test 2 notes:

open a text file and read it using readline:

```
fp=open("text.txt", "r")
line=fp.readline()
while line != " ":
  process
  the
  line(s)
  line=fp.readline()
def bubbleSort(alist):
    for k in range(len(alist)-1,0,-1):
        for i in range(k):
            if alist[i]>alist[i+1]:
                temp=alist[i]
                alist[i]=alist[i+1]
                alist[i+1]=temp
    return(alist)
def selectionSort(alist):
    for j in range(len(alist)-1,0,-1):
        aMax=0
        for k in range(1,j+1):
            if alist[k]>alist[aMax]:
                aMax=k
        temp=alist[j]
        alist[j]=alist[aMax]
        alist[aMax]=temp
    return(alist)
```

InsertionSort:It always maintains a sorted sublist in the lower positions of the list. Each new item is then “inserted” back into the previous sublist such that the sorted sublist is one item larger.
MergeSort:Merge sort is a recursive algorithm that continually splits a list in half. If the list is empty or has one item, it is sorted. Then it goes back and merges the small sublist together and keeps doing it until you have one large sorted list
QuickSort:quick sort uses divide and conquer to gain the same advantages as the merge sort, while not using additional storage. As a trade-off, however, it is possible that the list may not be divided in half. When this happens, we will see that performance is diminished.

```
class TreeNode:
    def __init__(self, key, value = None):
        self.key = key
        self.value = value
        self.parent = self.leftChild = self.rightChild = None
```

a priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it.
locating the parent:i//2
left child:2i
right child:2i+1
maxheap : every subtree of the heap has its largest value at the root
minheap : every subtree of the heap has its smallest value at the root
percUp:swapping the value of the node with the one above it to make the values decrease in value as you go down the tree.
percDown:swapping the value of the node with the one above it to make the values increase in value as you go down the tree.
balancedBST:
graph:
vertex:A vertex (also called a “node”) is a fundamental part of a graph. It can have a name, which we will call the “key.” A vertex may also have additional information. We will call this additional information the “payload.”
edge:An edge connects two vertices to show that there is a relationship between them.
weight:Edges may be weighted to show that there is a cost to go from one vertex to another.
path:a sequence of vertices that are connected by edges.
cycle:a path that starts and ends at the same vertex.
GraphADT:
adjacency matrix representation: