**סיבוכיות:**

**דוגמא להסבר סיבוכיות:**

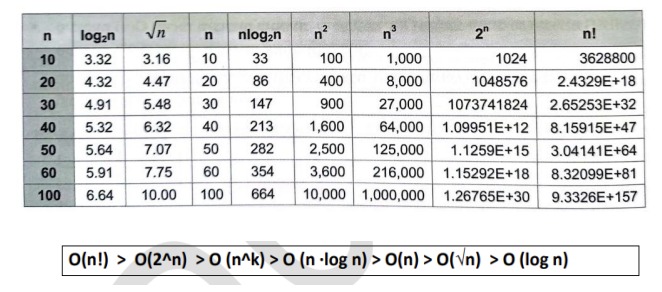
Node Length:

בהנחה ש[אורך הקלט] n מייצג את גודל הרשימה, סיבוכיות הפעולה היא O(n) [לינארית] כיוון שהאלגוריתם מבצע מעבר על כל אחד [n] איברי הרשימה (כיוון שהלולאה מתבצעת n פעמים)

וגודל הצעד הבסיסי הוא קבוע O(1)[קבוע], כך ש- n \* O(1) = O(n)

(כמות העבודה שכל איבר היה מבצע, הוא קבוע)

**איזו סיבוכיות יותר גדולה:**



**הדרך המילולית להגדרת סיבוכיות:**

O(1) - קבוע

O(logn) - לוגריתמית

O(n) - לינארית

O(n log n) - לינארי לוגריתמי

O(n^2) – ריבועית

O(n^3) - זמן קובי

O(2^n) – אקספוננציאלי

O(n!) - פקטוריאלי

**דוגמאות לרקורסיה:**

//Entry claim: The function input an integer

//Exit claim: The function print from 1 - n(the integer inputed)

static void PrintUp(int n)

{

if(n != 0)

{

PrintUp(n - 1);

Console.Write($"{n} ");

}

}

// Entry: Takes an integer `n` as input.

// Exit: Calculates the factorial of `n` using a recursive approach and returns the result.

static int Factorial(int n)

{

if (n == 0)

{

return 1;

}

else

{

return n \* Factorial(n - 1);

}

}

// Entry: Takes an integer `n` as input.

// Exit: Prints numbers from `n` down to 1 and then back up to `n` in the pattern `n, n-1, ..., 1, 1, 2, ..., n`.

static void PrintFromNumToNum(int n)

{

if (n != 1)

{

Console.Write($"{n} ");

PrintFromNumToNum(n - 1);

Console.Write($"{n} ");

}

else

{

Console.Write($"{n} ");

}

}

// Entry: Takes an integer `n` as input.

// Exit: Calculates the sum of numbers from `1` to `n` using a recursive approach and returns the result.

static int calcSumToNumber(int n)

{

if (n == 1)

{

return 1;

}

else

{

return n + calcSumToNumber(n - 1);

}

}

**רקורסיה של מערכים:**

//Entry: The function inputs and array

//Exit: the functions returns True if the array is in an ascending order,

False if not

static bool AscendingOrder(int[] arr)

{

if (i == arr.Length-1)

return true;

else

if (arr[i] >= arr[i+1])

return false;

return AscendingOrder(arr, i + 1);

}

//Entry: the function input an array of integers

//Exit: the function returns True if there is a negative number in the array, False if not

static bool IsThereNegative(int[] arr, int i)

{

if (i == arr.Length - 1)

{

if (arr[i] < 0)

return true;

return false;

}

if (arr[i] < 0)

return true;

return IsThereNegative(arr, i + 1);

}

//Entry: The function inputs an array

//Exit: The function returns the biggest number in the array

static int MaxNumber(int[] arr)

{

if (i == arr.Length - 1)

return arr[i];

return Math.Max(MaxNumber(arr, i), arr[i]);

}

//Entry: The function inputs an array of integers

//Exit the function returns the sum of all the positive numbers in the array

static int SumOfAllPositive(int[] arr, int i)

{

if (i == arr.Length - 1)

{

if (arr[i] > 0)

return arr[i];

return 0;

}

if (arr[i] > 0)

return SumOfAllPositive(arr, i + 1) + arr[i];

return SumOfAllPositive(arr, i + 1);

}

**רשימה מקושרת:**

public class Node<T>

{

private T value;//The value

private Node<T> next;//points onto the next node

/// <summary>

/// The Function sets the this.value To the type and its value inputed and this.next into the T Node inputed

/// </summary>

/// <param name="value">The value this.value is set to</param>

/// <param name="next">The T Node this.next becomes the pointer of</param>

public Node(T value, Node<T> next)

{

this.value = value;

this.next = next;

}

/// <summary>

/// Build Function that inputs only a T and sets its value as this.value and this.next as null

/// </summary>

/// <param name="value">The value inputed</param>

public Node(T value)

{

this.value = value;

this.next = null;

}

public T GetValue()

{

return this.value;

}

public void SetValue(T value)

{

this.value = value;

}

public bool HasNext()

{

return this.next != null;

}

public Node<T> GetNext()

{

return this.next;

}

public void SetNext(Node<T> next)

{

this.next = next;

}

public override string ToString()

{

return $"{this.value} -> {this.next}";

}

}

**פעולות חשובות לרשימה מקושרת:**

**Int:**

//Entry: The function input an integer 'n'

//Exit: The function creates and returns an int Node in length of 'n', in the order of input

static Node<int> BuildList(int n)

{

Console.Write($"Enter an integer: ");

Node<int> list = new Node<int>(int.Parse(Console.ReadLine()));

Node<int> tempList = list;

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

{

Console.Write($"Enter an integer: ");

list.SetNext(new Node<int>(int.Parse(Console.ReadLine())));

list = list.GetNext();

}

return tempList;

}

//Entry: The function inputs an integer 'n'

//Exit: The function creates and returns an int Node in length of 'n', in the opposite order of input

static Node<int> BuildReverseList(int n)

{

Console.Write($"Enter an integer: ");

Node<int> list = new Node<int>(int.Parse(Console.ReadLine()));

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

{

Console.Write($"Enter an integer: ");

list = new Node<int>(int.Parse(Console.ReadLine()), list);

}

return list;

}

static Random rnd = new Random();

//Entry: the functions inputs three integers, 'length' the length of the list,'from' the smallest number in the range, 'until' the biggest number in the range

//Exit: the function returns a int Node with randomized values in the renge inputed

static Node<int> CreateRandomIntNode(int length, int from, int until)

{

Node<int> list = new Node<int>(rnd.Next(from, until + 1));

Node<int> rtnlist = list;

int cnt = 1;

while (cnt <= length)

{

list.SetNext(new Node<int>(rnd.Next(from, until + 1)));

list = list.GetNext();

cnt++;

}

return rtnlist;

}

//Entry: The function inputs an integer 'n'

//Exit: The function creats and returs an int Node sorted list

static Node<int> BuildSortedList(int n)

{

Console.Write($"Enter an integer: ");

Node<int> list = new Node<int>(int.Parse(Console.ReadLine()));

Node<int> startOfList = list;

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

{

Console.Write($"Enter an integer: ");

int num = int.Parse(Console.ReadLine());

if (num <= startOfList.GetValue())

{

startOfList = new Node<int>(num, startOfList);

}

else

{

list = startOfList;

while (list.HasNext() && num > list.GetNext().GetValue())

{

list = list.GetNext();

}

list.SetNext(new Node<int>(num, list.GetNext()));

}

}

return startOfList;

}

//Entry: the function inputs a int node and an integer 'num'

//Exit: the fucntion returns True if 'num' exist in the linked list,

False if not

static bool Exist(Node<int> list, int num)

{

while (list != null)

{

if (num == list.GetValue())

return true;

list = list.GetNext();

}

return false;

}

//Entry: The function inputs an the first int Node in the linked list

//Exit: The function returns the length of the linked list

static int GetLength(Node<int> list)

{

int count = 0;

while (list != null)

{

count++;

list = list.GetNext();

}

return count;

}

//Entry: The function inputs a pointer to the first node in the linked list and a Node

//Exit: The function returns a pointer to the node inputed in the linked list, null if it doesn't exist

static Node<int> GetPosOfNode(Node<int> list, Node<int> node)

{

while (list.HasNext())

{

if (list == node)

return list;

list = list.GetNext();

}

return null;

}

//Entry: The function inputs a pointer to the first node in the linked list, and an integer 'num'

//Exit: The function returns a pointer to a node with the value of 'num', null if doesn't exist

static Node<int> GetPosOfNum(Node<int> list, int num)

{

while (list.HasNext())

{

if (list.GetValue() == num)

return list;

list = list.GetNext();

}

return null;

}

**String:**

//Entry: The function inputs a pointer to the first string Node

//Exis: The function return the shortest name/string in linked list

static string ShortestName(Node<string> list)

{

if (!list.HasNext())

{

return list.GetValue();

}

else

{

int lenghtNow = list.GetValue().Length;

string nameNext = ShortestName(list.GetNext());

int lengthNext = nameNext.Length;

if (lenghtNow > lengthNext)

return nameNext;

return list.GetValue();

}

}

//Entry: The function inputs a pointer to the first node in a string linked list

//Exit: The function returns True if the string are in an asceing order, False if not

static bool AscendingOrderByABC(Node<string> list)

{

while (list.HasNext())

{

if (list.GetValue().CompareTo(list.GetNext().GetValue()) > 0)

return false;

list = list.GetNext();

}

return true;

}

//Entry: The function inputs a pointer to the first node in a string linked list

//Exit: The function prints all of the letters that don't appear in any of the string in the linked list

static void AllLetterThatListDontAppear(Node<string> list)

{

for (char i = 'a'; i <= 'z'; i++)

{

bool contains = false;

Node<string> temp = list;

while (temp.HasNext() && !contains)

{

if (temp.GetValue().Contains(i.ToString()))

contains = true;

temp = temp.GetNext();

}

if (!contains)

Console.Write($"{i} ");

}

Console.WriteLine();

}

**פעולות למערך חד-ממדי:**

// Entry: The function inputs an integer array 'arr' and an integer 'num' to be found in the array.

// Exit: The function returns true if the 'num' is found in the sorted array 'arr' using binary search, otherwise, false.

static bool binarySearch(int[] arr, int num)

{

int left = 0;

int right = arr.Length - 1;

while (left <= right)

{

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

if (arr[mid] == num)

return true;

if (arr[mid] < num)

left = mid + 1;

else

right = mid - 1;

}

return false;

}

// Entry: The function inputs an integer array 'arr' to be sorted in ascending order.

// Exit: The function modifies the input array 'arr' such that its elements are sorted in ascending order using the bubble sort algorithm.

static void BubbleSort(int[] arr)

{

int tempnum;

for (int i = 0; i < arr.Length; i++)

{

for (int j = i; j < arr.Length - 1; j++)

{

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

{

tempnum = arr[j];

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

arr[j + 1] = tempnum;

}

}

}

}

// Entry: The function prompts the user to input the length and elements of an integer array.

// Exit: The function returns an integer array filled with user-inputted elements.

static int[] CreatintArray()

{

int length;

Console.Write("Enter the length of the Array to be created:");

length = int.Parse(Console.ReadLine());

int[] arr = new int[length];

Console.WriteLine($"Enter {length} things you want in the Array");

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

{

arr[i] = int.Parse(Console.ReadLine());

}

return arr;

}

// Entry: The function generates a random integer array of specified length within a specified range.

// Exit: The function returns a random integer array within the specified range and length.

static Random rnd = new Random();

static int[] CreateRndArray(int length, int from, int until)

{

int[] arr = new int[length];

for (int i = 0; i < arr.Length; i++)

{

arr[i] = rnd.Next(from, until + 1);

}

return arr;

}

// Entry: The function checks if a given integer 'num' exists in the provided integer array 'arr'.

// Exit: The function returns true if 'num' exists in the array 'arr', otherwise, false.

static bool Exist(int num, int[] arr)

{

for (int i = 0; i < arr.Length; i++)

{

if (arr[i] == num)

{

return true;

}

}

return false;

}

// Entry: The function inserts an integer 'num' into an integer array 'arr' while maintaining its sorted order up to 'amountOfInputed' elements.

// Exit: The function modifies the array 'arr' by inserting 'num' at the appropriate position to maintain sorted order.

static void InsertionSort(int num, int amountOfInputed, int[] arr)

{

bool found = false;

if (amountOfInputed == 0)

{

arr[0] = num;

}

else if (amountOfInputed < arr.Length)

{

for (int i = 0; i < amountOfInputed && !found; i++)

{

if (num < arr[i])

{

for (int j = amountOfInputed; j > i; j--)

{

arr[j] = arr[j - 1];

}

arr[i] = num;

found = true;

}

}

if (!found)

arr[amountOfInputed] = num;

}

}

// Entry: The function prints the elements of the provided integer array 'arr' on the console.

// Exit: The function prints the elements of the input array 'arr'.

static void PrintArray(int[] arr)

{

for (int i = 0; i < arr.Length; i++)

{

Console.Write($"|{arr[i]}| ");

}

Console.WriteLine("");

}

**פעולות למערך דו-ממדי:**

//Entry: The function inputs an int matrix

//Exit: The functions prints the matrix

static void PrintMatrix(int[,] mat)

{

for (int i = 0; i < mat.GetLength(0); i++)

{

for (int j = 0; j < mat.GetLength(1); j++)

{

Console.Write($"|{mat[i, j]}|");

}

Console.WriteLine();

}

}

//Entry: The fucntion inputs for integers, 'rows' amount of rows, 'cols' amoutn of collumns, 'from' the smallest number in the renge, 'to' the biggest number in the renge

//Exit: The function returns a randomized int matrix with the values inputed

static Random rnd = new Random();

static int[,] CreateRandomIntMatrix(int rows, int cols, int from, int to)

{

int[,] mat = new int[rows, cols];

for (int i = 0; i < mat.GetLength(0); i++)

{

for (int j = 0; j < mat.GetLength(1); j++)

{

mat[i, j] = rnd.Next(from, to + 1);

}

}

return mat;

}

**פעולות שאולי יהיה צריך:**

//Entry: The function inputs an integer 'n'

//Exit: The function returns true if the numbers is a Prime number, False if not

static bool IsPrime(int n)

{

if (n <= 1)

return false;

for (int i = 2; i <= Math.Sqrt(n); i++)

{

if (n % i == 0)

{

return false;

}

}

return true;

}