CSci 242: Algorithms and Data Structures **Fall, 2019**

Instructor: Dr. M. E. Kim Date: October 24, 2019

Due: by the end of day, November 1st (Fri.), 2019. Name : Elena Corpus

**Home Assignment 5: 60 points + 90 (implementation) + 10 (Optional) 154/150**

Q1. [10] **Quick Sort**

Suppose the quick-sort algorithm is modified so that the pivot is always chosen at index ë*n*/2û, i.e. an element in the middle of the sequence. What is the running time of this modified quick-sort on a sequence that is *already sorted*?

Since the array is already sorted, quick sort takes O(N log N), because pivot will always be the median value. Thus, resulting in the arrays being made in half the size during the two recursive calls.

Q2. [10] **Ordering**

Suppose we’re given a sequence S of *n* elements, each of which is colored red or blue. Assume S is represented as an array. Give an in-place method, Blue-Red(S, *n*), in a pseudo code for ordering S so that all the blue elements are listed before all the red elements.

Blue-Red(S,n)

Input: an array S[0, … , n – 1]

Output: an array S[0, … , n – 1] that all blue elements are listed before all the red elements

b<-0

r<- n- 1

While b <= r do

If S[b] = “b”

b <- b + 1

Else

Swap (S[b], s[r])

r <- r – 1

Q2B [10, optional]

In Q2, suppose each element is colored red, blue or green. Give an in-place method in a pseudo code for ordering S so that all the blue elements are listed before all the red ones while all the red ones are listed before green elements.

Blue-Red-Green (S, N)

b<- 0

r<-0

g<- n-1

while r <= g do

if S[r] = “r”

swap (S[b], S[r])

r<- r+1

b<- b + 1

elif s[r]= “b”

r<-r+1

else S[r] = “g”

swap (S[r], S[g])

g<-g – 1

Q3. [10] **Inversion**

Let S be an array of *n* elements on which a total order relation is defined. An inversion in S is a pair of indices *i* and *j* such that *i* < *j* but S[*i*] > S[*j*]. Write an Count-inversion(S) algorithm that runs in O(*n* log *n* ) time for determining the number of inversions in S. The beginning index in S is 1, i.e. S[1 .. *n*].

Hint: Try to modify the merge-sort algorithm to solve it.

Merge(array, tempArray, left, right)

Input : two arrays

Output: merged array in sorted order

count = 0

if right > left, then

mid = (left + right)/2

count = Merge(array, tempArray, left, mid)

count =+ Merge(array, tempArray, mid + 1, right)

count =+ Merge(array, tempArray, left, mid +1, right)

return count

Q4. [10] **In-Place** **Quick-Selection**

Give the ***in-place quick-select*** algorithm that selects *kt*h ***largest*** element in the array A of *n* elements in a pseudo code. i.e. no use of L, E and G but every operation is performed in the array A.

curLargest = None

count = 0

for (i = 0, largest = 0, i < n, count < k)

if (A[i] > largest) and (A[i] < curLargest)

largest = A[I]

if i == n – 1

curLargest = largest

Print(largest)

Q5. [5] **Weighted Median**

In finding the weighted median element, what does the weighted median algorithm return if the weights of all the elements are equal?

If the weights of all elements are equal, then the weighted median algorithm will return the average of all the elements in the given array. Thus, it returns the average of the given numbers

**The weighted median algorithm returns the *median* element in this case**

Q6. [10] **Mode**

Given an array A of *n* numbers in the range from 1 to *n*, write an algorithm that runs in O(*n*) time for finding the ***mode***, i.e. the number that occurs most frequently in A.

getMode(A,n)

for (i = array.length) do

count[A[i]] += 1

maxCount = 0

maxNum = 0

for i = 1 to array.length() do

if count[i] > maxCount then

maxCount = count[i]

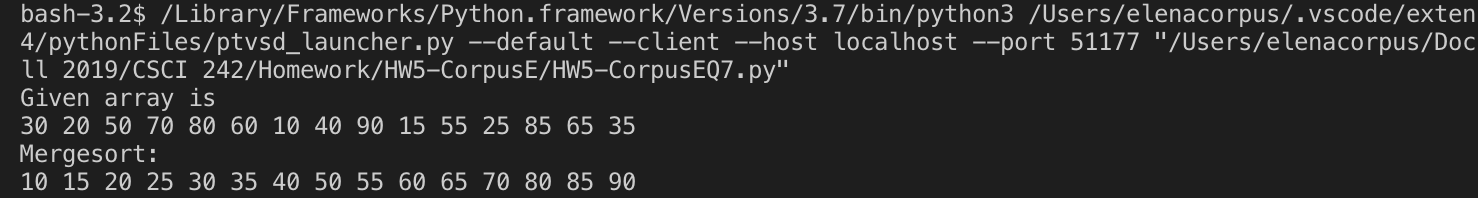
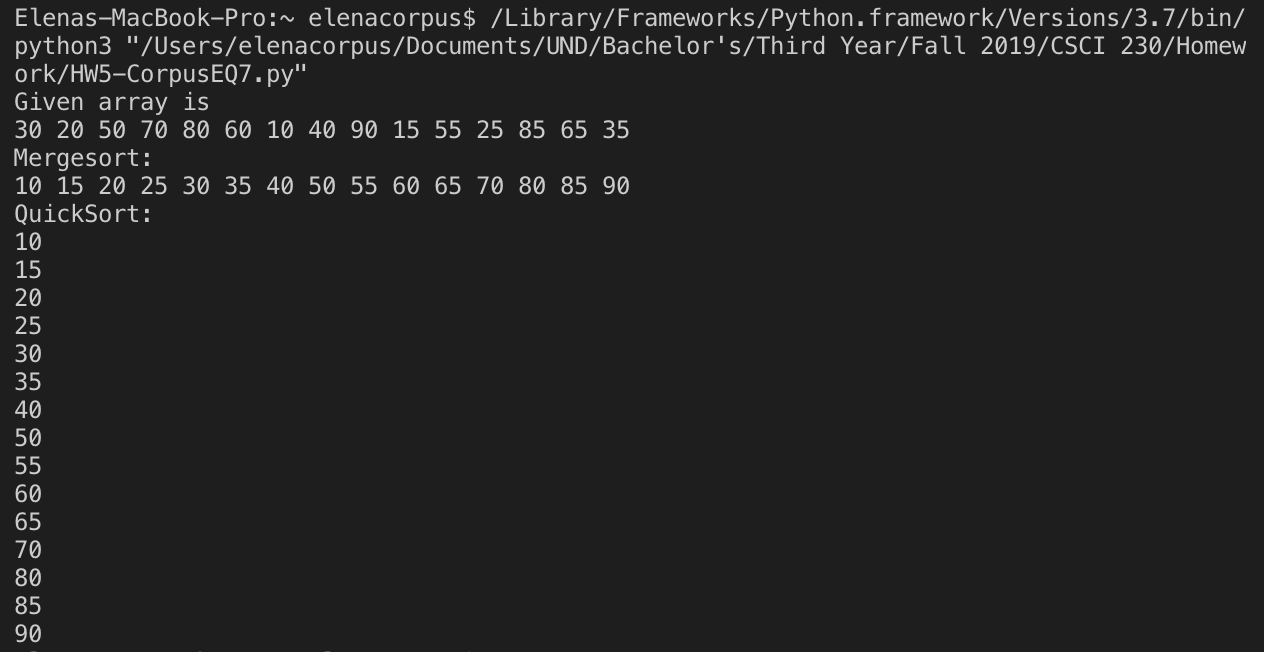
maxNum = i

Return maxNum

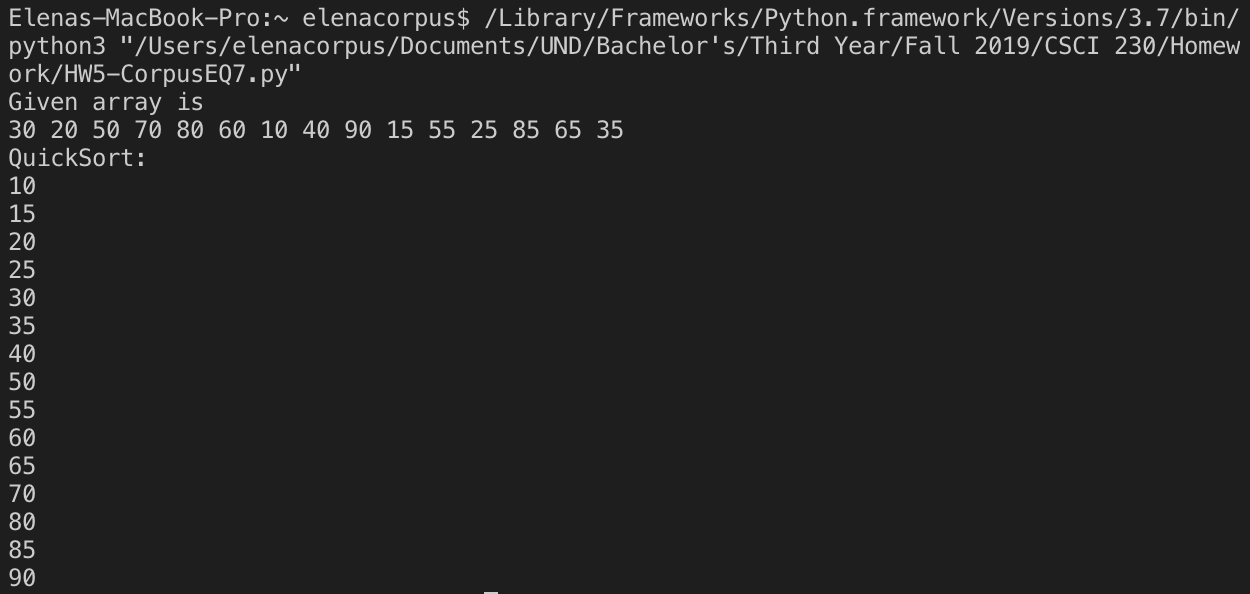
**Q7 – Q10: Implementation in Python/Java**

In the array A[30, 20, 50, 70, 80, 60, 10, 40, 90, 15, 55, 25, 85, 65, 35]

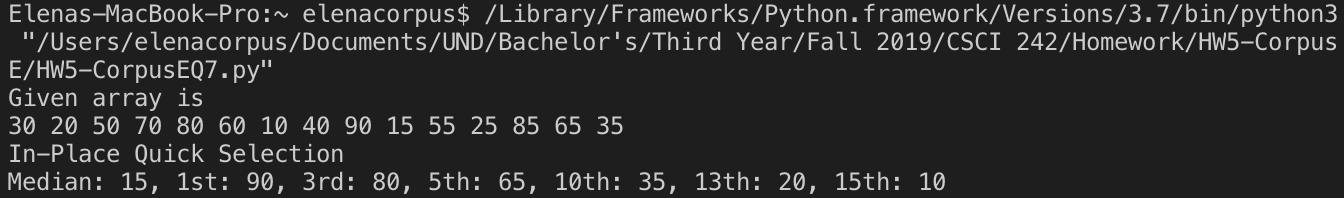
Q7. [25] Sort the array A in the ascending order by Merge Sort and print the final array A.

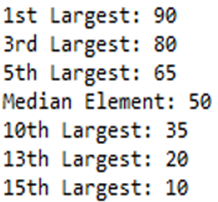


Q8. [25] Sort the array A in the ascending order by Quick Sort and print the final array A.



Q9. [19] Find the *k*th largest element in the array A by an ***In-Place Quick Selection*** and print the median element, the 1st, 3rd, 5th, 10th, 13th and the15th largest elements.





Q10. [20] Find the *k*th largest element in the array A by a ***deterministic Selection*** and print the 1st, 3rd, 5th, 10th, 13th, 15th largest elements.

