

## Camp3



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ


وَأَنْ لَّيْسَ لِلْإِنْسَانِ إِلَّا مَا سَعَى (39) وَأَنَّ سَعْيَهُ سَوْفَ يُرَى (40) ثُمَّ يُجْزَاهُ الْجَزَاءُ الْأَوْفَى (41) وَأَنَّ إِلَى رَبِّكَ الْمُنْتَهَى (42)


## Algorithms & Data Structure & SQL Queries

### Algorithms

#### Sorting Algorithms :

| the common types of sorting algorithms:

1. **Bubble Sort** ⇒ go to camp2
2. **Selection Sort** ⇒ go to camp2
3. **Insertion Sort** ⇒ go to camp2
4. **Merge Sort** ⇒ 
5. **Quick Sort** ⇒ go to camp2

- 6. Heap Sort ⇒ 
- 7. Counting Sort ⇒ wait in camp4
- 8. Radix Sort ⇒ wait in camp4
- 9. Bucket Sort ⇒ wait in camp5
- 10. Shell Sort ⇒ wait in camp5

## Merge Sort

### Definition:

**Merge Sort** ("divide and conquer") هي خوارزمية ترتيب تعتمد على مبدأ (الفرز بالدمج) ، يعني بتقسم المشكلة لأجزاء صغيرة وترتب كل جزء لوحده، وبعدين تدمجهم في النهاية بشكل مرتب.

### الفكرة الأساسية:

#### 1. تقسيم (Divide):

نقسم المصفوفة إلى نصفين (نكرر ده لحد ما نوصل لعناصر فردية أو مصفوفات صغيرة جدًا من عنصر واحد).

#### 2. حل كل جزء (Conquer):

نرتب كل جزء صغير (لو فيه عنصر واحد فهو مرتب أصلاً).

#### 3. دمج (Merge):

ندمج الأجزاء الصغيرة مع بعض بطريقة مرتبة، ونكرر الدمج لحد ما نرجع مصفوفة كاملة مرتبة.

### مثال توضيحي:

لو عندك المصفوفة

[6, 3, 8, 5]

#### 1. نقسم:

و [5, 8] [6, 3]

#### 2. نقسم ثاني:

[6] [3] [5] [8] g

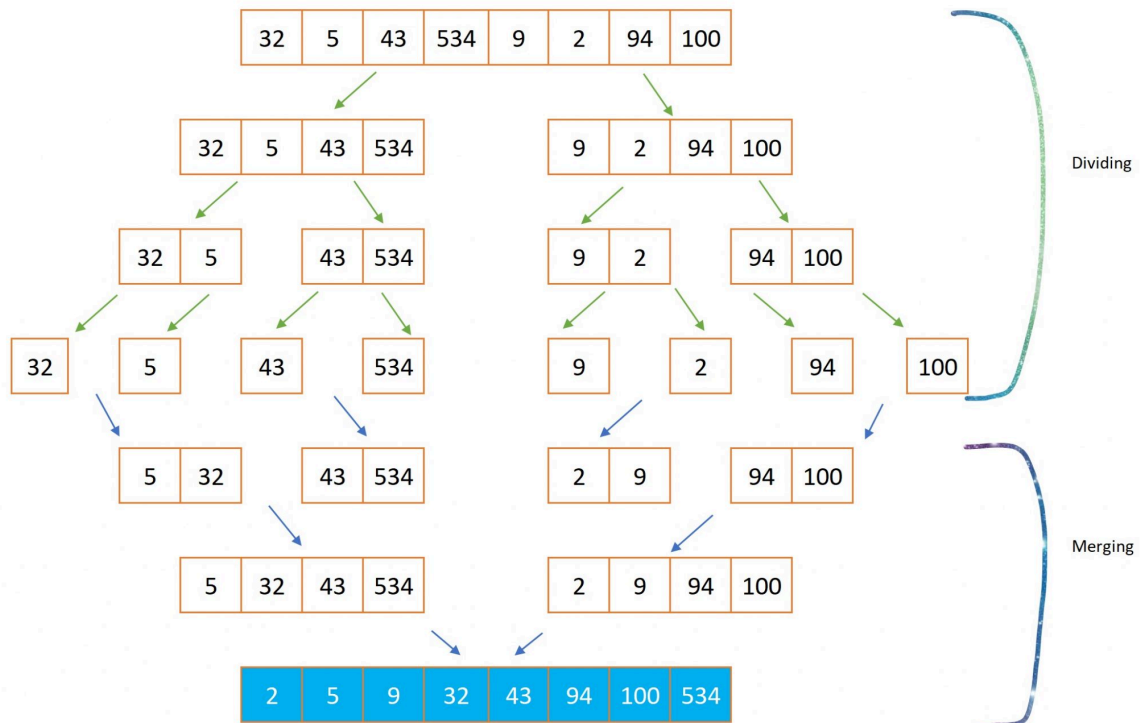
### 3. ندمج كل زوج بترتيب:

[3, 6] [8, 5] g

### 4. ندمج النتيجة:

[3, 5, 6, 8]

| الحالة            | Time Complexity | Space Complexity |
|-------------------|-----------------|------------------|
| أفضل حالة (Best)  | $O(n \log n)$   | $O(n)$           |
| الحالة المتوسطة   | $O(n \log n)$   | $O(n)$           |
| أسوأ حالة (Worst) | $O(n \log n)$   | $O(n)$           |



## Code :

```
namespace Camp3
{

    class Program
    {

        // Merges two subarrays of []arr.
        // First subarray is arr[l..m]
        // Second subarray is arr[m+1..r]
        static void merge(int[] arr, int l, int m, int r)
        {
            // Find sizes of two
            // subarrays to be merged
            int n1 = m - l + 1;
            int n2 = r - m;

            // Create temp arrays
            int[] L = new int[n1];
            int[] R = new int[n2];
            int i, j;

            // Copy data to temp arrays
            for (i = 0; i < n1; ++i)
                L[i] = arr[l + i];
            for (j = 0; j < n2; ++j)
                R[j] = arr[m + 1 + j];

            // Merge the temp arrays

            // Initial indexes of first
            // and second subarrays
            i = 0;
            j = 0;

            // Initial index of merged
            // subarray array
            int k = l;
            while (i < n1 && j < n2)
            {
```

```

        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy remaining elements
    // of L[] if any
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy remaining elements
    // of R[] if any
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Main function that
// sorts arr[l..r] using
// merge()
static void mergeSort(int[] arr, int l, int r)
{
    if (l < r)
    {
        // Find the middle point
        int m = l + (r - l) / 2;
    }
}

```

```

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

// A utility function to
// print array of size n
static void printArray(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n; ++i)
        Console.Write(arr[i] + " ");
    Console.WriteLine();
}

// Driver code
public static void Main(String[] args)
{
    int[] arr = { 12, 11, 13, 5, 6, 7 };
    Console.WriteLine("Given array is");
    printArray(arr);
    mergeSort(arr, 0, arr.Length - 1);
    Console.WriteLine("\nSorted array is");
    printArray(arr);
}
}
}

```

## Heap Sort

### ✓ Definition:

**Heap Sort** (تحديداً: Max-Heap) تعتمد على بنية بيانات اسمها (Sorting Algorithm) هي خوارزمية ترتيب (Min-Heap أو Max-Heap).

بتشتغل الخوارزمية على مرحلتين رئيسيتين

1. بحيث يكون العنصر الأكبر (أو Min-Heap أو Max-Heap) تحويل المصفوفة إلى **بناء هيب (Heap Building)** (الأصغر) في الجذر.
2. الجذر مع آخر عنصر، ثم إعادة هيكلة الهيب بدون العنصر المأخوذ، وتكرار (Swap) يتم تبادل **الفرز (Sorting)** العملية.

### فكرة عملها ببساطة:

1. (يعني شجرة مرتبة فيها كل أب أكبر أو أصغر من أولاده) **Heap** بتحوّل المصفوفة لـ.
2. وتحطه في آخر مكان (**Heap** حسب نوع ال) بتأخذ العنصر الأكبر/الأصغر.
3. **بتكرر العملية** لحد ما المصفوفة كلها تبقى مرتبة.

### Heap إيه هو الـ

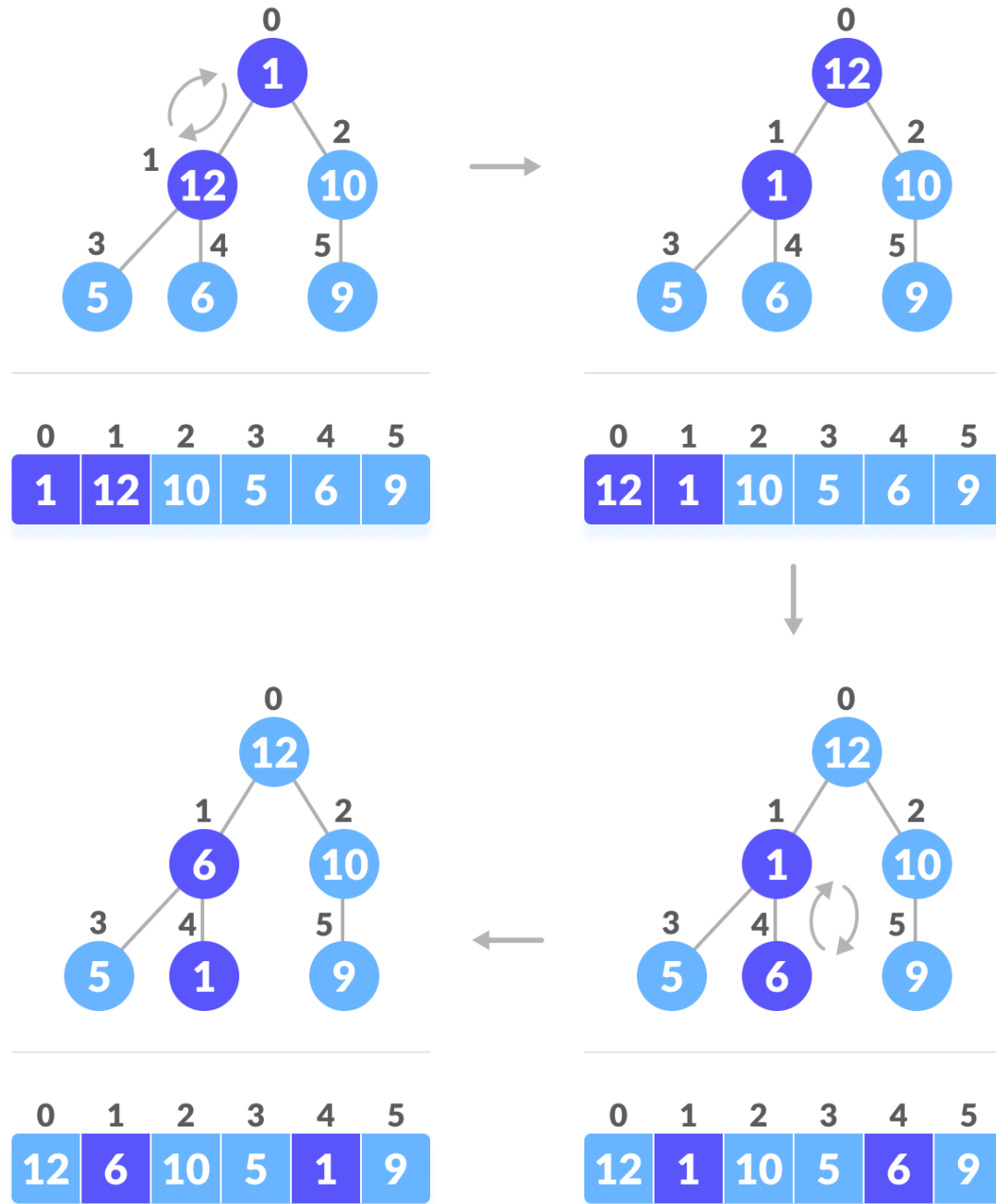
- (Binary Tree) هو شكل خاص من الشجرة الثنائية
- أكبر من أولاده (Node) كل عنصر **Max-Heap** في
- كل عنصر أصغر من أولاده **Min-Heap** في

### Heap Sort خطوات:

1. من المصفوفة **Max-Heap** ابني.
2. **بذل** العنصر الأول (الأكبر) مع آخر عنصر.
3. (اعتبر إن العنصر الأخير خارج اللعبة) **Heap** **قلل حجم الـ**.
4. **Max-Heap** الجزء اللي لسه في اللعبة علشان ترجع (**Heapify**) **أعد تنظيم**.
5. كرر الخطوات دي لحد ما المصفوفة تترتب.

| الحالة (Case)       | Time Complexity | Space Complexity |
|---------------------|-----------------|------------------|
| <b>Best Case</b>    | $O(n \log n)$   | $O(1)$           |
| <b>Average Case</b> | $O(n \log n)$   | $O(1)$           |
| <b>Worst Case</b>   | $O(n \log n)$   | $O(1)$           |

$i = 0 \rightarrow \text{heapify}(\text{arr}, 6, 0)$







show this video to understand ⇒ <https://youtu.be/mdOaBGxxF6M?si=fT7TPKjCntLeyQ1L>

## Code:

```
namespace Camp3
{

    using System;

    class Program
    {
        // To heapify a subtree rooted with node i
        // which is an index in arr[].
        static void Heapify(int[] arr, int n, int i)
        {

            // Initialize largest as root
            int largest = i;

            // left index = 2*i + 1
            int l = 2 * i + 1;

            // right index = 2*i + 2
            int r = 2 * i + 2;

            // If left child is larger than root
            if (l < n && arr[l] > arr[largest])
            {
                largest = l;
            }

            // If right child is larger than largest so far
            if (r < n && arr[r] > arr[largest])
            {
                largest = r;
            }

            // If largest is not root
            if (largest != i)
```

```

    {
        int temp = arr[i]; // Swap
        arr[i] = arr[largest];
        arr[largest] = temp;

        // Recursively heapify the affected sub-tree
        Heapify(arr, n, largest);
    }
}

// Main function to do heap sort
static void HeapSortArray(int[] arr)
{
    int n = arr.Length;

    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
    {
        Heapify(arr, n, i);
    }

    // One by one extract an element from heap
    for (int i = n - 1; i > 0; i--) //max element in front of max heap array
    {

        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        // Call max heapify on the reduced heap
        Heapify(arr, i, 0); //0 here because large element in front of array
        // use i here not n because element put in first location
    }
}

// A utility function to print array of size n
static void PrintArray(int[] arr)
{
    foreach (int value in arr)
    {
        Console.Write(value + " ");
    }
}

```

```
        Console.WriteLine();
    }

    // Driver's code
    public static void Main(string[] args)
    {
        int[] arr = { 9, 4, 3, 8, 10, 2, 5 };
        HeapSortArray(arr);
        Console.WriteLine("Sorted array is ");
        PrintArray(arr);
    }
}

}
```

## Some Problem Solving :



## Problem1

### Check If a number is prime

الرقم الاول هو الذي يقبل القسمة علي نفسه وعلي الواحد فقط مثلا  
2,3,5,7,11,.....

### Code :

```
namespace Camp3
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.Write("Enter Number : ");
            int n = Convert.ToInt32(Console.ReadLine());

            bool isPrime = isPrimeNumber(n);

            if (isPrime)
            {
                Console.WriteLine($"{n} is a prime number.");
            }
            else
            {
                Console.WriteLine($"{n} is not a prime number.");
            }
        }

        private static bool isPrimeNumber(int n)
        {
            if (n <= 1) return false;
            if (n == 2) return true;
            if (n % 2 == 0) return false;

            for (int i = 3; i <= Math.Sqrt(n); i += 2)
            {
                if (n % i == 0)
                    return false;
            }
        }
    }
}
```

```
    }  
    return true;  
  }  
}  
  
}
```



## Problem 2

### Reverse Digits of integer

Code:

```
namespace Camp3
{
    class Program
    {
        public static void Main(string[] args)
        {
            int n = 47192;
            Console.WriteLine($"The original number is {n}.");
            int reversed = ReverseDigits(n);
            Console.WriteLine($"The reverse of {n} is {reversed}.");
        }

        static int ReverseDigits(int number)
        {
            int revnum = 0;
            while(number > 0)
            {
                revnum= revnum * 10 + number % 10;
                number = number / 10;
            }
            return revnum;
        }
    }
}
```



### Problem 3

**Compute the sum of digits in a number**

**Code:**

```
namespace Camp3
{

    class Program
    {

        public static void Main(string[] args)
        {
            int n = 47192;
            Console.WriteLine("Number : "+n);
            int sum = SumOfDigits(n);
            Console.WriteLine("Sum of digits : " + sum);

        }

        static int SumOfDigits(int number)
        {
            int sum = 0;
            while(number != 0)
            {
                int last = number % 10;

                sum+=last;

                number /= 10;
            }
            return sum;
        }
    }

}
```







## Problem 4

### Check if a string is a Palindrome

#### Code:

```
namespace Camp3
{

    class Program
    {

        public static void Main(string[] args)
        {
            Console.WriteLine("Enter a string : ");
            string input = Console.ReadLine();

            string str = "";

            foreach (var c in input)
            {
                if(c != ' ')
                    str += c;
            }

            bool ispalin = true;
            for (int i = 0; i < str.Length / 2; i++)
            {
                if (str[i] != str[str.Length - 1 - i])
                {
                    ispalin = false;
                    break;
                }
            }
            if (ispalin)
            {
                Console.WriteLine("The string is a palindrome.");
            }
            else
            {
                Console.WriteLine("The string is not a palindrome.");
            }
        }
    }
}
```

```
}  
  
}  
  
}  
  
}
```



## Problem 5

### Count the number of vowels and consonants in a string

عدد الحروف الساكنة والمتحركة

#### Code :

```
namespace Camp3
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Enter a string : ");
            string input = Console.ReadLine().ToLower();

            int vowelCount = 0;
            int consonantCount = 0;

            foreach (char c in input)
            {
                if (char.IsLetter(c))
                {
                    switch (c)
                    {
                        case 'a':
                        case 'e':
                        case 'i':
                        case 'o':
                        case 'u':
                            vowelCount++;
                            break;
                        default:
                            consonantCount++;
                            break;
                    }
                }
            }
            // Output the counts
        }
    }
}
```

```
        Console.WriteLine($"Vowels: {vowelCount}, Consonants: {consonantCount}");
    }

}

}
```

## Data Structure

### Queue

#### Definition:

أي أن أول عنصر يدخل هو أول عنصر **FIFO (First In First Out)** هي هيكل بيانات خطي يعمل بنظام Queue الـ يخرج.

تخيل طابور الانتظار في المطعم: أول شخص يقف في الصف هو أول من يُخدم، بينما آخر شخص يقف هو آخر من يُخدم.

#### Queue كيف تعمل الـ

1. **(Rear)** تُضاف العناصر من النهاية الخلفية: **(Enqueue)** إضافة عنصر.
2. **(Front)** تُزال العناصر من المقدمة: **(Dequeue)** إزالة عنصر.
3. مشاهدة العنصر الأول دون حذفه: **(Peek/Front)** التحقق من العنصر الأمامي.
4. تحتوي على عناصر أم لا Queue معرفة هل الـ **(is Empty)** التحقق من كونها فارغة.

#### Queue متى نستخدم الـ

نُستخدم عندما نريد معالجة البيانات بنفس ترتيب وصولها، مثل

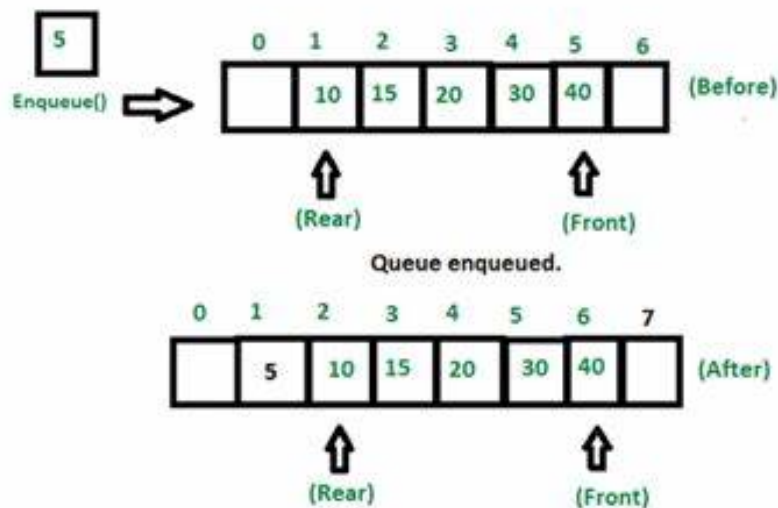
- إدارة المهام في أنظمة التشغيل (مهام الطباعة، جدولة العمليات).
- معالجة الطلبات في الخوادم (الطلبات تُعالج بالترتيب الذي وصلت به).
- نظام طابور الانتظار في التطبيقات (مثل طلبات توصيل الطعام).
- في خوارزميات البحث في الرسوم البيانية **Breadth-First Search (BFS)** نظام الـ.

## Queue مميزات الـ

- ✓ (FIFO) تحافظ على ترتيب العناصر
- ✓ سهولة الفهم والتطبيق
- ✓ (Real-time systems) مفيدة في أنظمة الوقت الفعلي

## Queue عيوب الـ

- ✗ لا تسمح بالوصول العشوائي للعناصر (مثل المصفوفة)
- ✗ قد تكون غير فعالة إذا احتجنا حذف/إضافة عناصر من المنتصف



Code:

```
using System;
using System.Collections.Generic;

public class Queue<T>
{
    private List<T> items;

    public Queue()
    {
        items = new List<T>();
    }
}
```

```

public bool IsEmpty()
{
    return items.Count == 0;
}

public void Enqueue(T item)
{
    items.Add(item);
}

public T Dequeue()
{
    if (!IsEmpty())
    {
        T item = items[0];
        items.RemoveAt(0);
        return item;
    }
    throw new InvalidOperationException("Queue is empty");
}

public T Peek()
{
    if (!IsEmpty())
    {
        return items[0];
    }
    throw new InvalidOperationException("Queue is empty");
}

public int Size()
{
    return items.Count;
}
}

```

## Circular Queue

يتم فيها استخدام المساحة بشكل أكثر (Queue) هي نوع من الطوابير (الطابور الدائري) **Circular Queue** الـ كفاءة عن طريق جعل الطابور "دائري". يعني إن أول العناصر وآخرها متصلين، فلو وصلت لنهاية الطابور، بترجع

للبداية لو فيه مكان فاضي.

### الفكرة باختصار:

- زي الطابور العادي، لكن لما يتمتلئ أو يتصقّر، بدل ما نضطر نعيد ترتيب العناصر أو نضيع مساحة، بنستخدم (المساحة الفاضية في البداية (لو أي عناصر اتشالت).
- بشكل دائري باستخدام (front gear) أو قائمة بحجم ثابت، وبتتعامل مع المؤشرات (array) بيستخدم مصفوفة (%). عملية modulo.

### الفرق عن الطابور العادي:

- في الطابور العادي، لما بتشيل عنصر من الأول، المساحة دي بتضيع وما بتستخدمش ثاني.
- في الطابور الدائري، المساحة بتُعاد استخدامها، فلو وصلت لنهاية المصفوفة وفي مكان فاضي في البداية، بيترجع يعبّي في البداية.

### مميزات:

- توفير المساحة.
- أداء أفضل لأنك ما بتحتاجش تعيد ترتيب العناصر.

### مثال بسيط:

لو عندك مصفوفة حجمها 5، وأضفت 3 عناصر، ثم أزلت عنصرين، في الطابور العادي المساحة دي بتضيع. لكن في الطابور الدائري، لو أضفت عنصر جديد، ممكن يتخط في المكان الفاضي في البداية.

### استخدامات:

- إدارة الذاكرة في أنظمة التشغيل.
- تطبيقات زي جدولة المهام أو إدارة البيانات في الشبكات.

!لو عايز كود أو شرح أعمق، قولني

### Code:

```
using System;

public class CustomQueue<T>
{
    private T[] elements;
    private int front;
    private int rear;
    private int capacity;
    private int count;
```

```

public CustomQueue(int size)
{
    elements = new T[size];
    front = 0;
    rear = -1;
    capacity = size;
    count = 0;
}

// Enqueue (Add an item to the rear)
public void Enqueue(T item)
{
    if (count == capacity)
        throw new InvalidOperationException("Queue is full!");

    rear = (rear + 1) % capacity;
    elements[rear] = item;
    count++;
}

// Dequeue (Remove and return the front item)
public T Dequeue()
{
    if (count == 0)
        throw new InvalidOperationException("Queue is empty!");

    T item = elements[front];
    front = (front + 1) % capacity;
    count--;
    return item;
}

// Peek (Get front item without removing)
public T Peek()
{
    if (count == 0)
        throw new InvalidOperationException("Queue is empty!");

    return elements[front];
}

// Check if queue is empty
public bool IsEmpty() ⇒ count == 0;

```



```

    // Get queue count
    public int Count => count;
}

class Program
{
    static void Main()
    {
        CustomQueue<int> queue = new CustomQueue<int>(5);

        queue.Enqueue(10);
        queue.Enqueue(20);
        queue.Enqueue(30);

        Console.WriteLine("Front element: " + queue.Peek()); // 10
        Console.WriteLine("Dequeued: " + queue.Dequeue()); // 10
        Console.WriteLine("Is empty? " + queue.IsEmpty()); // False
    }
}

```

## SQL Queries



## Query 1

### Problem Statement: 196. Delete Duplicate Emails

#### Description:

Write an SQL query to delete all duplicate email entries in a table named `Person`, keeping only unique emails based on their smallest `Id`.

#### Table Structure:

The table `Person` has the following columns:

- `Id`: The primary key column.
- `Email`: The email address.

#### Example:

##### Input:

```
+----+-----+
| Id | Email       |
+----+-----+
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
+----+-----+
```

##### Output:

After running your query, the `Person` table should have the following rows:

```
+----+-----+
| Id | Email       |
+----+-----+
| 1  | john@example.com |
| 2  | bob@example.com  |
+----+-----+
```

#### Note:

- Your output is the whole `Person` table after executing your SQL query.
- Use the `DELETE` statement to remove duplicate entries.

### SQL Query:

Here's the SQL query to achieve the desired result:

```
Delete P1  
from Person P1  
join Person P2  
on P1.Email = P2.Email  
where P1.Id > P2.Id
```



## Query 2

### Problem Statement: 197. Rising Temperature

#### Description:

Write an SQL query to find all dates' `Id` where the temperature is higher than the previous day (yesterday). Return the result table in any order.

عوزين نجيب Ids لما درجة الحرارة تكون اكبر من اليوم السابق لها

#### Table Structure:

The table `Weather` has the following columns:

- `Id` : The primary key column.
- `RecordDate` : The date of the record.
- `Temperature` : The temperature on that day.

#### Example:

##### Input:

```
+----+-----+-----+
| Id | RecordDate | Temperature |
+----+-----+-----+
| 1 | 2015-01-01 | 10          |
| 2 | 2015-01-02 | 25          |
| 3 | 215-01-03 | 20          |
| 4 | 2015-01-04 | 30          |
+----+-----+-----+
```

##### Output:

After running your query, the result should be:

```
+----+
| Id |
+----+
| 2 |
| 4 |
+----+
```

### Explanation:

- For `Id = 2`, the temperature ( `25` ) is higher than the previous day ( `10` ).
- For `Id = 4`, the temperature ( `30` ) is higher than the previous day ( `20` ).

### SQL Query:

Here's the SQL query to achieve the desired result:

```
Select W1.Id
from Weather W1
join Weather W2
ON DATEDIFF(w1.RecordDate, w2.RecordDate) = 1
where W1.Temperature > W2.Temperature
```



### Query 3

- **262. Trips and Users | Hard | LeetCode**

- **Table: Trips**

| +-----+-----+ |      |
|---------------|------|
| Column Name   | Type |
| +-----+-----+ |      |
| Id            | int  |
| Client_Id     | int  |
| Driver_Id     | int  |
| City_Id       | int  |
| Status        | enum |
| Request_at    | date |
| +-----+-----+ |      |

- (Id is the primary key for this table.)
- (Status is an ENUM type of ('completed', 'cancelled\_by\_driver', 'cancelled\_by\_client').)

- **Table: Users**

| +-----+-----+ |      |
|---------------|------|
| Column Name   | Type |
| +-----+-----+ |      |
| Users_Id      | int  |
| Banned        | enum |
| Role          | enum |
| +-----+-----+ |      |

- (The table holds all users. Each user has a unique Users\_Id, and Role is an ENUM type of ('client', 'driver', 'partner').)
- (Status is an ENUM type of ('Yes', 'No').)
- **Write a SQL to find the cancellation rate of requests with unbanned users (both client and driver must not be banned) each day between "2013-10-01" and "2013-10-03".**
- **The cancellation rate is computed by dividing the number of cancelled trips by the total number of trips with unbanned users (both client and driver must not be banned) in a specific date.**
  - Return the result table in the format of the second table below.

- The query result format is in the following example:

- **Table: Trips**

| Id | Client_Id | Driver_Id | City_Id | Status              | Request_at |
|----|-----------|-----------|---------|---------------------|------------|
| 1  | 1         | 10        | 1       | completed           | 2013-10-01 |
| 2  | 2         | 11        | 1       | cancelled_by_driver | 2013-10-01 |
| 3  | 3         | 12        | 6       | completed           | 2013-10-01 |
| 4  | 4         | 13        | 6       | cancelled_by_client | 2013-10-01 |
| 5  | 1         | 10        | 1       | completed           | 2013-10-02 |
| 6  | 2         | 11        | 6       | completed           | 2013-10-02 |
| 7  | 3         | 12        | 6       | cancelled_by_driver | 2013-10-02 |
| 8  | 2         | 12        | 12      | completed           | 2013-10-03 |
| 9  | 3         | 10        | 12      | completed           | 2013-10-03 |
| 10 | 4         | 13        | 12      | cancelled_by_driver | 2013-10-03 |

- **Table: Users**

| Users_Id | Banned | Role   |
|----------|--------|--------|
| 1        | No     | client |
| 2        | Yes    | client |
| 3        | No     | client |
| 4        | No     | client |
| 10       | No     | driver |
| 11       | No     | driver |
| 12       | No     | driver |
| 13       | No     | driver |

- **Result table:**

| Day        | Cancellation Rate |
|------------|-------------------|
| 2013-10-01 | 0.50              |
| 2013-10-02 | 0.50              |
| 2013-10-03 | 0.33              |

- **Explanation:**

- On 2013-10-01:
  - There were 4 requests, 2 of which were canceled.
  - However, user 2 was banned, so the request with (id = 2) was ignored.
  - The other request with (id = 4) was from a banned client (user 4), so it was also ignored.
  - So, there were 2 requests in total, and 1 of them was canceled, so the cancellation rate is  $(1 / 2) = 0.50$ .
- On 2013-10-02:
  - There were 2 requests, 1 of which was canceled.
  - However, user 2 was banned, so the request with (id = 6) was ignored.
  - So, there was 1 request in total, and it was completed, so the cancellation rate is  $(0 / 1) = 0.00$ .
- On 2013-10-03:
  - There were 2 requests, 1 of which was canceled.
  - The request with (id = 8) was from an unbanned client and driver, and it was completed.
  - The request with (id = 10) was from an unbanned client and driver, and it was canceled, so the cancellation rate is  $(1 / 2) = 0.50$ .

عندك نظام ادارة رحلات وبيحصل عمليات الغاء للرحلات فالمطلوب هو حساب معدل الغاء الرحلات التي تتم من خلال العميل وليست الملغاه من السائق وذلك من تاريخ لتاريخ اخر

الحل:

هنتحتاج نعمل دمج بين جدول الرحلات وجدول المستخدمين مرتين مرة للعميل ومرة للسائق بس بشرط .. ان ميكنش المستخدم محظور والشرط بتعنا من تاريخ كذا لتاريخ كذا

نلاحظ ان كل تاريخ هيجصل فيه حاجة وبالتالي نقسمهم مجموعت علي حسب التاريخ ونرتبهم برديو ونيجي نختار من كل دا التاريخ و حساب المعدل الي هيكون عبارة عن عدد الرحلات الملغاه علي عدد الرحلات الكلي

- **Solution:**

```
SELECT
Request_at AS Day,
ROUND(
SUM(CASE WHEN Status LIKE 'cancelled%' THEN 1 ELSE 0 END) /
```



```
        COUNT(*),  
        2  
    ) AS 'Cancellation Rate'  
FROM Trips t  
JOIN Users client ON t.Client_Id = client.Users_Id AND client.Banned = 'No'  
JOIN Users driver ON t.Driver_Id = driver.Users_Id AND driver.Banned = 'No'  
WHERE Request_at BETWEEN '2013-10-01' AND '2013-10-03'  
GROUP BY Request_at  
ORDER BY Request_at;
```



#### Query 4

- **511. Game Play Analysis I | Easy | LeetCode**

- **Table: Activity**

| +-----+-----+ |      |
|---------------|------|
| Column Name   | Type |
| +-----+-----+ |      |
| player_id     | int  |
| device_id     | int  |
| event_date    | date |
| games_played  | int  |
| +-----+-----+ |      |

- (This table shows the activity of players of some games.)
- (Each row is a record of a player that logged in and played a number of games.)

- **Write a SQL query that reports the first login date for each player.**

- The query result format is in the following example:

- **Activity table:**

| +-----+-----+-----+-----+ |           |            |              |
|---------------------------|-----------|------------|--------------|
| player_id                 | device_id | event_date | games_played |
| +-----+-----+-----+-----+ |           |            |              |
| 1                         | 2         | 2016-03-01 | 5            |
| 1                         | 2         | 2016-05-02 | 6            |
| 2                         | 3         | 2017-06-25 | 1            |
| 3                         | 1         | 2016-03-02 | 0            |
| 3                         | 4         | 2018-07-03 | 5            |
| +-----+-----+-----+-----+ |           |            |              |



- **Result table:**

| +-----+-----+ |             |
|---------------|-------------|
| player_id     | first_login |
| +-----+-----+ |             |
| 1             | 2016-03-01  |
| 2             | 2017-06-25  |
| 3             | 2016-03-02  |
| +-----+-----+ |             |

- **Solution:**

```
SELECT player_id, MIN(event_date) AS first_login  
FROM Activity  
GROUP BY player_id;
```

## My personal accounts links

|  |  |
|--|--|
|  LinkedIn | <a href="https://www.linkedin.com/in/ahmed-hany-899a9a321?utm_source=share&amp;utm_campaign=share_via&amp;utm_content=profile&amp;utm_medium=android_app">https://www.linkedin.com/in/ahmed-hany-899a9a321?<br/>utm_source=share&amp;utm_campaign=share_via&amp;utm_content=profile&amp;utm_medium=android_app</a> |
|  WhatsApp | <a href="https://wa.me/qr/7KNUQ7ZI3KO2N1">https://wa.me/qr/7KNUQ7ZI3KO2N1</a>  |
|  Facebook | <a href="https://www.facebook.com/share/1NFM1PfSjc/">https://www.facebook.com/share/1NFM1PfSjc/</a>  |