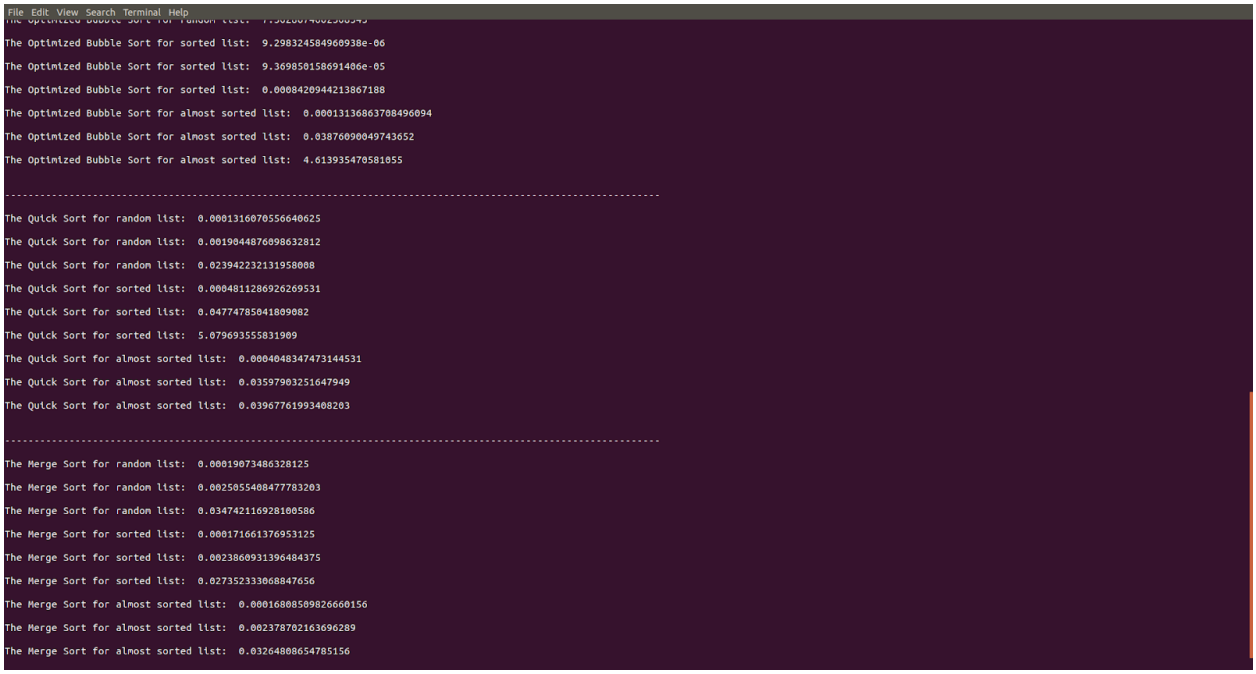
Ervin Hennrich
CMP SCI 3130

Project 2

```
File  Edit  View  Search  Terminal  Help
ervin@ervin-desktop:~/School/algo2$ ./p2.py
The selection-sort random list:  0.00032901763916015625

The selection-sort random list:  0.03334498405456543

The selection-sort random list:  3.141071319580078

The selection-sort sorted list:  0.00031876564025878906

The selection-sort sorted list:  0.031868934631347656

The selection-sort sorted list:  3.117546558380127

The selection-sort almost sorted list:  0.00032210350036621094

The selection-sort almost sorted list:  0.03185915946960449

The selection-sort almost sorted list:  3.1913676261901855

---------------------------------------------------------------------------------------------------

The insert-random list:  0.00044655799865722656

The insert-random list:  0.033860206604003906

The insert-random list:  3.5748705863952637

The insert-sorted list:  3.2901763916015625e-05

The insert-sorted list:  0.00018262863159179688

The insert-sorted list:  0.0015475749969482422

The insert-almost sorted list:  0.00010848045349121094

The insert-almost sorted list:  0.00635933876037597 66

The insert-almost sorted list:  0.46051549911499023

---------------------------------------------------------------------------------------------------

The Bubble Sort for random list:  0.0007121562957763672

The Bubble Sort for random list:  0.07410073280334473

The Bubble Sort for random list:  7.368000507354736

The Bubble Sort for sorted list:  0.0003693103790283203

The Bubble Sort for sorted list:  0.03812670707702637

The Bubble Sort for sorted list:  4.06822395324707

The Bubble Sort for almost sorted list:  0.0004146099090576172
```

```
File  Edit  View  Search  Terminal  Help
---------------------------------------------------------------------------------------------------

The Bubble Sort for random list:  0.0007121562957763672

The Bubble Sort for random list:  0.07410073280334473

The Bubble Sort for random list:  7.368000507354736

The Bubble Sort for sorted list:  0.0003693103790283203

The Bubble Sort for sorted list:  0.03812670707702637

The Bubble Sort for sorted list:  4.06822395324707

The Bubble Sort for almost sorted list:  0.0004146099090576172

The Bubble Sort for almost sorted list:  0.043732643127441406

The Bubble Sort for almost sorted list:  4.646875858306885

---------------------------------------------------------------------------------------------------

The Optimized Bubble Sort for random list:  0.00065755844116210 94

The Optimized Bubble Sort for random list:  0.0706624984741211

The Optimized Bubble Sort for random list:  7.5028674602508545

The Optimized Bubble Sort for sorted list:  9.298324584960938e-06

The Optimized Bubble Sort for sorted list:  9.369850158691406e-05

The Optimized Bubble Sort for sorted list:  0.0008420944213867188

The Optimized Bubble Sort for almost sorted list:  0.0001313686370849609 4

The Optimized Bubble Sort for almost sorted list:  0.03876090049743652

The Optimized Bubble Sort for almost sorted list:  4.613935470581055

---------------------------------------------------------------------------------------------------

The Quick Sort for random list:  0.0001316070556640625

The Quick Sort for random list:  0.0019044876098632812

The Quick Sort for random list:  0.023942232131958008

The Quick Sort for sorted list:  0.0004811286926269531

The Quick Sort for sorted list:  0.04774785041809082

The Quick Sort for sorted list:  5.079693555831909
```



#!/usr/bin/env python3.6

```
#Created by Ervin Hennrich
#Project 2 for 3130

#This program displays the running times of various sorting algorithms on lists (arrays) of random numbers, almost random numbers and already sorted lists. Each of these has a 100, 1000 and 10000 element version.
#These lists are generated then stored in another list, which is then deepcopied 6 times and sent to each of the functions to be sorted and timed

import time
import random
import copy
import sys

#Selection Sort
def selsort(linum):
    #Go through the list of lists
    for i in range(len(linum)):
        start=time.time()
        for k in range(len(linum[i])): #go through the list of numbers now
            min_idx = k
            for j in range(k+1, len(linum[i])):
                if linum[i][min_idx] > linum[i][j]:
                    min_idx = j
            linum[i][k], linum[i][min_idx] = linum[i][min_idx], linum[i][k]
        end=time.time()
        if i < 3:
            print("The selection-sort random list: ", end-start, "\n")
        if i < 6 and i > 2:
            print("The selection-sort sorted list: ", end-start, "\n")
        if i < 10 and i > 5:
            print("The selection-sort almost sorted list: ", end-start, "\n")


#Insertion Sort
def insertsort(linum):
    for k in range(len(linum)):
        start=time.time()
        for i in range(len(linum[k])):
            key = linum[k][i]
            j = i - 1
            while j >= 0 and key < linum[k][j]:
                linum[k][j+1] = linum[k][j]
```

```
            j -= 1
        linum[k][j+1] = key
    end=time.time()

    if k < 3:
        print("The insert-random list: ", end-start, "\n")
    if k < 6 and k > 2:
        print("The insert-sorted list: ", end-start, "\n")
    if k < 10 and k > 5:
        print("The insert-almost sorted list: ", end-start, "\n")


#The Optimized Bubble Sort Function
def bubsortwo(linum):
    for k in range(len(linum)):
        start=time.time()
        n = len(linum[k])
        for i in range(n):
            swapped = False
            for j in range(0, n-i-1):

                if linum[k][j] > linum[k][j+1] :
                    linum[k][j], linum[k][j+1] = linum[k][j+1], linum[k][j]
                    swapped = True

            if swapped == False:
                break
        end=time.time()
        if k < 3:
            print("The Optimized Bubble Sort for random list: ", end-start, "\n")
        if k < 6 and k > 2:
            print("The Optimized Bubble Sort for sorted list: ", end-start, "\n")
        if k < 10 and k > 5:
            print("The Optimized Bubble Sort for almost sorted list: ", end-start, "\n")


#The Bubble Sort Function
def bubsort(linum):
    for k in range(len(linum)):
        start=time.time()
        for i in range(len(linum[k])):
            for j in range(0, len(linum[k])-i-1):
            # Swapper
```

```
            if linum[k][j] > linum[k][j+1] :
                linum[k][j], linum[k][j+1] = linum[k][j+1], linum[k][j]
        end=time.time()
        if k < 3:
            print("The Bubble Sort for random list: ", end-start, "\n")
        if k < 6 and k > 2:
            print("The Bubble Sort for sorted list: ", end-start, "\n")
        if k < 10 and k > 5:
            print("The Bubble Sort for almost sorted list: ", end-start, "\n")

#The Quicksort Function
#I used Encapsulation for this function
def quick(linum):
    for k in range(len(linum)):
        start = time.time()

        def partition(arr,low,high):
            i = ( low-1 )         # index of smaller element
            pivot = arr[high]     # pivot

            for j in range(low , high):
                if   arr[j] <= pivot:
                    i = i+1
                    arr[i],arr[j] = arr[j],arr[i]

                    arr[i+1],arr[high] = arr[high],arr[i+1]
            return ( i+1 )
        def quickSort(arr,low,high):
            if low < high:
                pi = partition(arr,low,high)
                quickSort(arr, low, pi-1)
                quickSort(arr, pi+1, high)
        quickSort(linum[k], 0, len(linum[k])-1)
        end=time.time()
        if k < 3:
            print("The Quick Sort for random list: ", end-start, "\n")
        if k < 6 and k > 2:
            print("The Quick Sort for sorted list: ", end-start, "\n")
        if k < 10 and k > 5:
            print("The Quick Sort for almost sorted list: ", end-start, "\n")

#The Mergesort function
def merger(linum):
```

```
for k in range(len(linum)):
    start = time.time()
    def mergeSort(alist):
        if len(alist)>1:
            mid = len(alist)//2
            lefthalf = alist[:mid]
            righthalf = alist[mid:]

            mergeSort(lefthalf)
            mergeSort(righthalf)

            i=0
            j=0
            k=0
            while i < len(lefthalf) and j < len(righthalf):
                if lefthalf[i] < righthalf[j]:
                    alist[k]=lefthalf[i]
                    i=i+1
                else:
                    alist[k]=righthalf[j]
                    j=j+1
                k=k+1

            while i < len(lefthalf):
                alist[k]=lefthalf[i]
                i=i+1
                k=k+1

            while j < len(righthalf):
                alist[k]=righthalf[j]
                j=j+1
                k=k+1

    mergeSort(linum[k])
    end=time.time()
    if k < 3:
        print("The Merge Sort for random list: ", end-start, "\n")
    if k < 6 and k > 2:
        print("The Merge Sort for sorted list: ", end-start, "\n")
    if k < 10 and k > 5:
        print("The Merge Sort for almost sorted list: ", end-start, "\n")
```

#Need random numbers, sorted list and almost sorted lists for each 100, 1k and 10k element lists. So 9 lists in total

```
li1r = []
li1kr = []
li10kr = []
li1s = []
li1ks = []
li10ks = []
li1a = []
li1ka = []
li10ka = []
random.seed(a=None)

for i in range(1,101):
    li1s.append(i)
    li1ks.append(i)
    li10ks.append(i)
    if i == 100:
        for j in range(101, 1001):
            li1ks.append(j)
            li10ks.append(j)
            if j == 1000:
                for k in range(1001,10001):
                    li10ks.append(k)

for x in range(1,101):
    li1r.append(random.randint(1, 10000))
    li1kr.append(random.randint(1,10000))
    li10kr.append(random.randint(1,10000))
    if x == 100:
        for g in range(101,1001):
            li1kr.append(random.randint(1,10000))
            li10kr.append(random.randint(1,10000))
            if g == 1000:
                for y in range(1001,10001):
                    li10kr.append(random.randint(1,10000))

for il in range(1,101):
    li1a.append(il)
    li1ka.append(il)
    li10ka.append(il)
    if il % 10 == 0:
```

```
        li1a[il-1] = random.randint(1,10000)
        li1ka[il-1] = random.randint(1,10000)
        li10ka[il-1] = random.randint(1,10000)
    if il == 100:
        for j in range(101,1001):
            li1ka.append(j)
            li10ka.append(j)
            if j % 10 == 0:
                li1ka[j-1] = random.randint(1,10000)
                li10ka[j-1] = random.randint(1,10000)
            if j == 1000:
                for k in range(1001,10001):
                    li10ka.append(k)
                    if k % 10 == 0:
                        li10ka[k-1] = random.randint(1,10000)


#Keeping the lists in a list, making it easier to send to functions

lol = []
lol.append(li1r)
lol.append(li1kr)
lol.append(li10kr)
lol.append(li1s)
lol.append(li1ks)
lol.append(li10ks)
lol.append(li1a)
lol.append(li1ka)
lol.append(li10ka)


#Need to make a deepcopy of the list of lists, so it isnt just a reference to the original.

lol_sel = copy.deepcopy(lol)
lol_ins = copy.deepcopy(lol)
lol_bub = copy.deepcopy(lol)
lol_bub2 = copy.deepcopy(lol)
lol_quick = copy.deepcopy(lol)
lol_merge = copy.deepcopy(lol)


#Run the algos with the copies
selsort(lol_sel)
```

```
print("\n----------\n")
insertsort(lol_ins)
print("\n----------\n")
bubsort(lol_bub)
print("\n----------\n")
bubsortwo(lol_bub2)
print("\n----------\n")
sys.setrecursionlimit(10000) #default is 1000 (normally), Need to increase max recursion depth so the recursive functions can finish
quick(lol_quick)
print("\n----------\n")
merger(lol_merge)
```