Reference:

<https://www.youtube.com/watch?v=DLo8dLBtI-0&list=PL88kafUXXgBaAgb0h3-ZMvzxb5J2qFrut&index=3>

Stack (LIFO)

* هو عبارة عن List بحذف وبضيف **من أخره فقط**

Queue (FIFO)

* هو عبارة عن List بضيف **ع الأخير وبحذف من الأول**

**مشان ماخربط بلش من اليسار الى اليمين**

Append

Pop(0)

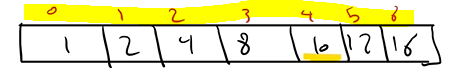
Searching Algorithm

1. Binary Search:

* بتختصر كل مرة نصف المسافة
* حسناها من ال Linear اللي تعقيدها O(n) الى O(Log n) للنصف
* يجب ان تكون العناصر **مرتبة**
* رح يكون عندي ثلاث مؤشرات Low, Height, Mid :

1. low = low Index
2. Height = Height Index
3. Mid = (low + height) / 2

مثال: ابحث عن الرقم 10 ضمن المصفوقة **المرتبة تصاعديا:**



static int binarySearch(int[] arr, int numberToFind)

{

int low = 0,

height = arr.Length - 1,

mid = (low + height) / 2;

while (low <= height)

{

mid = (low + height) / 2;

if (arr[mid] == numberToFind)

return mid;

else if (numberToFind > arr[mid])

low = mid + 1;

else

height = mid - 1;

}

return -1;

}

Sorting Algorithm

1. Selection Sort

* كل مرة نبحث عن العنصر الأصغر ونضعه بأول خانة
* Time Complex = o(n2)
* Space Complex = o (1)
* Check GitHub for Code.

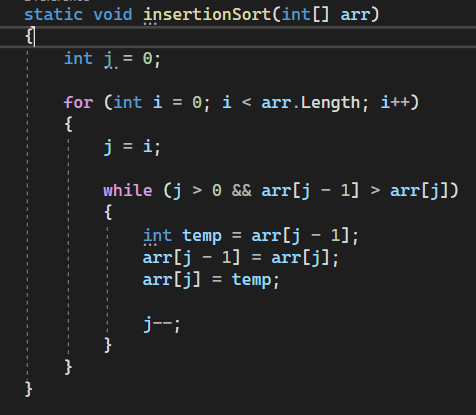
1. Bubble Sort

* باخد أول رقمين
* بقارهن مع بعض
* بحط الرقم الأكبر على اليمين
* عند كل لفة بصير الرقم الاكبر بآخر المصفوفة
* بكرر هي العملية حتى تترتب معي المصفوفة
* Time Complex = o(n2)
* Space Complex = o(1)
* Check Code on Github
* **على الرغم من أن ال Selection Sort and Bubble Sort الهن نفس تعقيد الوقت الا أنه ال Bubble Sort أسرع (لأنها أذكى) بسبب أنه في حال صار عندي لفة بدون عملية تبديل -< هاد الشي بدلني على انه المصفوفة صارت مرتبة صح فبالتالي بعمل Break ولكن في حالة ال Selection Sort رح يمر على كل العناصر حتى لو أجت المصفوفة مرتبة بالأساس**

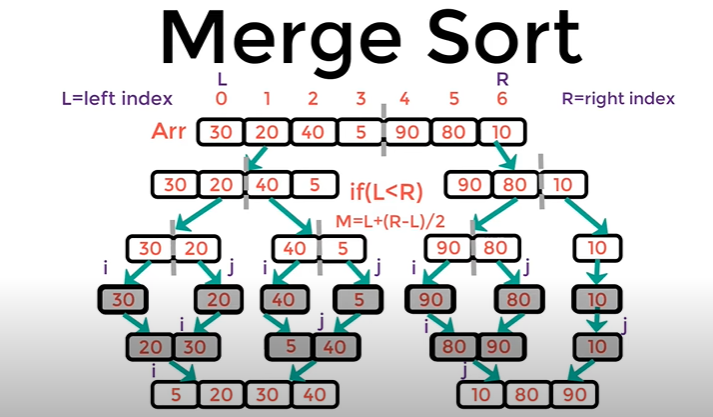
1. Insertion Sort

* ببلش من تاني عنصر وبحطه بمكانه الصحيح على يسار العنصر الأول وهكذا لباقي العناصر
* Time Comp = O(n2)

Space Comp = O(1)

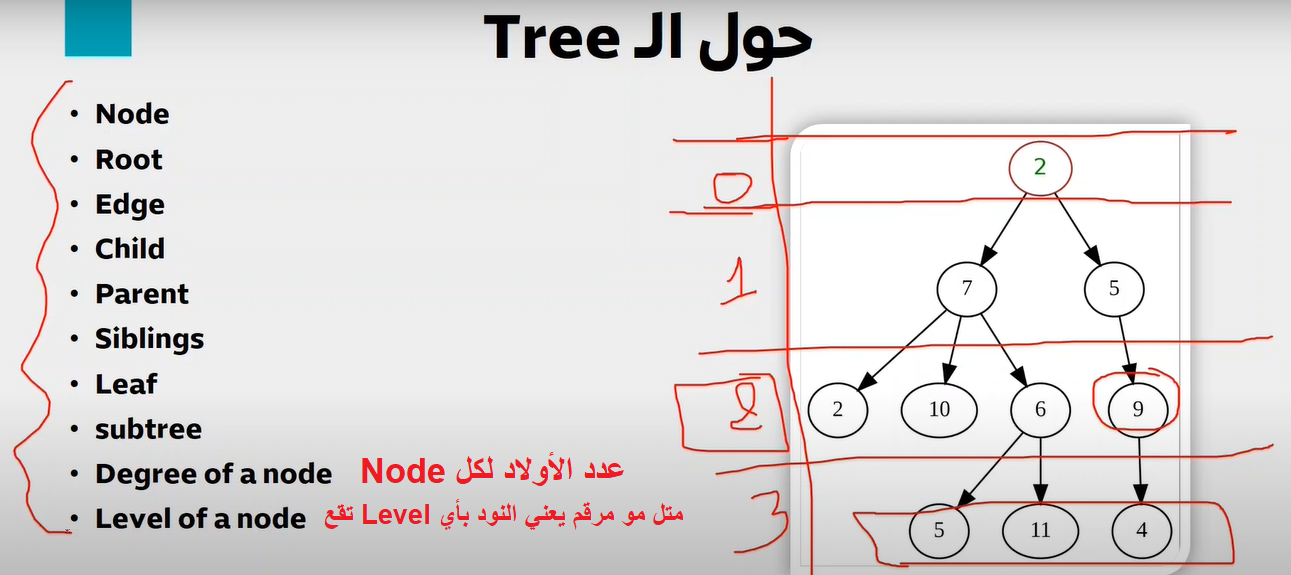


1. Merge Sort



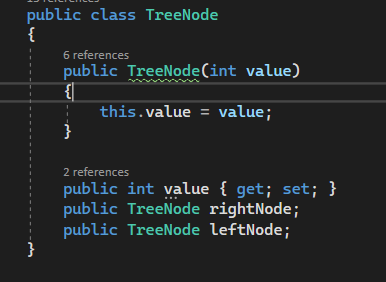
Tree Data Structure

مفاهيم عامة حول ال Tree



Binary Tree:

1. لديها ابنين فقط
2. سهل جدا برمجتها



Binary Search Tree (BST):

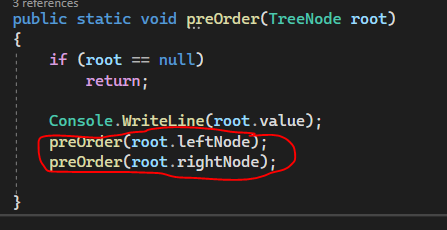
* هي حالة خاصة من ال Binary Tree بحيث يكون **الولد اليساري أصغر من الأب والولد اليميني أكبر من الأب (وأصغر من ال Root)**
* السؤال هنا ماهو الفائدة منها؟ من اسمها هو تسريع عملية البحث
* Time Compx = O(log(n)) لأنه كل مرة منكب نص العناصر بالبحث

Traverse Binary Tree

* يوجد عدة طرق للمرور على كافة عناصر ال Tree :

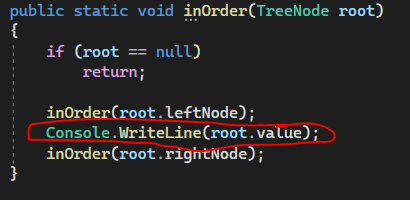
1. Pre-Order:

* ببلش من ال root وبطبع الطرف اليساري كاملا (دائما باخد ال Node وبطبع اليسار للاخير بعدين بصي بطبع الطرف اليميني)
* برمجته جدا سهلة بستدعيه لليسار بعدين لليمين متل موشايف بالكود:



1. In-Order:

* يسار - Root- يمين
* سترجع لك القيم **مرتبة تصاعديا**
* سهل جدا بالبرمجة (**نفس السابق بس بحط الطباعة بالنص** متل مو مبين بالكود)

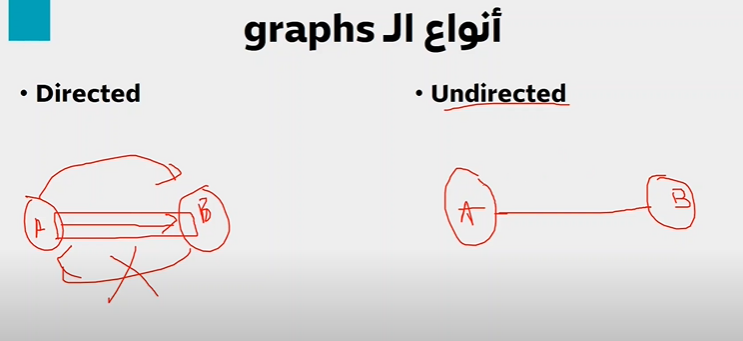


Graph

يقسم ال graph حسب عدة عوامل منها :

1. Directed/UnDirected :

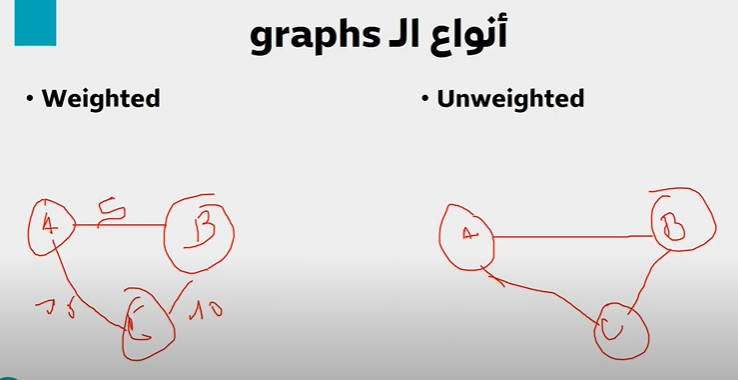
متل مومبين بالشكل أنه ال Directed اله سهم من A->B



1. Weighted/Un Weighted :

Un Weighted : يعني فقط يوجد علاقة بين نود وأخرى

Weighted :يعني الطريق من نود الى نود يكون اله وزن أي قيمة



* **يمكن مزج بين التصنيف الأول والتصنيف الثاني للحصول على أنواع عديدة من ال graph**

**(مثل graph Un weighted-Directed وهكذا..)**

* حالات خاصة لل graph:

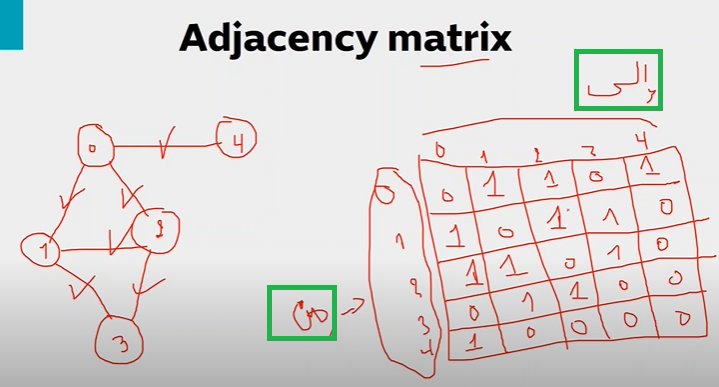
1. Tree (علاقة أب و ابن) ولايوجد حلقات مثل ال graph (يعني بنطلق من عنصر وبلف على كذا عنصر بعدين برجع للعنصر نفسه)
2. Disconected graph : يعني ممكن ال graph نفسه يحتوي على مكونين منفصلين



كيف يمكن أن نبرمج ال graph ؟

1. Adjacency Matrix Representation:

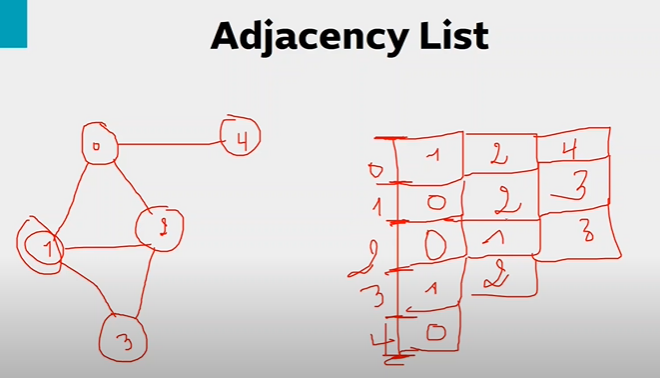
* ببساطة برسم جدول الأسطر تمثل (من) والأعمدة تمثل (إلى) وبربط بينهم عن طريق وضع 1 وبس خلصهن بحط الباقي 0
* تعتمد هذه الطريقة على ال Matrix أو 2 Dimentional Array
* Space Comp : الأمر واضح عندي n سطر و n عمود فبالتالي **O(n2)**



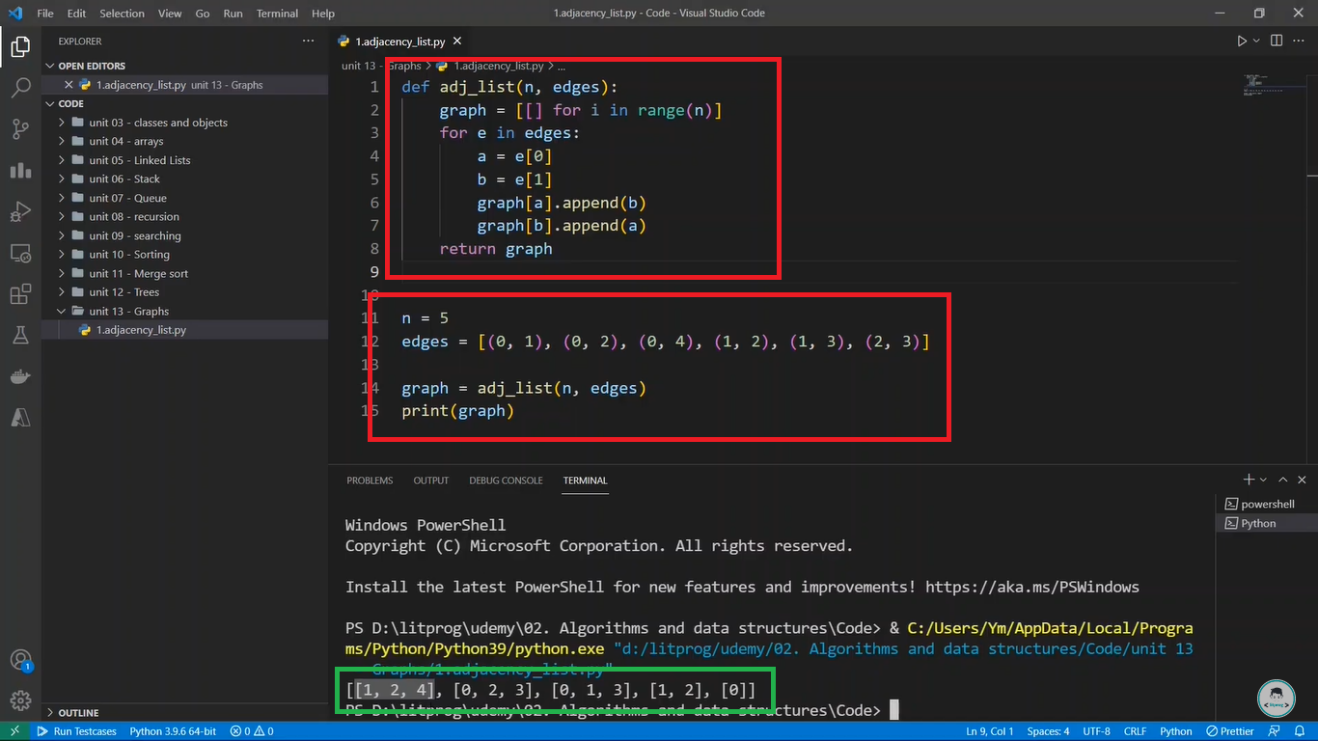
1. Adjacency List Representation:

* بحط كل node بسطر وبحط جنبها العناصر للي بتصلها فقط (يعني هي النود هي عبارة عن عنوان بيربطك مع list اللي بتوصل هي النود مع جيرانها)
* List Of Lists
* SpaceCompx :

كل Edge بسجله مرتين (ذهاب واياب) وعليه 2\*m حيث m هي عدد ال edges بالإضافة الى عد عناصر النودات (n) فبالتالي التعقيد هو : 2\*m+n بحف الثوابت بصير O(m+n)



* بالصورة تحت تمثلك الكود الخاص بال Adjacency List





1. الطريقة الأولى:

* مساحة أكبر
* أسرع بالإستجاع:

في حال بدي أعرف أنه في طريق من 0 الى 4 ببساطة بحط ال arr[0,4] في حال كانت 1 يعني موجود علاقة واذا كانت صفر يعني مافي علاقة

1. الطرقة الثانية :

* مساحة أقل
* أبطا لآن بدي مر على جميع جيران العنصر