CMPSC 462 – Assignment-4 (30 points)

Due date: 9/27/2022

Note: attach screenshots of your program and results under each programming exercises. Please make sure that the screenshot is readable. Don't attach a very small screenshot image.

Exercise-1: 10 points

Give an analysis of the running time (Big-Oh notation) for each of the following 2 program fragments. Note that the running time corresponds here to the number of times the operation sum++ is executed. sqrt is the function that returns the square root of a given number.

a)
```
sum = 0
for i in range(int(math.sqrt(n)/2)):
  sum+=1
for j in range(int(math.sqrt(n)/4)):
  sum+=1
for k in range(8+j):
  sum+=1
```

first loop ran sqrt(n)/2 times
2nd loop ran sqrt(n)/4 times
3rd loop ran 8 + final number of j which is sqrt(n)/4 -1
since all the loop are back to back the results are added up.

sqrt(n)/2 + sqrt(n)/4 + 8 + sqrt(n)/4 -1
sqrt(n)/2 is the dominating terms as it is the number that increases the biggest
we get ½ * sqrt(n), we can get rid of coeffience of ½
the final big O notation is O(sqrt(n))

b)
```
sum = 0
for i in range(int(math.sqrt(n)/2)):
   j=i
  for j in range(8+i):
     k=j
    for k in range(8+j):
      sum+=1
```

1st loop ran for sqrt(n)/2 times
2nd loop ran for 8 + final number of 1st loop which is sqrt(n)/2 – 1
3rd loop ran for 8 + final number of 2nd loop which is (sqrt(n)/2 – 1) – 1
since the loop are nested we multiply it together.

sqrt(n)/2 * (8 + sqrt(n)/2 - 1) * (8 + sqrt(n)/2 – 2)
we can get rid of constants and coefficients
sqrt(n) * sqrt(n) * sqrt(n)
the final big O notation is O(sqrt(n)^3)

c. If it takes 10ms to run program (b) for n=100, how long will it take to run for n=400 ?
   Let c = instructions the program can run per ms
   Sqrt(100)^3 / c = 10ms
   1000 / c = 10
   100 = c

   Sqrt(400)^3 / 100 = t
   8000 / 100 = t
   80 = t
   program b will take 80 ms to solve n = 400

d. If it takes 10ms to run program (a) for n=100, how large a problem can be solved in 40ms ?
   Let c = instructions the program can run per ms
   Sqrt(100) / c = 10 ms
   10 / c = 10
   1 = c

   Sqrt(n) / 1 = 40ms
   sqrt(n) = 40ms
   n = 40^2
   n = 1600
   program a can solve n = 1600 in 40 ms

Exercise-2: 5 points

Design an algorithm that takes two arrays, and returns true if the arrays are disjoint, i.e. have no elements in common. Write down your algorithm as pseudocode. You don't need to write a python code. Give the asymptotic analysis for time complexity in best-case and worst-case scenario.

```
Function disjoint (first, second)
        for item in first
                for thing in second
                        if item equal to thing
                                return false
        return true
```

best-case: in the best case scenario the code only need to check 1st item of both array which is constant O(1)

worst-case: the function need to loop through second array n amount of items inside the first array
n * n
which is O(n^2) time complexity.

Exercise-3: 10 points

Consider the following list. List1 = {10, 30, 95, 80, 55, 5, 60, 35} ;
What is the resulting list after two passes of the sorting phase i.e. after two iterations/recursive calls, if the following is performed? Show the steps using a rough sketch. You can draw the steps for each sorting algorithm in a paper, take a picture of it and attach it here.

a) Selection Sort: looking for smallest in the array

1st pass

{**10**, 30, 95, 80, 55, 5, 60, 35}
{**10**, **30**, 95, 80, 55, 5, 60, 35}
{**10,** 30, **95**, 80, 55, 5, 60, 35}
{**10**, 30, 95, **80**, 55, 5, 60, 35}
{**10**, 30, 95, 80, **55**, 5, 60, 35}
{**10**, 30, 95, 80, 55, **5**, 60, 35}
{**10**, 30, 95, 80, 55, 5, **60**, 35}
{**10**, 30, 95, 80, 55, 5, 60, **35**}
{***5***, 30, 95, 80, 55, ***10***, 60, 35}
5 is the smallest, switch 1st element with 5

2nd pass

{5, **30**, 95, 80, 55, 10, 60, 35}
{5, **30**, **95**, 80, 55, 10, 60, 35}
{5, **30**, 95, **80**, 55, 10, 60, 35}
{5, **30**, 95, 80, **55**, 10, 60, 35}
{5, **30**, 95, 80, 55, **10**, 60, 35}
{5, **30**, 95, 80, 55, 10, **60**, 35}
{5, **30**, 95, 80, 55, 10, 60, **35**}
{5, ***10***, 95, 80, 55, ***30***, 60, 35}
10 is the smallest of the remaining element so switch 2nd element with 10

b) Insertion Sort

1st pass

{**10**, **30**, 95, 80, 55, 5, 60, 35}
{10, **30**, **95**, 80, 55, 5, 60, 35}
{10, 30, **95, 80,** 55, 5, 60, 35}
{10, 30, ***80, 95***, 55, 5, 60, 35}
95 > 80 swap
{10, **30, 80**, 95, 55, 5, 60, 35}
30 < 80 don't swap

2nd pass

{10, 30, 80, **95, 55,** 5, 60, 35}
{10, 30, 80, ***55, 95,*** 5, 60, 35}
95 > 55 swap
{10, 30, **80, *55***, 95, 5, 60, 35}
{10, 30, ***55, 80***, 95, 5, 60, 35}
80 > 55 swap
{10, **30, 55**, 80, 95, 5, 60, 35}
30 < 55 don't swap

c) Bubble Sort

1st pass

{**10**, **30**, 95, 80, 55, 5, 60, 35}
{10, **30**, **95**, 80, 55, 5, 60, 35}
{10, 30, **95**, **80**, 55, 5, 60, 35}
{10, 30, ***80***, ***95***, 55, 5, 60, 35}
95 > 80 so swap them
{10, 30, 80, **95**, **55**, 5, 60, 35}
{10, 30, 80, ***55***, ***95***, 5, 60, 35}
95 > 55 so swap them
{10, 30, 80, 55, **95**, **5**, 60, 35}
{10, 30, 80, 55, ***5***, ***95***, 60, 35}
95 > 5 so swap them
{10, 30, 80, 55, 5**, 95**, **60**, 35}
{10, 30, 80, 55, 5**,** ***60***, ***95***, 35}
95 > 60 so swap them
{10, 30, 80, 55, 5**,** 60, **95, 35**}
{10, 30, 80, 55, 5**,** 60, ***35, 95***}
95 > 35 so swap them

2nd pass

{**10**, **30**, 80, 55, 5**,** 60, 35, 95}
{10, **30, 80,** 55, 5**,** 60, 35, 95}
{10, 30, **80, 55**, 5**,** 60, 35, 95}
{10, 30, ***55, 80***, 5**,** 60, 35, 95}
80 > 55 so swap
{10, 30, 55, **80**, **5,** 60, 35, 95}
{10, 30, 55, ***5, 80,*** 60, 35, 95}
80 > 5 so swap
{10, 30, 55, 5**, 80, 60**, 35, 95}
{10, 30, 55, 5***, 60, 80,*** 35, 95}
80 > 60 so swap
{10, 30, 55, 5, 60**, 80, 35**, 95}
{10, 30, 55, 5, 60***, 35, 80***, 95}
80 > 35 so swap
{10, 30, 55, 5, 60, 35, **80**, **95**}
80 < 95 do nothing

Exercise-4: 5 points

Assume that an Insertion sort algorithm in the worst case takes 4 minutes and 15 seconds for an input of pool size 30. What will be the maximum input pool size of a problem that can be solved in 22 minutes and 20 seconds in the worst case?

Time = 4 min and 15 secs = 4* 60 + 15 secs = 255 seconds
insertion sort time complexity is O(n^2)
let c = intructions per second

30^2 / c = 255s
900 / c = 255
900 / 255 = c

time2 = 22 * 60 + 20 = 1340s

n^2 / c = 1340
n^2 * 255 / 900 = 1340
n^2 = 900 * 1340 / 255
n^2 = 4729.41176
n = sqrt(4729.41176)
n = 68.7707188
n ~= 68
the maxium input pool size that the algorithm can solve within 22 minutes and 20 seconds are 68.