15-110: Principles of Computing, Spring 2018

Problem Set 4 (PS4)

Due: Friday, February 16 by 2:30PM via Gradescope Hand-in

HANDIN INSTRUCTIONS

Download a copy of this PDF file. You have two ways to fill in your answers:

1. **Just edit (preferred) - Use any PDF editor (e.g., Preview on Mac, iAnnotate on mobile, Acrobat Pro on pretty much anything) to typeset your answers in the given spaces. You can even draw pictures or take a picture of a drawing and import it in the correct place in the document. That's it. (Acrobat Pro is available on all cluster machines.)**

2. **Print and Scan - Alternatively, print this file, write your answers neatly by hand, and then scan it into a PDF file. This is labor-intensive and must be done by the deadline.**

Once you have prepared your submission, submit it on Gradescope. A link to Gradescope is provided in our Canvas course portal.

Fill in your answers ONLY in the spaces provided. Any answers entered outside of the spaces provided may not be graded. Do not add additional pages. We will only score answers in the given answer spaces provided. If we cannot read your answer or it contains ambiguous information, you will not receive credit for that answer.

Be sure to enter your full name below along with your section letter (A, B, C, etc.) and your Andrew ID. Submit your work on Gradescope by 2:30PM on the Friday given above.

REMINDER: Sharing your answers with another student who is completing the assignment, even in another semester, is a violation of the academic integrity policies of this course. Please keep these answers to yourself.

Name (First Last) ____

Section __________ Andrew ID ______________________________

1. (3 pts) In class, we discussed the linear search algorithm, shown below in Python:

```
def search(datalist, key):
     index = 0
     while index < len(datalist):
         if datalist[index] == key:
             return index
         index = index + 1
     return None
```

Suppose that we know the additional facts that the list is sorted in "*increasing order*" (i.e. `datalist[i]` ≤ `datalist[i+1]`, for all `i` such that 0 ≤ `i` < `len(datalist)`-1. For example, if our list has the values [25, 37, 45, 61, 73, 79, 82, 90], then if we want to search for the key 70 using linear search, we can stop when we reach 73 and return `None` (since we know the list is sorted in increasing order and we can't possibly find 70 once we encounter 73).

a. Revise the function above so that it also returns `None` as soon as it can be determined that the key cannot be in the list, assuming that the list is sorted in increasing order. (HINT: You only need to add an additional instruction inside the `while` loop.)

b. You are a historian and you have a stack of photographs, ordered in increasing order from earliest year to latest year. You use linear search to search for a specific year in the stack. Let's say that it takes you 20 seconds to search the stack of photos in the worst case using our search algorithm. Suppose someone gives you 12 times as many photographs, also ordered from earliest year to latest year. In this case, approximately how long would it take to search for a specific photo in the worst case using our search algorithm, expressed in minutes?

c. In order to use your new function from (a), you should probably have a method that allows you to check to make sure that the list is sorted in increasing order before you use the search method. Complete the Python function `is_sorted(datalist)` below that returns the boolean `True` if a given list is sorted in increasing order or boolean `False` if it is not.

   Your function will be similar to the search function on the previous page, except you don't need a key to search for. For each index in the loop, if datalist[index] is not less than or equal to datalist[index+1], you know immediately that the list is not sorted in increasing order, so you can exit. (What do you return?) If it is less than or equal, then repeat with the next index. If you get through the entire list, then the loop will finish and you know the list is sorted in increasing order. (What do you return in this case?)

```
def is_sorted(datalist):

     for index in range(__________, ____________________):

          if datalist[index] > datalist[index+1]:

               return ________________

     return ________________
```

d. We replace the `for` loop above with the iterator version that does not specify the range:

```
     for element in datalist:

          if element > element+1:
```

   etc.

   This does not function correctly. Why?

2. (2 pts) If a list is sorted in increasing order, we can search the list using another algorithm called Binary Search. The basic idea is to find the middle element, then if that is not the key, you search either the first half of the list or the second half of the list, depending on the half that could contain the key. The process is repeated iteratively until we either find the key or we run out of elements to examine. Here is an implementation of Binary Search in Python:

```
def bsearch(datalist, key):
     lo_index = -1
     hi_index = len(datalist)
     while lo_index + 1 != hi_index:
         mid = (lo_index+hi_index)//2
         if datalist[mid] == key:
             return mid
         if key > datalist[mid]:
             lo_index = mid
         else:
             hi_index = mid
     return None
```

Let `datalist =`

`[5, 12, 14, 19, 23, 27, 33, 39, 45, 56, 61, 70, 79, 81, 98]`.

a. Trace the function above for the function call `bsearch(datalist, 81)`, where the key is 81, showing the values of `lo_index` and `hi_index` after each iteration of the while loop is completed. Also, write down the value returned by the function. We have started the trace with the initial values of `lo_index` and `hi_index`.

```
lo_index  -1 |                                          returns:
---------------------------------------------
hi_index  15 |                                          _________
```

b. Trace the function above for the function call `bsearch(datalist, 13),` where the key is 13, and write down the value returned by the function.

```
lo_index  -1 |                                          returns:
---------------------------------------------
hi_index  15 |                                          _________
```

3. (2 pts) Using the binary search function from the previous problem, answer the following questions clearly and concisely.

   a. If the list has an even number of elements, how does the function determine the location of the "middle element"? Give an example to illustrate your answer.

   b. If the list has 7 elements, what is the maximum number of elements that will be examined in binary search?

      If the list has 15 elements (approximately twice as many elements as before), what is the maximum number of elements that will be examined in binary search?

      If the list has 31 elements (approximately twice as many elements again), what is the maximum number of elements that will be examined in binary search?

      If the list has $2^n - 1$ elements, for $n > 0$, what is the maximum number of elements that will be examined in binary search as a function of n? (Look for a pattern above.)

4. (3 pts) In class, we developed an "in place" version of Insertion Sort. For each element at index i, what we did was remove that element, then search from the beginning of the list L up to position i for the location of the first value that was greater than the removed element. We then inserted the removed element back into the list at that location.

Here is another version of Insertion Sort. In this version, for each element at index i, we copy (not remove) the element into x. Now, we search backwards from position i-1 for the first value that is less than or equal to x. As we do this, we copy each value we look at over one position. Once we find the location of the first value that is less than or equal to x, we copy x into the position after that location.

For example, suppose `i` is 6 and we have the following so far in our list `L` (note that `L[0..6)`, shown in red, is sorted already):

```
L = [14, 26, 30, 53, 76, 91, 68, 42]
i =                           6
```

We copy 68 into `x`. `(x = 68)` We don't remove it.

Now we look at 91 at index 5. Since 91 is not less than or equal to `x`, we copy 91 over one position:

```
L = [14, 26, 30, 53, 76, 91, 91, 42]
i =                           6
```

Now we look at 76 at index 4. Since 76 is not less than or equal to `x`, we copy 76 over one position:

```
L = [14, 26, 30, 53, 76, 76, 91, 42]
i =                           6
```

Now we look at 53 at index 3. Since 53 is less than or equal to `x`, we have found our insert point (i.e. after 53, at index 3+1=4), so we copy x into position 4 and add 1 to `i`:

```
L = [14, 26, 30, 53, 68, 76, 91, 42]
i =                               7
```

Note that `L[0..7)` is now sorted (shown in red). This process is repeated for each index `i` = 1,2,...

Before we implement this version, we should note that there is a special case to consider! As we search backward from position i-1 for the first element that is less than or equal to x, all of these values could be greater than x. For example, if we try to insert x=14 into `L[0..4)` below:

```
L = [26, 30, 53, 76, 14, 91, 68, 42]
i =                   4
```

In this case, as we search backward, we will fall off the list, which means we copy x into index 0.

(a) Complete this new version of Insertion Sort below. Note that if you complete this correctly, it will handle the special case as well.

```
def isort(datalist):

    i = 1

    while i < len(datalist):

        # copy the element at index i into x:

        ____________________________________________

        # j keeps track of each position as we search backward:

        j = i - 1

        # while j is valid and the element at index j is greater than x:

        while ___________________ and datalist[j] > x:

            # copy the element at index j into index j+1:

            _______________________________________________

            # step j backwards:

             j = j – 1

        # we found the insert position (j+1), so copy x there

        _______________________________________________

        i = i + 1

    return datalist
```

REMEMBER: You can enter this into a file and run it in python3 to try out your solution!

(b) In class, we saw that Insertion Sort is $O(n^2)$ in the worst case. This means that the amount of work our computation does is proportional to the square of the amount of data (n).

If we double the amount of data, this algorithm will run approximately _________ times longer.

If we triple the amount of data, this algorithm will run approximately _________ times longer.