

Camp2



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

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


Algorithms & Data Structure & SQL Queries

Algorithms

Sorting Algorithms :

| the common types of sorting algorithms:

1. **Bubble Sort** ⇒ ✓
2. **Selection Sort** ⇒ ✓
3. **Insertion Sort** ⇒ ✓

4. Merge Sort ⇒ wait in camp3
5. Quick Sort ⇒ 
6. Heap Sort ⇒ wait in camp3
7. Counting Sort ⇒ wait in camp3
8. Radix Sort ⇒ wait in camp4
9. Bucket Sort ⇒ wait in camp4
10. Shell Sort ⇒ wait in camp4

Bubble Sort

تعريف Bubble Sort:

هو خوارزمية بسيطة تُستخدم لترتيب العناصر داخل قائمة (مثل ترتيب الأرقام من (ترتيب الفقاعات) Bubble Sort الأصغر إلى الأكبر). وهي تعمل من خلال مقارنة كل زوج من العناصر المتجاورة وتبديل أماكنها إذا كانت بالترتيب الخاطئ. تستمر هذه العملية حتى لا تبقى هناك أي تبديلات، ما يعني أن القائمة أصبحت مرتبة.

شرح بسيط:

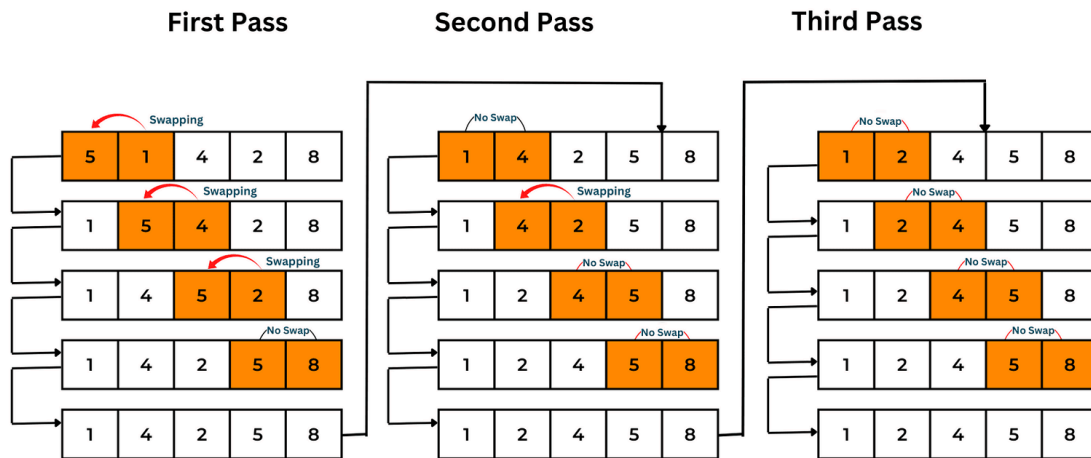
تخيل أن لديك مجموعة من الكرات بأحجام مختلفة مرصوفة على خط، وتريد ترتيبها من الأصغر إلى الأكبر. تبدأ من أول كرتين وتقارن بينهما:

- إذا كانت الكرة الأولى أكبر من الثانية، تقوم بتبديلهما.
- تنتقل للزوج التالي وتكرر نفس الشيء.

ومن هنا جاء) عند الانتهاء من المرور على كل الكرات، ستكون الكرة الأكبر قد "طففت" إلى النهاية، مثل الفقاعة اسم "Bubble").

تعيد هذه العملية عدة مرات، وفي كل مرة "تطفو" الكرة التالية الأكبر إلى مكانها الصحيح، حتى تصبح القائمة كلها مرتبة.

BUBBLE SORTING



Case	Time Complexity	Explanation
Best Case	$O(n)$	When the list is already sorted. Only one pass needed.
Average Case	$O(n^2)$	Random order: many comparisons and swaps are needed.
Worst Case	$O(n^2)$	When the list is in reverse order: most swaps needed.
Space Complexity	$O(1)$	In-place algorithm: no extra memory used (except temp).

Code:

```
namespace Camp2
{
    class Program
    {
        static void Main()
        {
            int[] array = { 5, 1, 4, 2, 8 };
            int length = array.Length;
            Console.WriteLine("Original array: " + string.Join(", ", array));
            BubbleSort(array, length);
            Console.WriteLine("Sorted array: " + string.Join(", ", array));
        }
    }
}
```

```

public static void BubbleSort(int[] array,int length)
{
    int i, j, temp;
    bool swapped;
    for(i=0;i<length-1;i++)
    {
        swapped = false;
        for (j=0;j<length-1-i;j++)
        {
            if (array[j] > array[j+1])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
                swapped = true;
            }
        }
    }

    if (!swapped)
    {
        break; // If no two elements were swapped, the array is sorted
    }
}
}
}
}
}

```

Selection Sort

هي خوارزمية ترتيب بسيطة تعمل عن طريق تقسيم القائمة إلى جزئين: جزء مرتب وجزء غير مرتب. في كل خطوة، تبحث عن **أصغر عنصر** في الجزء غير المرتب وتضعه في المكان الصحيح في الجزء المرتب.

شرح بسيط:

تخيل أن لديك مجموعة من البطاقات بأرقام مختلفة وتريد ترتيبها من الأصغر إلى الأكبر.

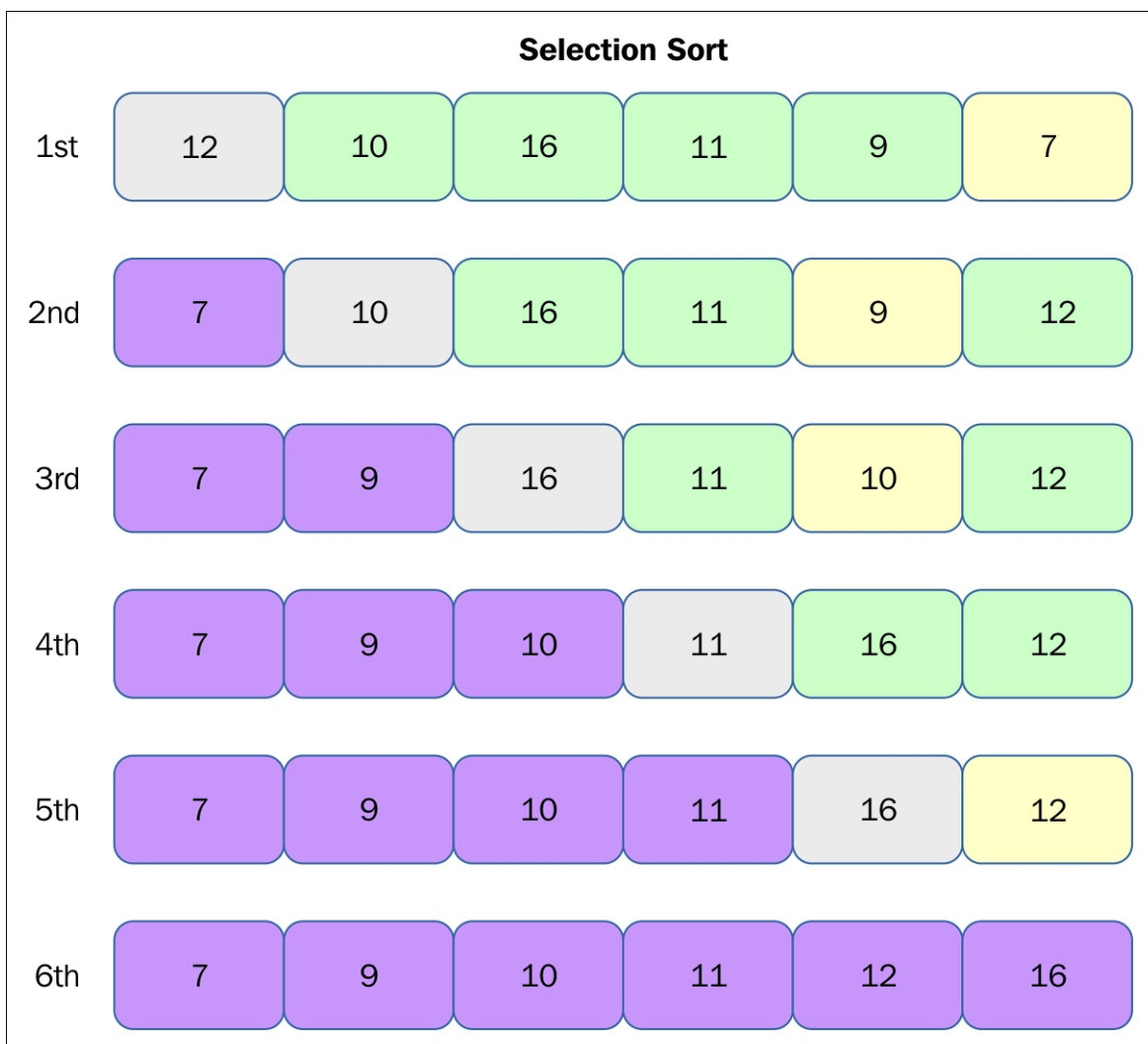
1. تبدأ بالبحث عن **أصغر رقم** في كل البطاقات.
2. بعد أن تجده، **تبذله** مع أول بطاقة في الصف.

3. ثم تبحث عن **أصغر رقم** في البطاقات المتبقية (ما عدا أول واحدة لأنها أصبحت في مكانها الصحيح).

4. تكرر العملية حتى ترتب كل البطاقات.

لأنها تقوم بـ"اختيار" العنصر المناسب في "**Selection**" كل مرة تختار أصغر عنصر وتضعه في مكانه، لذلك سُميت كل خطوة.

Case	Time Complexity	Explanation
Best Case	$O(n^2)$	Even if the list is sorted, it still checks every element to find the minimum.
Average Case	$O(n^2)$	Always makes the same number of comparisons, regardless of order.
Worst Case	$O(n^2)$	Same as above: always checks all remaining elements for the minimum.
Space Complexity	$O(1)$	No extra memory used — sorting is done in-place.



Code:

```
namespace Camp2
{

    using System;
    class Program
    {

        static void selectionSort(int[] arr)
        {
            int n = arr.Length;
            for (int i = 0; i < n - 1; i++)
            {

                // Assume the current position holds
                // the minimum element
                int min_idx = i;

                // Iterate through the unsorted portion
                // to find the actual minimum
                for (int j = i + 1; j < n; j++)
                {
                    if (arr[j] < arr[min_idx])
                    {

                        // Update min_idx if a smaller element
                        // is found
                        min_idx = j;
                    }
                }

                // Move minimum element to its
                // correct position
                int temp = arr[i];
                arr[i] = arr[min_idx];
                arr[min_idx] = temp;
            }
        }
    }
}
```

```

public static void Main()
{
    int[] arr = {12,10,16,11,9,7};
    Console.WriteLine("Before sorting:");
    Console.WriteLine(string.Join(" ", arr));
    selectionSort(arr);
    Console.WriteLine("After sorting:");
    Console.WriteLine(string.Join(" ", arr));
}
}
}

```

Insertion Sort

هي خوارزمية ترتيب بسيطة تعمل بطريقة تشبه ترتيب أوراق اللعب يدويًا: في كل مرة تأخذ ورقة (عنصر) وتضعها في المكان الصحيح بين الأوراق التي تم ترتيبها سابقًا.

شرح بسيط:

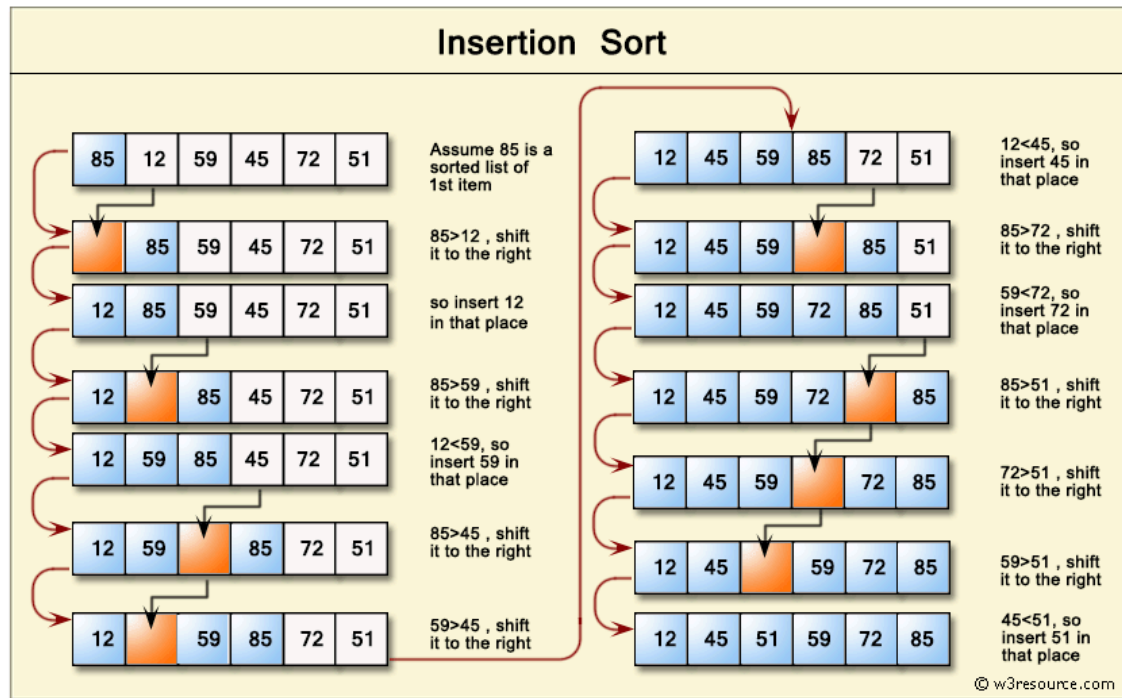
تخيل أنك تمسك أوراق لعب وتريد ترتيبها من الأصغر إلى الأكبر:

1. تبدأ بالورقة الأولى وتعتبرها مرتبة.
2. تأخذ الورقة التالية وتقارنها مع التي قبلها.
3. إذا كانت أصغر، تُحرك الورقة الأكبر إلى اليمين وتُدخل الورقة في مكانها الصحيح.
4. تكرر العملية لكل ورقة حتى يتم ترتيب المجموعة كلها.

في كل خطوة، يتم إدراج العنصر الحالي في المكان الصحيح بين العناصر التي تم ترتيبها بالفعل — ومن هنا جاء اسم **"Insertion" (إدراج)**.

Case	Time Complexity	Explanation
Best Case	$O(n)$	When the list is already sorted: only one comparison per element.
Average Case	$O(n^2)$	Elements are in random order: shifting and comparing needed.

Worst Case	$O(n^2)$	When the list is in reverse order: most shifting required for each element.
Space Complexity	$O(1)$	In-place algorithm: no extra memory used.



Code:

```
namespace Camp2
{
    using System;

    class Program
    {
        public static void InsertionSort(int[] arr)
        {
            int n = arr.Length;
            for(int i=1;i<n;i++)
            {
                int key = arr[i];
```



```

        int j = j = i - 1;
        while(j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

public static void Main()
{
    int[] arr = {85,12,59,45,72,51};
    Console.WriteLine("Unsorted array:");
    Console.WriteLine(string.Join(" ", arr));
    InsertionSort(arr);
    Console.WriteLine("Sorted array:");
    Console.WriteLine(string.Join(" ", arr));
}
}
}

```

Quick Sort

تقوم باختيار عنصر من **(Divide and Conquer)** هي خوارزمية ترتيب تعتمد على أسلوب "القسمه والفصل على يساره، والعناصر الأكبر pivot وتعيد ترتيب العناصر بحيث تكون العناصر الأصغر من الـ (pivot يسمى) القائمة على يمينه، ثم تُطبق نفس العملية على الجزئين بشكل متكرر.

✓ شرح بسيط بأسلوب الحياة اليومية:

تخيل أنك ترتب مجموعة من الأرقام

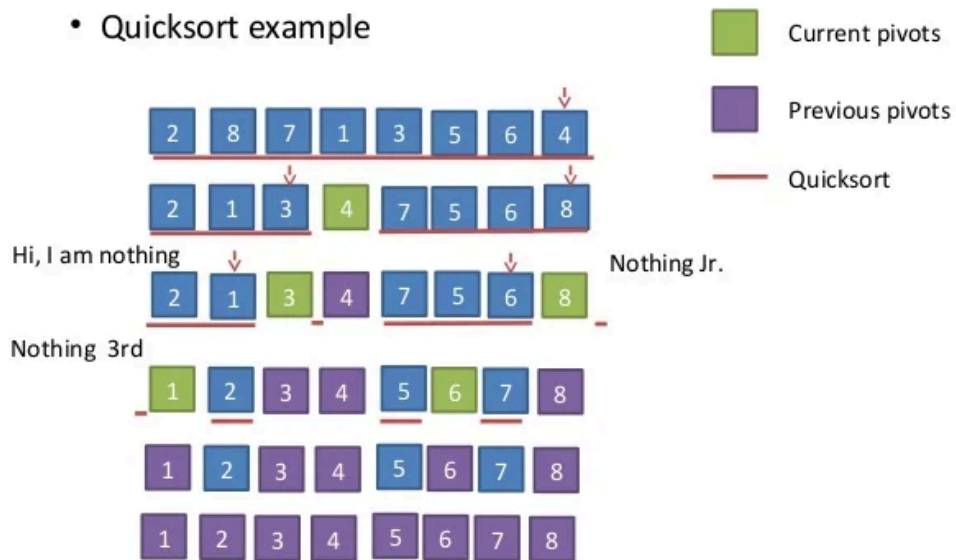
1. تختار رقمًا كـ "محور" (مثلًا الرقم في الوسط).
2. تقارن بقية الأرقام:

- الأرقام الأصغر تضعها على اليسار.
- الأرقام الأكبر تضعها على اليمين.

3. ثم تُعيد نفس الترتيب بشكل متكرر على كل مجموعة (يسار ويمين)، حتى تُرتَّب كل العناصر.

Quicksort Algorithm

• Quicksort example



Case	Time Complexity	Explanation
Best Case	$O(n \log n)$	When the pivot splits the list into two equal halves.
Average Case	$O(n \log n)$	Most cases fall here, even with random data.
Worst Case	$O(n^2)$	When pivot is always the smallest or largest (bad split each time).
Space Complexity	$O(\log n)$	Due to recursive calls (in-place sorting with stack usage only).

Code:

```
namespace Camp2
{
```

```
using System;
```

```
using System;
```

```
class Program
```

```
{
```

```
    // Partition function
```

```
    static int Partition(int[] arr,int low,int high)
```

```
    {
```

```
        int pivot = arr[high]; // pivot
```

```
        int i=low-1; // Index of smaller element
```

```
        for(int j=low;j<=high-1;j++)
```

```
        {
```

```
            if(arr[j] < pivot)
```

```
            {
```

```
                i++; // increment index of smaller element
```

```
                Swap(arr, i, j);
```

```
            }
```

```
        }
```

```
        Swap(arr, i + 1, high);
```

```
        return i + 1; // Return the partitioning index
```

```
    }
```

```
    // Swap function
```

```
    static void Swap(int[] arr,int i,int j)
```

```
    {
```

```
        int temp = arr[i];
```

```
        arr[i] = arr[j];
```

```
        arr[j] = temp;
```

```
    }
```

```
    // The QuickSort function implementation
```

```
    static void QuickSort(int[] arr,int low,int high)
```

```
    {
```

```
        if(low<high)
```

```
        {
```

```
            int pivot_index=Partition(arr,low,high);
```

```
            // Recursively sort elements before and after partition
```

```

        QuickSort(arr, low, pivot_index - 1);
        QuickSort(arr, pivot_index + 1, high);
    }
}

static void Main(string[] args)
{
    int[] arr = {2,8,7,1,3, 5 ,6,4};
    int n = arr.Length;
    Console.WriteLine("Unsorted array");
    Console.WriteLine(string.Join(" ", arr));
    QuickSort(arr, 0, n - 1);
    Console.WriteLine("Sorted array");
    Console.WriteLine(string.Join(" ", arr));
}
}

}

```

Some Problem Solving



1. write a program to find first & second smallest number

عاوزين نجيب اصغر رقمين في ال array

عشان نضمن ان الرقم اصغر لازم افرض الرقمين باكبر قيمة ممكنة

وابداً اقارنها مع عنصر عنصر بحيث نحصل علي اصغر عنصر ثم الاكبر منو

Code:

```
namespace Camp2
{

    using System;

    using System;
    using System.Collections.Generic;

    class Program
    {
        static void Main(string[] args)
        {
            int[] arr = { 5, 2, 8, 1, 4, 9, 3 };
            Console.WriteLine("Original array: " + string.Join(", ", arr));
            if (arr.Length < 2)
            {
                Console.WriteLine("Array must contain at least two elements.");
                return;
            }

            int min1 = int.MaxValue;
            int min2 = int.MaxValue;

            foreach(int num in arr)
            {
                if(num<min1)
                {
                    min2=min1;
                    min1 = num;
                }
                else if (num < min2 && num != min1)
                {
                    min2 = num;
                }
            }
        }
    }
}
```

```
    }  
  
    }  
  
    if (min2 == int.MaxValue)  
    {  
        Console.WriteLine("There is no second smallest element (all elements are equal o  
    }  
    else  
    {  
        Console.WriteLine("First smallest = " + min1);  
        Console.WriteLine("Second smallest = " + min2);  
    }  
    }  
    }  
  
}
```



2. write a program to find first & second Largest number

المرادي عوزين نجيب اكبر رقمين في الحاله دي هنعكس الي عملناه في المشكله الي فاتت

Code:

```
namespace Camp2
{

    using System;

    using System;
    using System.Collections.Generic;

    class Program
    {
        static void Main(string[] args)
        {
            int[] arr = { 5, 2, 8, 1, 4, 9, 3 };
            Console.WriteLine("Original array: " + string.Join(", ", arr));
            if (arr.Length < 2)
            {
                Console.WriteLine("Array must contain at least two elements.");
                return;
            }

            int max1 = int.MinValue;
            int max2 = int.MinValue;

            foreach(int num in arr)
            {
                if(num > max1)
                {
                    max2= max1;
                    max1 = num;
                }
                else if (num > max2 && num != max1)
                {
                    max2 = num;
                }
            }
        }
    }
}
```

```
        if (max2 == int.MinValue)
        {
            Console.WriteLine("There is no second Larger element (all elements are equal or c
        }
        else
        {
            Console.WriteLine("First larg = " + max1);
            Console.WriteLine("Second larg = " + max2);
        }
    }
}

}
```




3. Write a program to reverse sorting array

Bubble عوزين نرتب بس بالعكس بدل ما مرتبين تصاعدي لا نرتب تنازلي الموضوع بسيط مثلا لو طبقنا طريقه من طرق الترتيب الي خدناها مثلاها

هنعكس شرط واحد فقط

Code:

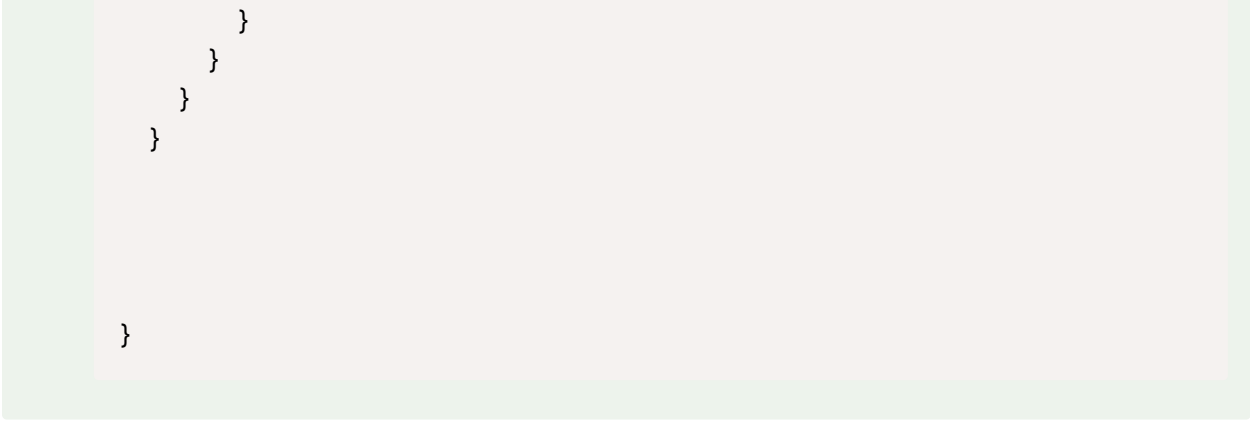
```
namespace Camp2
{

    using System;

    using System;
    using System.Collections.Generic;

    class Program
    {
        static void Main(string[] args)
        {
            int[] arr = { 5, 2, 8, 1, 4, 9, 3 };
            Console.WriteLine("Original array: \n" + string.Join(" ", arr));
            BubblesortDesc(arr);
            Console.WriteLine("Sorted array in descending order: \n" + string.Join(" ", arr));
        }

        public static void BubblesortDesc(int[] arr)
        {
            int n = arr.Length;
            for(int i=0;i<n-1; i++)
            {
                for(int j=0;j<n-1-i;j++)
                {
                    if(arr[j] < arr[j + 1])
                    {
                        // Swap arr[j] and arr[j + 1]
                        int temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                }
            }
        }
    }
}
```





4. write a program for Fibonacci series

هو عبارة عن مجموعة ارقام كل رقم يساوي مجموع الرقمين الي قبلو

0 1 1 2 3 5 8 13 21 34

Code:

```
namespace Camp2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter the number of terms: ");
            int n = int.Parse(Console.ReadLine());
            int a = 0, b = 1;
            Console.WriteLine("Fibonacci Series up to " + n + " terms:");

            for(int i=0;i<n;i++)
            {
                Console.Write(a+ " ");
                int next = a + b;
                a = b;
                b = next;
            }

        }
    }
}
```

Data Structure

Stack

Definition

هو طريقة لتخزين البيانات بتشتغل بنظام (يعني "مكدّس" أو كومة) **Stack** الـ "اللي يدخل آخر يطلع أول" - **Last In, First Out (LIFO)**.
يعني زي كأنك بتحط كتب فوق بعض، أول كتاب تحطه فوق هو أول واحد تشيله.

✓ العمليات الأساسية:

- **Push**: بتحط عنصر جديد فوق.
 - **Pop**: بتشيل العنصر اللي فوق.
 - **Peek**: تبص على اللي فوق من غير ما تشيله.
 - **IsEmpty**: تتأكد إذا كان فاضي ولا لا.
-

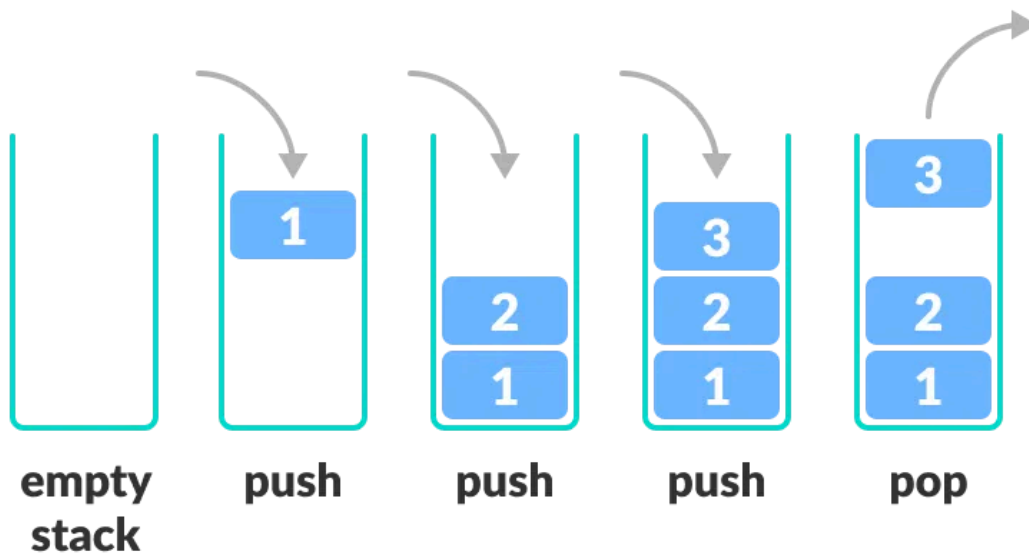
✓ مثال واقعي:

تخيل إنك بتحط أطباق فوق بعض:

- لما تيجي تشيل، بتبدأ من اللي فوق.
 - متقدرش توصل للطبق اللي تحت غير لما تشيل اللي فوقه.
-

✓ استخداماته:

- **Call Stack** (استدعاء الدوال في البرمجة).
 - في البرامج (Undo) "زر" تراجع.
 - التحقق من الأقواس في المعادلات.
 - **Backtracking** (خوارزميات زي الرجوع للخلف).
-



Code:

```
namespace Camp2
{
    // C# program to implement a stack using singly linked list
    using System;

    // Class representing a node in the linked list
    class Node
    {
        public int data;
        public Node next;
        public Node(int new_data)
        {
            this.data = new_data;
            this.next = null;
        }
    }

    // Class to implement stack using a singly linked list
    class Stack
    {
```

```

// head of the linked list
private Node head;

// Constructor to initialize the stack
public Stack() { this.head = null; }

// Function to check if the stack is empty
public bool isEmpty()
{
    // If head is null, the stack is empty
    return head == null;
}

// Function to push an element onto the stack
public void push(int new_data)
{
    // Create a new node with given data
    Node new_node = new Node(new_data);

    // Check if memory allocation for the new node
    // failed
    if (new_node == null)
    {
        Console.WriteLine("\nStack Overflow");
        return;
    }

    // Link the new node to the current top node
    new_node.next = head;

    // Update the top to the new node
    head = new_node;
}

// Function to remove the top element from the stack
public void pop()
{
    // Check for stack underflow
    if (this.isEmpty())

```

```

    {
        Console.WriteLine("\nStack Underflow");
    }
    else
    {

        // Update the top to the next node
        head = head.next;
        /* No need to manually free the memory of the
        * old head in C# */
    }
}

// Function to return the top element of the stack
public int peek()
{

    // If stack is not empty, return the top element
    if (!isEmpty())
        return head.data;
    else
    {
        Console.WriteLine("\nStack is empty");
        return int.MinValue;
    }
}
}

// Driver program to test the stack implementation
class Program
{
    static void Main(string[] args)
    {

        // Creating a stack
        Stack st = new Stack();

        // Push elements onto the stack
        st.push(11);
        st.push(22);
        st.push(33);
        st.push(44);
    }
}

```

```
// Print top element of the stack
Console.WriteLine("Top element is " + st.peak());

// removing two elements from the top
Console.WriteLine("Removing two elements...");
st.pop();
st.pop();

// Print top element of the stack
Console.WriteLine("Top element is " + st.peak());
    }
}
}
```

Infix to Postfix Expression

Infix to Postfix Conversion

Infix Expression: $(A/(B-C)*D+E)$

Symbol Scanned	Stack	Output
((-
A	(A
/	(/	A
((/(A
B	(/(AB
-	(/(-	AB
C	(/(-	ABC
)	(/	ABC-
*	(*	ABC-/
D	(*	ABC-/D
+	(+	ABC-/D*
E	(+	ABC-/D*E
)	Empty	ABC-/D*E+

Postfix Expression: $ABC-/D*E+$

Code :

```
namespace Camp2
{
    using System;
    using System.Collections.Generic;

    class Program
    {
        static int Prec(char c)
        {
            if (c == '^') return 3;
            else if (c == '/' || c == '*') return 2;
            else if (c == '+' || c == '-') return 1;
        }
    }
}
```

```

    else return -1;
}

static string InfixToPostfix(string s)
{
    Stack<char> st = new Stack<char>();
    string res = "";

    foreach (char c in s)
    {
        // If the scanned character is an operand,
        // add it to the output string.
        if (char.IsLetterOrDigit(c))
        {
            res += c;
        }

        // If the scanned character is an '(',
        // push it to the stack.
        else if (c == '(')
        {
            st.Push(c);
        }

        // If the scanned character is an ')',
        // pop and add to the output string from the stack
        // until an '(' is encountered.
        else if (c == ')')
        {
            while (st.Count > 0 && st.Peek() != '(')
            {
                res += st.Pop();
            }
            st.Pop();
        }

        // If an operator is scanned
        else
        {
            while (st.Count > 0 && Prec(c) <= Prec(st.Peek()))
            {
                res += st.Pop();
            }
        }
    }
}

```

```

        }
        st.Push(c);
    }
}

// Pop all the remaining elements from the stack
while (st.Count > 0)
{
    res += st.Pop();
}

return res;
}

static void Main()
{
    string exp = "a+b*(c^d-e)^(f+g*h)-i";
    Console.WriteLine(InfixToPostfix(exp));
}
}

```

SQL Queries



Query 1

Problem Description:

The `Employee` table holds all employees, including their managers. Each employee has an `Id`, and there is also a column for the `ManagerId`.

Table Structure:

```
+-----+-----+-----+-----+
| Id | Name | Salary | ManagerId |
+-----+-----+-----+-----+
| 1 | Joe | 70000 | 3 |
| 2 | Henry | 80000 | 4 |
| 3 | Sam | 60000 | NULL |
| 4 | Max | 90000 | NULL |
+-----+-----+-----+-----+
```

Task:

Write a SQL query to find employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

عوزين نجيب الموظفين الي بيقبضو اكثر من المديرين بتعهم

الحل:

هنعمل دمج بين الجدول ونفسو والشرط بتعنا ان مرتب الموظف اكثر من المدير

Expected Output:

```
+-----+
| Employee |
+-----+
| Joe |
+-----+
```

Code:

```
Select E.Name
from Employee E
join Employee M
```

```
on E.ManagerId = M.Id  
And E.Salary > M.Salary
```



Query 2

Problem Description:

Write a SQL query to find all duplicate emails in a table named `Person`.

عوزين نجيب الايميلات المكررة اكثر من مرة

الحل:

هناقسهم مجموعات علي حسب الايميلات وناخذ منهم الي متكرر اكثر من مرة

Table Structure:

```
+----+-----+
| Id | Email |
+----+-----+
| 1 | a@b.com |
| 2 | c@d.com |
| 3 | a@b.com |
+----+-----+
```

Task:

Identify and return all emails that appear more than once in the `Person` table.

Expected Output:

```
+-----+
| Email |
+-----+
| a@b.com |
+-----+
```

SQL Query:

```
SELECT Email
FROM Person
GROUP BY Email
HAVING COUNT(Email) > 1;
```




Query 3

عوزين نجيب العملاء الي مطلوبوش اي اوردرات هنعمل دمج عادي بين الجدولين ونزود شرط في حاله العميل معندوش طلبات

Problem Description:

Suppose that a website contains two tables: the `Customers` table and the `Orders` table. Write a SQL query to find all customers who never placed an order.

Table Structures:

1. Table: Customers

```
+----+-----+
| Id | Name |
+----+-----+
| 1 | Joe  |
| 2 | Henry|
| 3 | Sam  |
| 4 | Max  |
+----+-----+
```

2. Table: Orders

```
+----+-----+
| Id | CustomerId |
+----+-----+
| 1 | 3          |
| 2 | 1          |
+----+-----+
```

Task:

Identify and return all customers from the `Customers` table who do not have any corresponding entries in the `Orders` table (i.e., customers who never placed an order).

Expected Output:

```
+-----+
| Customers|
+-----+
| Henry    |
```


| Max |
+-----+

Solution

```
Select C.Name As Customers
from Customers C
left join Orders O
on O.CustomerId = C.Id
where O.CustomerId is Null
```

--or

```
Select C.Name As Customers
from Customers C
where not in (
select CustomerId
from Orders
)
```



Query4

Problem Description:

Write an SQL query to find employees who have the highest salary in each department.

عاوزين نجيب اكبر موظف بياخد مرتب في كل قسم

الحل:

هنعمل sub query تجبلنا اكبر مرتب في كل قسم ونخترهم

Table Structures:

1. Table: Employee

Id	Name	Salary	DepartmentId
1	Joe	70000	1
2	Henry	80000	2
3	Sam	60000	1
4	Max	90000	1
5	Janet	69000	2
6	Jim	90000	3

2. Table: Department

Id	Name
1	IT
2	Sales
3	Engineering

Task:

Identify and return all employees who have the highest salary in their respective departments.

Expected Output:

Department	Employee	Salary
------------	----------	--------

IT	Max	90000
Sales	Henry	80000
Engineering	Jim	90000

Solution

```

Select D.Name as Department, E.Name as Employee, E.Salary as Salary
from Employee E
join Department D
on E.DepartmentId =D.Id
where (DepartmentId,Salary) in
(
select DepartmentId,Max(Salary)
from Employee
group by DepartmentId
)

```



Problem Description:

Write an SQL query to find employees who earn the top three salaries in each department.

عوزين نجيب اكبر 3 موظفين بياخدو مرتبات في كل قسم

الحل:

نبدأ نقسم لاقسام وكل قسم نرتب فيه علي حسب المرتبات تنازلي

وكل واحد هياخذ رتبة نبدأ بقا ناخذ اول 3 رتب منهم

Table Structures:

1. Table: Employee

Id	Name	Salary	DepartmentId
1	Joe	70000	1
2	Henry	80000	2
3	Sam	60000	1
4	Max	90000	1
5	Janet	69000	2
6	Jim	90000	3
7	Will	80000	1
8	Randy	85000	1
9	Tom	76000	2

2. Table: Department

```
+-----+
| Id | Name  |
+-----+
| 1  | IT    |
| 2  | Sales |
| 3  | Engineering|
+-----+
```

Task:

Identify and return all employees who have the top three salaries in their respective departments.

Expected Output:




```
+-----+-----+-----+
| Department | Employee | Salary |
+-----+-----+-----+
| IT         | Max     | 90000  |
| IT         | Randy   | 85000  |
| IT         | Will    | 80000  |
| Sales      | Henry   | 80000  |
| Sales      | Tom     | 76000  |
| Sales      | Janet   | 69000  |
| Engineering| Jim     | 90000  |
+-----+-----+-----+
```

Solution

```
with RankedEmployees as
(
  SELECT
    e.Id,
    e.Name,
    e.Salary,
    d.Name AS Department,
    DENSE_RANK() OVER (PARTITION BY e.DepartmentId ORDER BY
      e.Salary DESC) AS SalaryRank
  FROM
    Employee e
  Join
    Department d ON e.DepartmentId = d.Id
)

select Department , Name as Employee , Salary
from RankedEmployees
SalaryRank <= 3
```

 **My personal accounts links**

 LinkedIn	https://www.linkedin.com/in/ahmed-hany-899a9a321? utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=android_app
 WhatsApp	https://wa.me/qr/7KNUQ7ZI3KO2N1
 Facebook	https://www.facebook.com/share/1NFM1PfSjc/