## Problem 1.

A demonstration of advanced debugging techniques is here: `https://www.youtube.com/watch?v=tM9YUWVQ26A`.

Download the files `gomoku.py` and `tester.py`. Demonstrate the following techniques to the TA.

- Debugging an infinite loop by interrupting the program and using postmortem debugging
- Identifying a failing test case, extracting the input that would cause the test case to fail, and then tracing through the function that returns the incorrect value

You do not need to actually debug the given `gomoku.py` (that's impossible – you'd just have to write the functions from scratch). You just need to show the TA that you are familiar with the techniques.

I recommend that you work in Pyzo on the ECF machines.

In both PyCharm and VS Code, postmortem debugging is somewhat possible by default: the IDE takes you to the place where the crash occurred. However, this is difficult to achieve after a keyboard interrupt in VS Code. In PyCharm, you can use the "pause" button to achieve the same effect. Just use Pyzo for this question, even if you usually use a different IDE.

## Problem 2.

Here is the Binary Search code from lecture

```
def binary_search(L, e):
   low = 0
   high = len(L)-1
   while high-low >= 2:
      mid = (low+high)//2 #e.g. 7//2 == 3
      if L[mid] > e:
         high = mid-1
      elif L[mid] < e:
         low = mid+1
      else:
         return mid
   if L[low] == e:
      return low
   elif L[high] == e:
      return high
   else:
      return None
```

### Part (a)

Demonstrate that the code works on sorted lists of size 10 by inputting sample inputs and making sure that the code runs.

### Part (b)

Modify the function to return the *number of iterations* that the while loop runs for. (Reminder: a function can return a tuple: the first element might be the index, and the second element the number of times that the while loop ran for)

### Part (c)

In the worst case, binary search does not "return early" because `L[mid]` is not equal to `e`. What would be a way to construct a list L such that that never happens?

### Part (d)

For list sizes n being 10, 100, 1000, 10000, ..., 10000000, output the number of iterations that the while loop takes, in the worst case, by constructing lists such that the number of iterations is as large as it can be for the given list size.

### Part (e)

You can get the number of seconds since Jan 1, 1970 using `time.time()` after using `import time`. This means you can measure the time a function took to run using

```
import time
start = time.time()
#run function
end = time.time()
print(end-start)
```

Compare the runtime of "linear search" (using L.index()) and binary search on input sizes 10, 100, 1000, 10000, ..., 10000000, in the worst case, and output the results.

Make your code as nice as possible: for example, write a function that measures the runtime rather than copy-pasting the code above multiple times.

## Problem 3.

For this question, you will compute word counts in a large corpus of text. Word frequency lists (a frequency of the word is the ratio of the number of times the word appears to the total number of words in a document) are commonly used in computational linguistics. You will write a function that finds the 10 most frequently occurring words in a text file, and a program that uses this function to find the 10 most frequently-occurring words in *Pride and Prejudice*. For the purposes of this problem, you may assume, if you like, that all words are separated by spaces and that there is no punctuation in the file, and all the words are in lowercase, so that the list of words in the file text.txt is given by

```
open("text.txt", encoding="latin-1").read().split()
```

### Part (a)

First, store the number of times that the word w appears in the text in `word_counts[w]`. For example, if the word "the" appears in the file 5 times, `word_counts["the"]` should be 5.

For example, if the contents of your file are the opening of *Notes from the Underground* by Fyodor Dostoyevsky, translated by Constance Garnett:

> I am a sick man. I am a spiteful man. I am an unattractive man. I believe my liver is diseased. However, I know nothing at all about my disease, and do not know for certain what ails me.

, and you do not process the text in any way other than using `read().split()`, the list of words will be:

```
["I", "am", "a", "sick", "man.", "I", "am", "a", "spiteful", "man.", "I", "am",
"an", "unattractive", "man.", "I", "believe", "my", "liver", "is", "diseased.",
"However,", "I", "know", "nothing", "at", "all", "about", "my", "disease,", "and",
"do", "not", "know", "for", "certain", "what", "ails", "me."]
```

`word_counts` should then be

```
{"sick": 1, "man.": 3, "at": 1, "what": 1, "nothing": 1, "do": 1, "is": 1, "me.": 1,
"I": 5, "ails": 1, "an": 1, "am": 3, "know": 2, "disease,": 1, "not": 1, "liver": 1,
"believe": 1, "all": 1, "my": 2, "certain": 1, "However,": 1, "and": 1, "for": 1,
"unattractive": 1, "spiteful": 1, "about": 1, "a": 2, "diseased.": 1}
```

For example, the word `"man."` (with the period at the end) appears 3 times in the text, so its entry in the dictionary `word_counts` is 3.

Create the file `test.txt` using Pyzo, and test your function using that file. Use a small file so that you can check that your function works correctly.

## Part (b)

Write a function with the signature `top10(L)` that takes in a list L of 100 different integers, and returns a list of the 10 largest integers in `L`.

## Part (c)

Now, obtain the top 10 most-frequent words from the dictionary `freq`. To do that, you need to sort the data by the word counts. You cannot sort dictionaries directly, but you can use the following trick:

```
inv_freq = {6: "the", 12: "a", 1:"hi"}
print(sorted(inv_freq.items()))
```

First, experiment with this code and understand what it is doing, and then apply the technique to finding the top 10 most frequent words. Test your function by creating a small text file where you can find the top n most-frequent words manually, running your code on this file, and comparing the results. Then, download the text of Pride and Prejudice from `http://www.gutenberg.org/files/1342/1342-0.txt` and obtain the top 10 most frequent words in Pride and Prejudice.