Midterm Exam 1B Answer key

15110 Principles of Computing Spring 2015

February 18, 2015

Name: ____________________

Andrew ID: ____________ Lab section: ____________

Instructions

- Answer each question neatly in the space provided.
- There are 7 questions totaling 34 subproblems on 15 pages. Not all problems are the same size or difficulty.
- Please read each question carefully. You have 50 minutes for this exam. No electronic devices allowed. Good luck!

	Max	Score
1	6	
2	16	
3	20	
4	21	
5	22	
6	10	
7	5	
Total:	100	

Questions

1. This question deals with history of computing devices.

   (a) (2 points) What is the significance of the invention of transistors in the history of computing?

   Solution: Transistors replaced vacuum tubes as building blocks for computers. Their invention led to increased reliability, and reduced size of the computers.

   (b) (2 points) Moore's Law says that the number of integrated circuit chips in a computer doubles every 2 years, which implies that computers become twice as powerful every 2 years. According to Moore's Law, how many years from now will computers be 64 times more powerful as they are now?

   Solution: $64 = 2^6$, which implies 6 doublings, 12 years.

   (c) (2 points) A Gigabyte (GB) is 2^{30} Bytes and a Megabyte (MB) is 2^{20} Bytes . If you have a storage device with a capacity of 16 GB, how many 32MB files can you fit in that device? Express the result as a power of 2.

   Solution: 16 GB = $2^4 * 2^{30}$ 32 MB = $2^5 * 2^{20}$ Answer: $2^{34}/2^{25} = 2^9$

2. This question focuses on expressions, data types, and variable assignments.

   (a) (4 points) For each of the following Python expressions, write down the value that is output when the expression is evaluated using a Python interpreter. Write **error** if you think the expression will raise an error.

   - 20 % 4

     Solution: 0

   - 12 + 3 * 5 == 75

     Solution: False

   - `"CS" + "15110"`

     Solution: `"CS15110"`

   - `"100" + 150`

     Solution: error

   (b) (4 points) Suppose that we type the following assignments and expressions in a Python shell **in the given order**. For each of the expressions below write down the value that will be output by Python.

```
>>> a = 20
>>> b = 30
>>> c = a + b
>>> a + 5

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

>>> a + b

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

>>> b = b + 1
>>> c

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


>>> a + b

------------------
```

   Solution: 25, 50, 50, 51

(c) (4 points) Assume the following list definition in Python.

```
>>> letters = ["a", "b", "o", "c", "p"]
```

What would be displayed in a Python shell for each of the following expressions if they are evaluated in the given order? If it would give an error then write **error**.

```
>>> letters[1]



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

>>> letters[len(letters)-2]



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


>>> letters + ["x"]



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

>>> letters



------------------
```

Solution:

```
"b"
"c"
["a", "b", "o", "c", "p","x"]
["a", "b", "o", "c", "p"]
```

(d) (2 points) Show how to create a list of every integer between 0 and 1000 , inclusive, named `nums1` using Python, sorted in increasing order.

```
nums1 = list(_______________________________________)
```

Solution: range(0,1001) or range(1001)

(e) (2 points) Let `nums3` and `nums2` be two non-empty lists. Write a Python command that will append the last element of `nums2` to the end of `nums3` .

```
________________.append(_____________________).
```

Solution: nums3.append(nums2[len(nums2)-1]) OR nums3.append(nums2[-1])

3. This question focuses on the basics of Python functions and tracing.

   (a) (3 points) In economics, the percentage rate of inflation for a period of time is calculated based on the final value F of a commodity and the initial value I of the commodity, using the formula $((F - I)/I) \times 100$. Write a Python function `inflation_rate(initial, final)` to compute and **return** the inflation rate given the initial and final values of a commodity.

   Solution:

   ```
   def inflation_rate(initial,final):
       return ((final - initial)/ initial) * 100
   ```

   (b) (4 points) **Using the function from part (a)**, write a Python function `average_inflation_rate()`, that computes and returns the average rate of inflation for the 3-year period represented in the table below:

   ```
   Year    Initial value    Final value
   1       23.50            24.00
   2       24.00            24.25
   3       24.25            24.38
   ```

   The function is **required** to call the function from part (a).

   Solution:

   ```
   def average_inflation_rate():
       year1 = inflation_rate(23.50,24.00)
       year2 = inflation_rate(24.00,24.25)
       year3 = inflation_rate(24.25,24.38)
       return (year1 + year2 + year3)/3
   ```

   (c) (3 points) Consider the simple function given below.

   ```
   def sq(n):
       print(n**2)
   ```

   When we compare `sq(5)` with 25, the Python interpreter would return False after printing 25 as shown below. Explain why it gives False as a result of the comparison.

   ```
   >>> sq(5) == 25
   25
   False
   ```

   Solution: Since $n**2$ is printed, not returned, sq(5) would return None. Comparing a None value to an integer would yield False.

   (d) (6 points) Consider the following Python function where `m` and `n` are assumed to be a positive integers:

   ```
   def mystery(n, m):
       p = 1
       e = 0
       while e < m:
   ```

```
        p = p * n
        e = e + 1
    return p
```

Trace this function for n = 4, m = 3, showing the value of e and p in the table above at the end of each iteration of the loop. The initial values of p and e are given for you in the table. Use as many spaces as you need.

```
 p              e
===================
 1              0


 ---            ---

 ---            ---

 ---            ---

 ---            ---
```

Solution:

```
 p              e
===================
 1              0

 4              1

16              2

64              3
```

(e) (2 points) Which of the following functions is being computed by mystery above? Circle your answer.

1. nm
2. $n + m$
3. n^m
4. m^n
5. None of these

Solution: (3)n^m.

(f) (2 points) Suppose that the return statement was indented as below. What would mystery(4, 3) return in this case?

```
def mystery(n, m):
    p = 1
    e = 0
    while e < m:
        p = p * n
        e = e + 1
        return p
```

Solution: 4

4. This question focuses on searching.

   (a) (5 points) Below is a Python function that takes a list of integers `nums` and an integer `root` as inputs, and searches for a number in the list that is the square of the value of `root`. It returns the **index** of the first occurrence of such a number, or None if there is no such number. For example, when the function is called with [101, 4, 12, 24] for `nums` and 2 for `root` it should return 1. This is because 4 is the first item in the list that is the square of 2. **Complete the missing parts of the function.**

```
def first_square(nums, root):

    for __________________________________:


        if ____________________________________:


            return ______________________________


    return _______________
```

Solution:

```
def first_square(nums, root):
    for i in range(len(nums)):       (1 point)
        if nums[i] == root * root:   (2 points)
            return i                 (1 point)
    return None                      (1 point)
```

Assuming that the list `vals` is [100, 66, 55, 64, 41, 35, 18, 64], write the output from each of the following calls to `first_square`.

   (b) (2 points) `first_square(vals, 8)`

Solution: 3

   (c) (2 points) `first_square(vals, 5)`

Solution: None

   (d) (2 points) How many times would the for-loop iterate if we evaluated `first_square(list(range(10, 0, -1)), 12)`?

Solution: 10

(e) (5 points) Suppose we want to return, not the first, but the last element in the list that is the square of the parameter `root`. **Complete the function definition below to accomplish this task.** If you're not able to make your answer conform to the partial definition, write your answer in the space provided.

```
def last_square(nums, root):

    for __________________________________:


        if ____________________________________:


            return ____________________________


    return _______________
```

Solution:

```
def last_square(nums, root):
    for i in range(len(nums)-1, -1, -1):
        if nums[i] == root * root:
            return i
    return None
```

or

```
def last_square(nums, root):
    answer = None
    for i in range(len(nums)):
        if nums[i] == root * root:
            answer = i
    return answer
```

or the equivalent while loops

(f) (2 points) Supposing the length of a list `vals` is fixed at n elements, what kind of list contents would result in the longest running time when evaluating `first_square(vals, 2)`?

Solution: A list not containing the number 4.

(g) (2 points) Supposing the length of a list `vals` is fixed at n elements. What kind of list contents would result in the shortest running time when evaluating `first_square(vals, 2)`?

Solution: A list with the value 4 as its first element.

(h) (1 point) Assuming the input list has length n, give the big-O notation for the **worst-case** time complexity for `first_square` .

Solution: $O(n)$

5. This question focuses on searching and sorting.

(a) (6 points) Recall that the recursive binary search algorithm we studied in class works with a range to search defined by two indexes: `lower` and `upper`. The value of `lower` is one less than the index of the first element in the range, and the value of `upper` is one greater than the index of the last element in the range. Finally, the algorithm also computes the midpoint `mid = (lower + upper) // 2`.

Suppose we do a binary search in the list

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32]

for the value 9. **Complete the table below to show how the values of `lower`, `upper`, and `mid` change during the search, until and including the point where the base case is reached.** (You should not need all the space provided.)

lower	upper	mid
-1	17	8

Solution: Final value of mid is optional.

lower	upper	mid
-1	17	8
-1	8	3
3	8	5
3	5	4
4	5	(4)

(b) (2 points) Give the big-O notation for the **worst-case** time complexity of binary search of a list of n elements.

Solution: $O(\log n)$

(c) (5 points) Below is a list of functions you have seen that describe the running times of algorithms for inputs of size n. Order them by asymptotic rate of growth (smallest rate to largest rate) by writing the corresponding numbers in order in the space provided.

1. $n \log n$
2. 1
3. n
4. n^2
5. $\log n$

Solution: 1, $\log n$, n, $n \log n$, n^2 so 2, 5, 3, 1, 4

(d) (1 point) What kind of input causes the binary search algorithm to exhibit its *worst-case* time complexity?

Solution: The key is larger than any element of the list. A slightly inaccurate but acceptable answer: the key does not occur in the list.

(e) (4 points) In order to use binary search, we must have a sorted list. Suppose we begin with an unsorted list of length n, sort it using mergesort, and then search it once using binary search. Give the big-O notation for the **worst-case** time complexity of the whole process. For full credit, make sure you give the smallest and simplest function possible, and show your work. (Remember, if there are multiple terms, you should throw out the ones with lower growth rates)

Solution: Mergesort: $O(n \log n)$, binary search: $O(\log n)$, both together: $O(n \log n + \log n)$ which simplifies to $O(n \log n)$ because $n \log n$ has a higher rate of growth than $\log n$.

(f) (2 points) If an algorithm has worst-case time complexity $O(\log n)$ for an input of size n, doubling the input size increases the running time by how much? Circle your answer.

1. the running time is squared
2. a factor of two
3. a factor of four
4. zero
5. a constant amount

Solution: (5) a constant amount

(g) (2 points) Suppose we sort a list of 8 elements using mergesort. There will be eight single-element lists to be pairwise merged.

List the other merge operations needed to complete the mergesort, giving the list sizes and number of lists of each size. For instance, you might say "five 3-element lists and two 5-element lists" (except that would be wrong!)

Solution: four 2-element lists and two 4-element lists

6. This question deals with correctness of functions and testing.

(a) (2 points) Complete the following function definition so that it finds the first pair of adjacent items in the input list that are equal to each other, and returns the index of the first element of the pair. If there is no such pair, it returns None. Example:

```
>>> search_pair([1,3,4,6,6,7,7])
3
```

Notice that 3 is the index of the first 6, which is the first element of the first pair of items that are equal to each other. (It may be helpful to remember that `range(n)`is empty, if `n` is 0 or less, and a for-loop on an empty range skips the loop body.)

```
def search_pair(items):
    for i in range(__________________________________):
        assert i+1 < len(items)
        if items[i] == items[i + 1]:
            return i
    return None
```

Solution:

```
def search_pair(items):
    for i in range(len(items) - 1):
        assert i+1 < len(items)
        if items[i] == items[i + 1]:
            return i
    return None
```

(b) (2 points) Explain in one sentence what purpose the `assert` statement serves in this function.

Solution: The least informative acceptable answer is: to stop the program if $i + 1 \geq$ the length of the input list.

The best answer is: to check that the list index $i + 1$ is a legal index, and to document for the reader that list index $i + 1$ is ok to use in the loop.

(c) (6 points) Below is a test function for `search_pair`. It only tests two kinds of input. Add three test cases: one for an empty list, one for a list containing only one element, and one for a list where the only two adjacent elements equal to each other are the last two.

```
def test_search_pair():
    assert search_pair(list(range(10))) == None
    assert search_pair([4, 5, 5, 6]) == 1




    print("Test complete")
```

Solution:

```
assert search_pair([]) == None
assert search_pair([1]) == None
assert search_pair([5, 4, 4]) == 1
```

7. This question is based on your readings from the book Blown to Bits.

   (a) (3 points) Here is a quote from the book:

   > Imagine one sick person infecting two healthy people, and the next day each of those two infects two others, and the next day after that each of those four infects two others, and so on. The number of newly infected each day grows from two to four to eight. In a week, 128 people come down with the disease in a single day, and twice that number are now sick, but in a population of ten million, no one notices. Even after two weeks, barely three people in a thousand are sick. But after another week, 40% of the population is sick, and society collapses.

   The hypothetical epidemic overwhelms society at the end of three weeks, with 2^{22} people infected. At what point in time was the epidemic only half as severe, with 2^{21} people infected?

   Solution: The day before the end of three weeks.

   (b) (2 points) Event Data Recorders (EDRs) are common in modern cars. What kind of data do they record, and why do police sometimes download the data from cars involved in an accident?

   Solution: They record data like speed, braking information, use of turn signals, and use of seat belts. Police take the data for accident reconstruction.