3/18/22

Exam 1 Review

① Write a short Python program that takes two arrays a & b of length n storing int values and returns the dot product of a & b. That is, it returns an array c of length n such that

$$c[i) = a[i] \cdot b[i], \text{ for } i=0 \ldots, n-1$$

```
a=[1,2,3]
b=[4,5,6]

def dot_product(a,b):
    if len(a) ! = len(b):
        raise ValueError("a and b must be the
                                  same length")

    c=[0] * len(a)
    for i in range(len(a)):
        c[i] = a[i] * b[i]
    return sum(c)
dp=dot_product(a,b)
print(dp)
```

② Class Vehicle, has three instance variables of type str, int, and float. Name of vehicle, its year, & price. Include constructor method that initializes each variable to appropriate value. Include methods for setting the value of each type and retrieving the value of each type

↳CH2, pg 70

```
class Vehicle:
  //constructor
  def __init__(self, name, year, price):
      self._name =name
      self._year = year
      self._price =price
  // retrieving values
  def get_name(self):
      return self._name

  def get_year(self):
      return self._year

  def get.price(self):
      return self._price
```

```
//setting values
def set_name(self, name):
    self._name=name
def set_year(self, year):
    self._year =year
def set_price(self, price):
    self._price =price
```

③

$N \cdot \log(n)$: the problem says $\log(n)$ for each number

$N \cdot N$: for the other case

best case: $O(\log n)$ → odd numbers

↳ only execute $O(\log n)$ statements

worst case: $O(n)$ → even numbers

↳ only execute $O(n)$ statements

4

↳ the requirement of binary search algorithm is that the sequence is sorted

↳ $O(\log n)$ if sorted

↳ Not sorted:

- ↳ search for target value w/ loop
- ↳ examine each element until found or end
- ↳ $O(n)$ complexity

5

```
def binary_search(data, target, low, high):
    ''' Return True if target is found in indicated
    portion of a Python list. The search only
    considers the portion from data[low] to
    data[high] inclusive.'''

    if low > high:
        return False # interval empty; no match

    else:
        mid = (low + high) // 2
        if target = = data[mid]: # found match
            return True
        elif target < data[mid]:
            #recur on the portion left of the middle
            return binary_search(data, target, low, mid-1)
        else:
            #recur on the portion right of the middle
            return binary_search(data, target, mid+1, high)
```

6

```
S = [ . . . ]
def insertion_sort(s):
    for k in range(1, len(s)): # from 1 to n-1
        cur = S[k] #current element to be inserted
        j = k #find correct index j for current
        while j >0 and S[j-1] > cur:
        #element A[j-1] must be after current
            S[j] = S[j-1]
            j -= 1
        S[j] = cur
```

↳ worst case → $O(n^2)$

↳ sorted() → $O(\log n)$

7

```
S = [ . . . ]
L= [ . . . ]

S. extend (L)
 print(S)
```

8 CH5 pg 219-221

Scores = [[100] * $\underset{C}{5}$ for j in range $(\underset{R}{3})$]

scores[1][1] = 50

3 x 5 matrix

$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 2 & 3 & 4 \\
1 & 5 & \text{\textcircled{6}} & 7 & 8 & 9 \\
2 & 10 & 11 & 12 & 13 & 14
\end{array}$$