &price1);
    printf("Enter diameter for first pizza");
    scanf("%d", &diam1);
    printf("Enter price for second pizza");
    scanf("%d", &price2);
    printf("Enter diameter for second pizza");
    scanf("%d", &diam2);

    double a1 = PI*(diam1/2.0)*(diam1/2.0);
    double a2 = PI*(diam2/2.0)*(diam2/2.0);
    double cost1 = price1/a1;
    double cost2 = price2/a2;

    printf("Price per square inch for first pizza is %f\n",
cost1);
    printf("Price per square inch for second pizza is %f\n",
cost2);

    if(cost1 > cost2) {
        printf("First pizza wins");
    } else if(cost2 > cost1) {
        printf("Second pizza wins");
    } else {
        if(a1 > a2) {
            printf("First pizza wins");
        } else if(a2 > a1) {
            printf("Second pizza wins");
        } else {

```

## Output:

### Case1:

```
PS D:\University\Training\IoT\Assignments> ./q4.exe
Enter price for first pizza1000
Enter diameter for first pizza1000
Enter price for second pizza5
Enter diameter for second pizza5
Price per square inch for first pizza is 0.001274
Price per square inch for second pizza is 0.254777
Second pizza wins
PS D:\University\Training\IoT\Assignments> █
```

### Case2:

```
● PS D:\University\Training\IoT\Assignments> ./q4.exe
Enter price for first pizza44
Enter diameter for first pizza3
Enter price for second pizza43
Enter diameter for second pizza54545
Price per square inch for first pizza is 6.227884
Price per square inch for second pizza is 0.000000
First pizza wins
○ PS D:\University\Training\IoT\Assignments> █
```

### Case3:

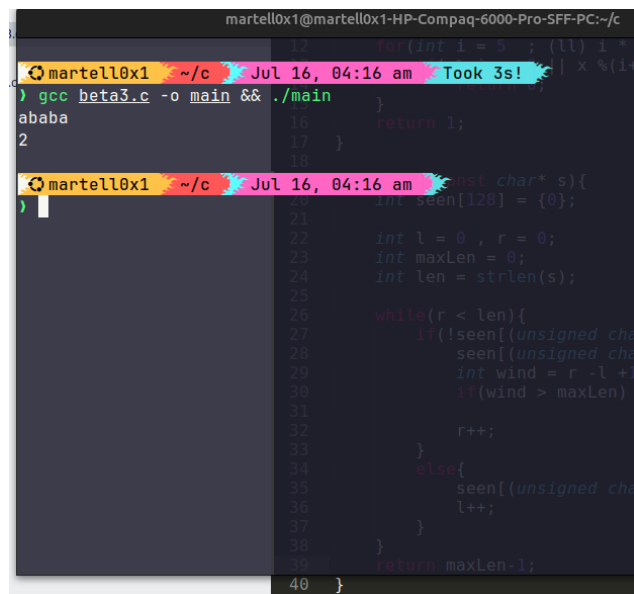
```
● PS D:\University\Training\IoT\Assignments> ./q4.exe
Enter price for first pizza5
Enter diameter for first pizza5
Enter price for second pizza5
Enter diameter for second pizza5
Price per square inch for first pizza is 0.254777
Price per square inch for second pizza is 0.254777
Equal price and size
○ PS D:\University\Training\IoT\Assignments> █
```

## Question5:

```
9  int solve(const char* s){
10     int seen[128] = {0};
11
12     int l = 0 , r = 0;
13     int maxLen = 0;
14     int len = strlen(s);
15
16     while(r < len){
17         if(!seen[(unsigned char)s[r]]){
18             seen[(unsigned char)s[r]]=1;
19             int wind = r - l + 1;
20             if(wind > maxLen) maxLen = wind;
21
22             r++;
23         }
24         else{
25             seen[(unsigned char)s[l]] = 0;
26             l++;
27         }
28     }
29     return maxLen-1;
30 }
31 int main(int argc, char const *argv[])
32 {
33     char input[MAX_LEN];
34     fgets(input , sizeof(input),stdin);
35     int re = solve(input);
36     printf("%d\n",re);
37 }
38 }
```

## Output:

### Case1:



```
martell0x1@martell0x1-HP-Compaq-6000-Pro-SFF-PC:~/c
12     for(int i = 5 ; (ll) i <
x %(1
martell0x1 ~/c Jul 16, 04:16 am Took 3s!
$ gcc beta3.c -o main && ./main
ababa
2
martell0x1 ~/c Jul 16, 04:16 am
const char* s){
int seen[128] = {0};
int l = 0 , r = 0;
int maxLen = 0;
int len = strlen(s);
while(r < len){
    if(!seen[(unsigned char)s[r]]){
        seen[(unsigned char)s[r]]=1;
        int wind = r - l + 1;
        if(wind > maxLen) maxLen = wind;
        r++;
    }
    else{
        seen[(unsigned char)s[l]] = 0;
        l++;
    }
}
return maxLen-1;
}
```



## Case2:

```
martell0x1@martell0x1-HP-Compaq-6000-Pro-SFF-PC:~/c
martell0x1 ~/c Jul 16, 04:17 am Took 6s!
) gcc beta3.c -o main && ./main
lllllllo
2

martell0x1 ~/c Jul 16, 04:17 am Took 3s!
) [
]

12     for(int i = 5 ; (ll) i *
13         % i == 0 || x %(i+
14         return 0;
15     }
16     return 1;
17 }
18
19 const char* s){
20     int seen[128] = {0};
21
22     int l = 0 , r = 0;
23     int maxLen = 0;
24     int len = strlen(s);
25
26     while(r < len){
27         if(!seen[(unsigned cha
28             seen[(unsigned cha
29             int wind = r -l +1
30             if(wind > maxLen)
31
32             r++;
33         }
34         else{
35             seen[(unsigned cha
36             l++;
37         }
38     }
39     return maxLen-1;
40 }
```

## Case3:

```
martell0x1@martell0x1-HP-Compaq-6000-Pro-SFF-PC:~/c
martell0x1 ~/c Jul 16, 04:18 am
) gcc beta3.c -o main && ./main
kks
2

martell0x1 ~/c Jul 16, 04:18 am
) [
]

12     for(int i = 5 ; (ll) i *
13         % i == 0 || x %(i+
14         return 0;
15     }
16     return 1;
17 }
18
19 const char* s){
20     int seen[128] = {0};
21
22     int l = 0 , r = 0;
23     int maxLen = 0;
24     int len = strlen(s);
25
26     while(r < len){
27         if(!seen[(unsigned cha
28             seen[(unsigned cha
29             int wind = r -l +1
30             if(wind > maxLen)
31
32             r++;
33         }
34         else{
35             seen[(unsigned cha
36             l++;
37         }
38     }
39     return maxLen-1;
40 }
```

## Question 6: String to Integer (myAtoi)

- converts a string to an integer by

- skipping leading spaces
- handling an optional sign
- digit characters into a number
- clamping the result within integer limits

```
C myAtoi.c > main()
1  #include <stdio.h>
2  #include <limits.h>
3  int myAtoi(char* str) {
4      int result = 0;
5      int sign = 1;
6
7      while (*str == ' ') {
8          str++;
9      }
10     if (*str == '-') {
11         sign = -1;
12         str++;
13     } else if (*str == '+') {
14         str++;
15     }
16     while (*str >= '0' && *str <= '9') {
17         result = result * 10 + (*str - '0');
18         if (sign == 1 && result > INT_MAX) return INT_MAX;
19         if (sign == -1 && -result < INT_MIN) return INT_MIN;
20         str++;
21     }
22     return sign * result;
23 }
24
```

## Output:

```
26  int main() {  
27      char str1[] = "42";  
28      int result = myAtoi(str1);  
29      printf("The converted integer is: %d\n", result);  
30      char str2[] = "-042";  
31      result = myAtoi(str2);  
32      printf("The converted integer is: %d\n", result);  
33      char str3[] = "1337cod3";  
34      result = myAtoi(str3);  
35      printf("The converted integer is: %d\n", result);  
36      char str4[] = "0-1";  
37      result = myAtoi(str4);  
38      printf("The converted integer is: %d\n", result);  
39      char str5[] = "words and 987";  
40      result = myAtoi(str5);  
41      printf("The converted integer is: %d\n", result);
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
The converted integer is: 42  
The converted integer is: -42  
The converted integer is: 1337  
The converted integer is: 0  
The converted integer is: 0
```

## Question7:

Median of 2 sorted arrays in  $O(\log(n+m))$

Some handwritten explanations:

Question 7 explanation:

Median of two sorted array

→ we can determine the size of the first partition using the other.

Example with odd size:

$-\infty$  | 1 | 2 | 3 | 5 | 5 | 6 | 7 | 8 |  $\infty$  total size = 13  
half = 6

$-\infty$  | 1 | 2 | 3 | 4 | 5 |  $\infty$   
L                      M                      R  
 $M = \frac{(5-0)}{2}$

∴ size of first partition =  
half-size of second partition  
= 6 - 3 = 3

∴ merged correctly median =  $\min(4, 5) = 4$

1 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 7 | 8  
6                      6

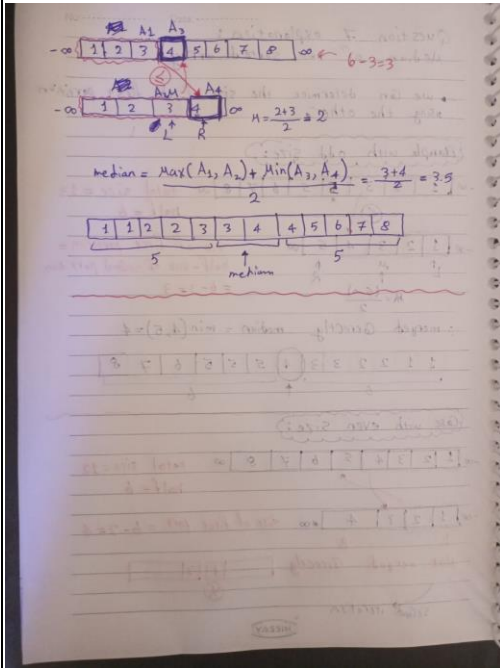
Case with even size:

$-\infty$  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  $\infty$  total size = 12  
half = 6

$-\infty$  | 1 | 2 | 3 | 4 |  $\infty$  size of first part. = 6 - 2 = 4  
L                      R

Not merged correctly [ ... | 4 | 3 | ... ]

second iteration



```
int max(int a, int b) {
    return (a > b) ? a : b;
}

int min(int a, int b) {
    return (a < b) ? a : b;
}

double findMedianSortedArrays(int* nums1, int* nums2, int size1,
int size2) {
    // make nums2 the smallest array
    if(size1 > size2) {
        return findMedianSortedArrays(nums2, nums1, size2, size1);
    }

    int totalSize = size1+size2;
    int half = (totalSize+1)/2;

    int l = 0, r = size1;
    while(l <= r) {
        int m1 = (l+r)/2;
        int m2 = half - m1;

        int a1 = m1 > 0 ? nums1[m1-1] : INT_MIN;
        int a2 = m2 > 0 ? nums2[m2-1] : INT_MIN;
        int a3 = m1 < size1 ? nums1[m1] : INT_MAX;
        int a4 = m2 < size2 ? nums2[m2] : INT_MAX;

        if(a1 <= a4 && a2 <= a3) {
            if(totalSize%2 == 0) return (max(a1, a2) + min(a3,
a4))/2.0;
            return max(a1, a2);
        }else if(a1 > a4) {
            r = m1-1;
        }else {
            l = m1+1;
        }
    }
    return 0.0;
}

int main() {
    int nums1[] = {1,3}, nums2[] = {2};

    int size1 = (sizeof(nums1)/sizeof(nums1[0]));
    int size2 = (sizeof(nums2)/sizeof(nums2[0]));

    printf("%f", findMedianSortedArrays(nums1, nums2, size1,
size2));
}
```

## Output:

### Case1:

```
44
45 int main() {
46     int nums1[] = {1,3}, nums2[] = {2};
47
48     int size1 = (sizeof(nums1)/sizeof(nums1[0]));
49     int size2 = (sizeof(nums2)/sizeof(nums2[0]));
50
51     printf("%f", findMedianSortedArrays(nums1, nums2, size1, size2));
52 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS POSTMAN CONSOLE

```
2.000000
PS D:\University\Training\IoT\Assignments> gcc q7.c -o q7
PS D:\University\Training\IoT\Assignments> ./q7.exe
2.000000
PS D:\University\Training\IoT\Assignments>
```

### Case2:

```
45 int main() {
46     int nums1[] = {1,2}, nums2[] = {3, 4};
47
48     int size1 = (sizeof(nums1)/sizeof(nums1[0]));
49     int size2 = (sizeof(nums2)/sizeof(nums2[0]));
50
51     printf("%f", findMedianSortedArrays(nums1, nums2, size1, size2));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS POSTMAN CONSOLE

```
PS D:\University\Training\IoT\Assignments> ./q7.exe
2.000000
PS D:\University\Training\IoT\Assignments> gcc q7.c -o q7
PS D:\University\Training\IoT\Assignments> ./q7.exe
2.500000
PS D:\University\Training\IoT\Assignments>
```

### Case3:

```
45 int main() {
46     int nums1[] = {-5, -3, 0}, nums2[] = {-4, -2, 1};
47
48     int size1 = (sizeof(nums1)/sizeof(nums1[0]));
49     int size2 = (sizeof(nums2)/sizeof(nums2[0]));
50
51     printf("%f", findMedianSortedArrays(nums1, nums2, size1, size2));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS POSTMAN CONSOLE

```
PS D:\University\Training\IoT\Assignments> ./q7.exe
2.000000
PS D:\University\Training\IoT\Assignments> gcc q7.c -o q7
PS D:\University\Training\IoT\Assignments> ./q7.exe
2.500000
PS D:\University\Training\IoT\Assignments> gcc q7.c -o q7
PS D:\University\Training\IoT\Assignments> ./q7.exe
-2.500000
PS D:\University\Training\IoT\Assignments>
```

## Question8: Find First and Last Position of Element in Sorted Array

Solution: To find first and last positions of a target in a sorted array, use modified binary searches that keep going left or right after finding the target to ensure you get the true first and last occurrences.

```
FindFirstLastTarget.c > main()
1  #include <stdio.h>
2
3  void FindFLTTarget(int* nums, int numsSize, int target, int* output) {
4      int left, right, mid;
5      output[0] = -1;
6      output[1] = -1;
7      left = 0;
8      right = numsSize - 1;
9      while (left <= right) {
10         mid = left + (right - left) / 2;
11         if (nums[mid] < target) {
12             left = mid + 1;
13         } else {
14             right = mid - 1;
15         }
16         if (nums[mid] == target) {
17             output[0] = mid;
18         }
19     }
20     if (output[0] == -1) {
21         return;
22     }
23     left = 0;
24     right = numsSize - 1;
25     while (left <= right) {
26         mid = left + (right - left) / 2;
27         if (nums[mid] > target) {
28             right = mid - 1;
29         } else {
30             left = mid + 1;
31         }
32         if (nums[mid] == target) {
33             output[1] = mid;
34         }
35     }
36 }
37
```

## Output:

```
39 int main(){
40     int nums[] = {1, 2, 2, 3, 4, 5, 6, 6, 7, 8, 8, 8, 9};
41     int numsSize = sizeof(nums) / sizeof(nums[0]);
42     int target = 8;
43     int output[2];
44     FindFLTarget(nums, numsSize, target, output);
45     printf("First and Last positions of target %d are: [%d, %d]\n", target, output[0], output[1]);
46     int nums2[] = {};
47     numsSize = sizeof(nums2) / sizeof(nums2[0]);
48     target = 6;
49     FindFLTarget(nums2, numsSize, target, output);
50     printf("First and Last positions of target %d are: [%d, %d]\n", target, output[0], output[1]);
51     int nums3[] = {1, 2, 3, 4, 5};
52     numsSize = sizeof(nums3) / sizeof(nums3[0]);
53     target = 7;
54     FindFLTarget(nums3, numsSize, target, output);
55     printf("First and Last positions of target %d are: [%d, %d]\n", target, output[0], output[1]);
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

PS M:\gitwork>

PS M:\gitwork> & 'c:\Users\Mohamed Akram\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\debugAdapters\t  
-qbxsvi0g.apq' '--stdout=Microsoft-MIEngine-Out-xusjvhpv.ze5' '--stderr=Microsoft-MIEngine-Error-dphwkgfb.13c' '  
s64\mingw64\bin\gdb.exe' '--interpreter=mi'

First and Last positions of target 8 are: [9, 11]

First and Last positions of target 6 are: [-1, -1]

First and Last positions of target 7 are: [-1, -1]

## Question9: Search Insert Position

```
1  #include <stdio.h>
2
3  int searchInsert(int* nums, int numsSize, int target) {
4      int left = 0, right = numsSize - 1;
5
6      while (left <= right) {
7          int mid = left + (right - left) / 2;
8
9          if (nums[mid] == target)
10             return mid;
11         else if (target < nums[mid])
12             right = mid - 1;
13         else
14             left = mid + 1;
15     }
16
17     return left;
18 }
```



## Output:

```
int main() {
    int nums[] = {1, 3, 5, 6};
    int target = 5;
    int result = searchInsert(nums, sizeof(nums) / sizeof(nums[0]), target);
    printf("Insert position for %d is: %d\n", target, result);

    target = 2;
    result = searchInsert(nums, sizeof(nums) / sizeof(nums[0]), target);
    printf("Insert position for %d is: %d\n", target, result);

    target = 7;
    result = searchInsert(nums, sizeof(nums) / sizeof(nums[0]), target);
    printf("Insert position for %d is: %d\n", target, result);

    return 0;
}
```

MS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
gitwork> & 'c:\Users\Mohamed Akram\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\bin\cpptools.exe' --stdin=Microsoft-MIEngine-In-skgc5qz5.hqp' --stdout=Microsoft-MIEngine-Out-f5ou2jsb
/bv' --pid=Microsoft-MIEngine-Pid-lqg5nze0.qsq' --dbgExe=C:\msys64\mingw64\bin\gdb.exe
Insert position for 5 is: 2
Insert position for 2 is: 1
Insert position for 7 is: 4
gitwork> |
```

## Question 10: Add Binary

```
int max(int a, int b) {
    return (a > b) ? a : b;
}

int main() {
    char a[] = "110";
    char b[] = "100000000000000001";

    // -1 for null character \0
    int s1 = sizeof(a)/sizeof(a[0]) -1;
```

```

int s2 = sizeof(b)/sizeof(b[0]) -1;
int end = max(s1, s2);

// handle case of different string size
char aa[end + 1], bb[end + 1], c[end + 1];
int carry[end + 1];

memset(aa, '0', end);
memset(bb, '0', end);
memset(c, '0', end);
memset(carry, 0, sizeof(carry));

aa[end] = '\0';
bb[end] = '\0';
c[end] = '\0';

for(int j = end-1, x = s1-1, y = s2-1; j >= 0; j--) {
    if(x >= 0) aa[j] = a[x--];

```

```

    if(y >= 0) bb[j] = b[y--];
}

printf(aa);
printf("\n");
printf(bb);
printf("\n-----\n");

for(int i = end-1; i >= 0; i--) {
    // 48 for ascii
    int sum = aa[i] - '0' + bb[i] - '0' + carry[i];
    c[i] = sum%2 + '0';

    if(i > 0) carry[i-1] = sum/2;
    else carry[0] = sum/2;
}

```

```

}

// merge all in one string with leading or not
if(carry[0]) {
    char ans[end+2];
    ans[0] = '1';
    for(int i = 0; i < end; i++) ans[i+1] = c[i];
    ans[end+1] = '\0';
    printf(ans);
}else {
    printf(c);
}
}

```

## Output:

### Case 1:

```

11
01
-----
100

```

### Case 2:

```

1010
1011
-----
10101

```

### Case 3:

```

0000000000000000110
100000000000000001
-----
1000000000000000111

```

## Question12: (bonus) String is palindrome

```
C isPalindromeString.c > main()
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <string.h>
4
5  bool isPalindrome(char* s) {
6      int left = 0;
7      int right = strlen(s) - 1;
8
9      while (left < right) {
10         if (s[left] != s[right]) {
11             return false;
12         }
13         left++;
14         right--;
15     }
16     return true;
17 }
18
19 int main(){
20     char str1[] = "racecar";
21     char str2[] = "ooooopoo";
22     printf("Is \"%s\" a palindrome? %s\n", str1, isPalindrome(str1) ? "Yes" : "No");
23     printf("Is \"%s\" a palindrome? %s\n", str2, isPalindrome(str2) ? "Yes" : "No");
24
25
26     return 0;
--
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

s64\mingw64\bin\gdb.exe' '--interpreter=mi'
Is "racecar" a palindrome? Yes
Is "ooooopoo" a palindrome? No
```

## [Bonus] Question 11: Word Search in a Grid

```
#include <stdbool.h>

#define MAX_LEN 50001
#define MAX_M 6
#define MAX_N 6
#define MAX_WORD_LEN 16

typedef long long ll;

int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

bool dfs(char board[MAX_M][MAX_N], int m, int n, int i, int j, const char* word, int idx, bool visited[MAX_M][MAX_N]) {
    if (word[idx] == '\0') return true;

    if (i < 0 || i >= m || j < 0 || j >= n || visited[i][j] || board[i][j] != word[idx])
        return false;

    visited[i][j] = true;

    for (int d = 0; d < 4; d++) {
        int ni = i + dx[d];
        int nj = j + dy[d];
        if (dfs(board, m, n, ni, nj, word, idx + 1, visited))
            return true;
    }

    visited[i][j] = false;
    return false;
}

bool exist(char board[MAX_M][MAX_N], int m, int n, const char* word) {
    bool visited[MAX_M][MAX_N] = { false };

```

```
bool exist(char board[MAX_M][MAX_N], int m, int n, const char* word) {
    bool visited[MAX_M][MAX_N] = { false };

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (dfs(board, m, n, i, j, word, 0, visited))
                return true;
        }
    }
    return false;
}

int main(int argc, char const *argv[])
{
    int m, n;
    char board[MAX_M][MAX_N];
    char word[MAX_WORD_LEN];

    scanf("%d %d", &m, &n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            scanf(" %c", &board[i][j]);
    scanf("%s", word);

    bool found = exist(board, m, n, word);

    printf("%s", found ? "yes:" : "no");
}

```

## Case 1:

```
martell0x1@martell0x1-HP-Compaq-6000-Pro-SFF-PC:~/c
martell0x1 ~/c Jul 16, 04:31 am Took 21s!
) gcc beta3.c -o main && ./main
3 4
w o r d
a b c d
e f g h
word
yes%

martell0x1 ~/c Jul 16, 04:31 am Took 11s!
)

27         int nj = j + dy[d];
28         if (dfs(board, m, n, ni, nj, word, idx + 1, vis
31
32         visited[i][j] = false;
33         return false;
34     }
35
36     bool exist(char board[MAX_M][MAX_N], int m, int n, cons
37     bool visited[MAX_M][MAX_N] = { false };
38
39     for (int i = 0; i < m; i++) {
40         for (int j = 0; j < n; j++) {
41             if (dfs(board, m, n, i, j, word, 0, visited
42                 return true;
43             }
44         }
45         return false;
46     }
47
48     int main(int argc, char const *argv[])
49     {
50         int m, n;
51         char board[MAX_M][MAX_N];
52         char word[MAX_WORD_LEN];
53
54         scanf("%d %d", &m, &n);
55         for (int i = 0; i < m; i++)
56             for (int j = 0; j < n; j++)
```

## Case 2:

```
martell0x1@martell0x1-HP-Compaq-6000-Pro-SFF-PC:~/c
martell0x1 ~/c Jul 16, 04:32 am Took 11s!
) gcc beta3.c -o main && ./main
3 4
a b c e
f k l o
o r d s
ford
yes%

martell0x1 ~/c Jul 16, 04:33 am Took 36s!
)

27         int nj = j + dy[d];
28         if (dfs(board, m, n, ni, nj, word, idx + 1, vis
31
32         visited[i][j] = false;
33         return false;
34     }
35
36     bool exist(char board[MAX_M][MAX_N], int m, int n, cons
37     bool visited[MAX_M][MAX_N] = { false };
38
39     for (int i = 0; i < m; i++) {
40         for (int j = 0; j < n; j++) {
41             if (dfs(board, m, n, i, j, word, 0, visited
42                 return true;
43             }
44         }
45         return false;
46     }
47
48     int main(int argc, char const *argv[])
49     {
50         int m, n;
51         char board[MAX_M][MAX_N];
52         char word[MAX_WORD_LEN];
53
54         scanf("%d %d", &m, &n);
55         for (int i = 0; i < m; i++)
56             for (int j = 0; j < n; j++)
```

## Case 3:

```
martell0x1@martell0x1-HP-Compaq-6000-Pro-SFF-PC:~/c
martell0x1 ~/c Jul 16, 04:33 am Took 36s!
) gcc beta3.c -o main && ./main
3 4
a b c d
e f g h
h y j k
see
no%

martell0x1 ~/c Jul 16, 04:33 am Took 12s!
)

27         int nj = j + dy[d];
28         if (dfs(board, m, n, ni, nj, word, idx + 1, vis
31
32         visited[i][j] = false;
33         return false;
34     }
35
36     bool exist(char board[MAX_M][MAX_N], int m, int n, cons
37     bool visited[MAX_M][MAX_N] = { false };
38
39     for (int i = 0; i < m; i++) {
40         for (int j = 0; j < n; j++) {
41             if (dfs(board, m, n, i, j, word, 0, visited
42                 return true;
43             }
44         }
45         return false;
46     }
47
48     int main(int argc, char const *argv[])
49     {
50         int m, n;
51         char board[MAX_M][MAX_N];
52         char word[MAX_WORD_LEN];
53
54         scanf("%d %d", &m, &n);
55         for (int i = 0; i < m; i++)
56             for (int j = 0; j < n; j++)
```