**Report.**

At first, we import *random* to get access to the **random** module in order to generate a random number in a given range or to pick a random element from a list. We then import the ***time*** module which provides various time-related functions. Next we import ***heapq.*** Heap data structure is mainly used to represent a priority queue. In python, it is available as **heapq** module which implements a Min-heap.

**import** random, time, heapq

We then import ***matplotlib*** which is a plotting library for the **Python** programming language and its numerical mathematics extension NumPy. Each **pyplot** function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the **plot** with labels, etc.

**import** matplotlib.pyplot **as** plt

**Bubblesort** is one of the most basic and simple algorithms. It may not be the most efficient, but it is very easy to implement. A bubble sort takes in an unsorted list and keeps comparing each element with its right side neighbour in order to sort the data. Whichever element is smaller gets shifted to the left. After completion of one round, the largest number ends up in its correct position. In other words, the largest number bubbles to the top or right in this case. Then, the process is repeated again and again until all of the data is sorted.

To swap the value of two elements,we need a ***temp***variable. This variable stores the value of *a*temporarily which is later assigned to *b****.***

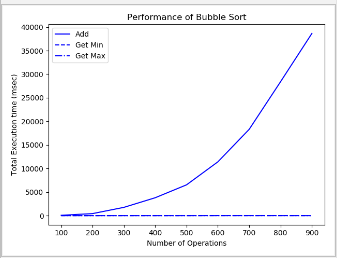
To implement the optimised version of bubblesort we must, for the first iteration, compare all the elements (n). For the subsequent runs, compare (n-1) (n-2) and so on. We then compare each element with it’s right side neighbour. Swap the smallest element to the left and keep repeating the whole process until the whole list is covered.

The pros of bubblesort are as follows:

1. Easy to understand
2. Easy to implement
3. No external memory needed.
4. Performs greatly when array is almost sorted.

The cons:

1. Expensive.
2. O(n^2) is the worst and average case
3. Requires many comparisons.
4. Slow.



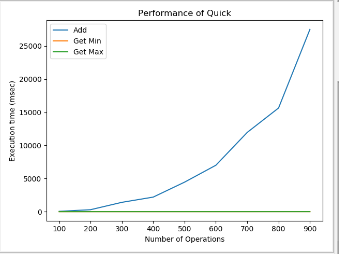
**Quicksort** is a fast sorting algorithm that takes a divide and conquer approach to sorting lists. The divide and conquer is a technique used for breaking algorithms down into sub-problems, solving the sub-problems, and then combining the results back together to solve the original problem. In this technique, we select the first element and call it the *pivot.* The idea is to group elements such that elements to the left of the pivot are smaller than the pivot and the elements to the right of the pivot are larger than the pivot. This is done by maintaining two pointers *left*and*right*. The *left*pointer points to the first element after the pivot.Similarly, the *right*pointer points to the farthest element on the right side of the list. we take an array and choose the pivot element, which we then move to the end of the array. Then, we look for two items, the item from the left which is larger than the pivot and the item from the right that is smaller than the pivot. we then swap the two items. We keep repeating until the item from the left has a higher index to the item from the right.

The pros for Quicksort are as follows:

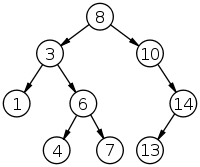
1. Easy implementation.
2. Faster than Bubblesort.
3. Average case running time of O(NlogN)

Cons:

1. Unstable.
2. Offers a worse case space complexity.
3. Worst-case still remains O(N^2)



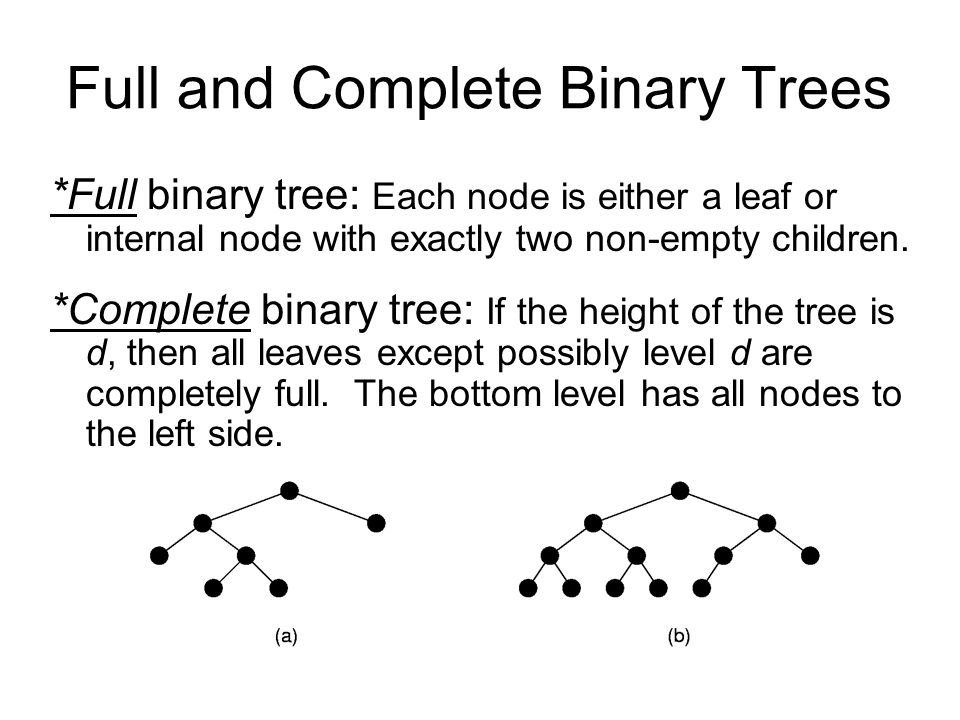
**Binary Tree** is a Python library which provides a simple API to generate, visualize, inspect and manipulate binary trees. Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures. A Binary tree is a tree data structure in which each node has at most, two children referred to as, the left child and the right child.

The top node (8) is referred to as the root. The (3) and (10) are the left and right children of the root respectively. The (3) also has two children and is referred to as the parent node of (1) and (6). The nodes found at the very bottom of the tree are called the leaves. The amount of times we have traversed down, starting from the root can be referred to as the depth. For example, starting at the root, we can call it level 1 and keep going down the tree counting all levels. In this case, the leaves are at depth 4.

There are different types of binary trees such as:

1. Complete tree.
2. Full binary tree.



**Python heap**

Heap is a special tree structure in which each parent node is less than or equal to its child node. Then it is called a Min Heap. If each parent node is greater than or equal to its child node then it is called a max heap. A heap is created by using python’s inbuilt library named heapq. This library has the relevant functions to carry out various operations on heap data structure.

Below is a list of these functions:

1. ***Heapify -*** converts a regular list to a heap. In the resulting heap the smallest element gets pushed to the index position 0. But the rest of the data elements aren’t necessarily sorted.

Input:

import heapq

H = [21,1,45,78,3,5]

heapq.heapify(H)

print(H)

Output:

[1, 3, 5, 78, 21, 45]

1. ***Heappush-*** adds an element to the heap without altering the current

heap.

Input:

heapq.heappush(H,8)

print(H)

1. ***Heappop-*** returns the smallest data element from the heap.

heapq.heappop(H)

print(H)

MaxHeap is a complete binary tree with each layer full, except the bottom layer which must be filled in from left to right.

Pros:

1. MaxHeap is very fast and we can insert new values in O(log n).
2. We can get the maximum value in O(1).
3. Easy to implement using an array.

Operations of MaxHeap:

1. Push(insert)
2. Peek(Get max)
3. Pop(remove max)



Finally we use the ***timeit*** function.

This module provides a simple way to find the execution time of small bits of python code. It has both a command-line interface and a callable one.

Pros:

1. timeit runs your snippet of code millions of time so that you get the statistically most relevant measurement of code execution time.
2. Simple to use.