Steven Sawtelle Homework 4

1 – Linear Search

Version 1

1. take in number of test cases, and that many times execute the main loop. the main loop will consist of asking for the size of the matrix, then the value to search for. following this will be a number of integers, with that number being equal to the size^2. on each execution, it will determine if the value to be found is in the given matrix using linear search. if yes, it prints the value and its x coordinate and y coordinate. if not, it prints the value with two dashes

Version 2

1. main:
    a. take in number of test cases
    b. for that many test cases, get two values initially: the size of the n x n matrix and then the value will be searching for
    c. for the value that is the size^2, take in integers and fill them into the array in row and the column order
    d. call search() with the created matrix and value and print out returned value
2. search(Matrix, key):
    a. take in matrix and key and search through the matrix linearly to find the provided key. if found return the key with the x and y coordinate. if not return the key with two dashes

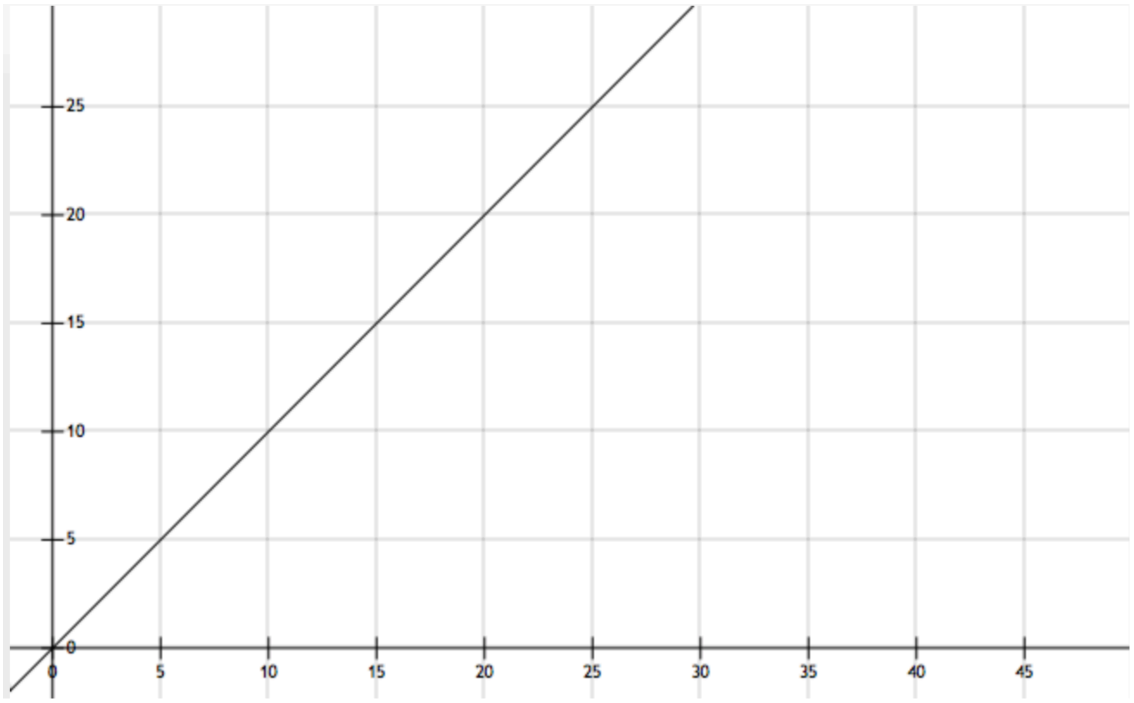
Version 3:

```
1. main:
      a. get number of test cases <- size
      b. for numbers in range 0,size <- x:
            i. get size of this matrix <- n
            ii. get key <- key
            iii. new 2d matrix <- M
            iv. 0 <- count
            v. [] <- templist
            vi. while count<n:
                  1. get next row <- templist
                  2. add templist to M
                  3. count++
            vii. print search(M, key)
            viii. clear M
2. search(M, key):
      a. if length of M is less than 1 return "/key/ - -"
      b. for numbers in range(0, length of m) <- i:
            i. for numbers in range(0, length of first element of m) <- j:
                  1. if M at i, j is equal to key return "/key/ /i/ /j/"
```

    c. return “/key/ - -“

2 – Time Complexity Analysis

The key operation in this algorithm is the comparison check that sees if the current element being checked is equal to a given key. This is performed linearly through the matrix until either this condition is met, or all elements have been exhausted. Because of that, this can be represented as the function f(n) = n after reduction. We can then say that because f(x) is O(g(x)) for all x with witnesses k=1 and c=1, that f(n) = O(n). The image below shows a graph of this relationship, with the x axis as n and the y-axis as f(n):



3 – Divide and Conquer

Version 1:

1. take in number of test cases, and that many times execute the main loop. the main loop will consist of asking for the size of the matrix, then the value to search for. following this will be a number of integers, with that number being equal to the size^2. on each execution, it will determine if the value to be found is in the given matrix using a divide and conquer approach. if yes, it prints the value and its x coordinate and y coordinate. if not, it prints the value with two dashes

Version 2:

1. main:
    a. take in number of test cases
    b. for that many test cases, get two values initially: the size of the n x n matrix and then the value will be searching for

```
        c. for the value that is the size^2, take in integers and fill them into the array in row
           and the column order
        d. call search() with the created matrix and value as well as starting coordinates
           and ending coordinates and print out returned value
    2. search(Matrix, smallx, smally, highx, highy, key):
        a. take in matrix and key and search through the matrix using the coordinates to
           find the provided key. this is done by breaking the matrix into parts by
           subtracting or adding 1 to the appropriate parameters and calling search() again
           with these new values. if it is equal, return the key with the x and y coordinate. if
           not return -1 -1

Version 3:
    1. main:
        a. get number of test cases <- size
        b. for numbers in range 0,size <- x:
               i. get size of this matrix <- n
              ii. get key <- key
             iii. new 2d matix <- M
              iv. 0 <- count
               v. while count<n:
                      1. get next row <- templist
                      2. add templist to M
                      3. count++
              vi. search(M, 0, 0, len(m)-1, len(m[0])-1, key) <- resr, resc
             vii. if resr != -1 and resc != -1
                      1. print "/key/ /resr/ /resc/"
            viii. else
                      1. print "/key/ - -"
    2. search(M, lowr, lowc, highr, highc, key):
        a. if lowr > highr or lowc > highc return -1, -1
        b. ceiling of lowr+highr/2 <- midr
        c. ceiling of lowc+highc/2 <- midc
        d. if M at midr, midc equals key return midr, midc
        e. if key is less than this:
               i. foundr, foundc = search(m, lowr, lowc, midr-1, highc, key)
              ii. if foundr == -1 or foundc == -1
                      1. return search(m, midr, lowc, highr, midc-1, key)
             iii. otherwise return foundr, foundc
        f. otherwise:
               i. foundr, foundc = search(m, lowr, midc+1, highr, highc, key)
              ii. if foundr == -1 or foundc == -1
                      1. return search(m, midr+1, lowc, highr, highc, key)
             iii. otherwise return foundr, foundc
```

5 – see attached hw4-linear.py

6 – see attached hw4-divide.py

4 not completed as instructed