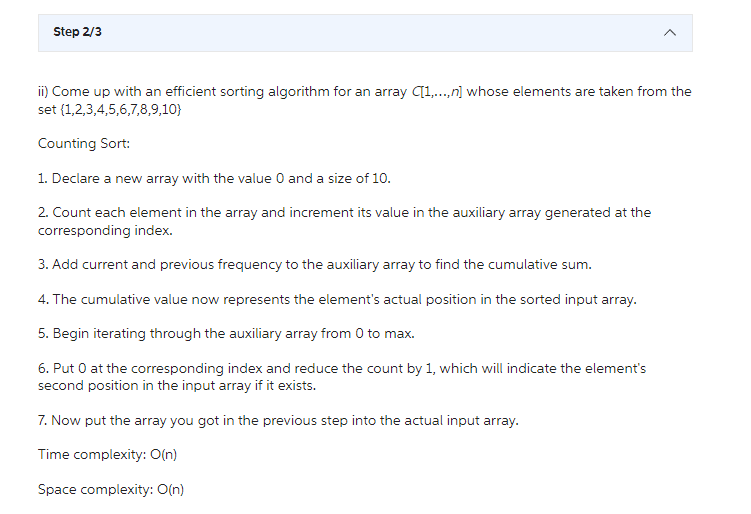
**Problem 1 Sorting Special Arrays (10 points)**

**Consider the problem of sorting an array A[1, ..., n] of integers. We presented an O(n log n)-time algorithm in class and, also, proved a lower bound of Ω(n log n) for any comparison-based algorithm.**

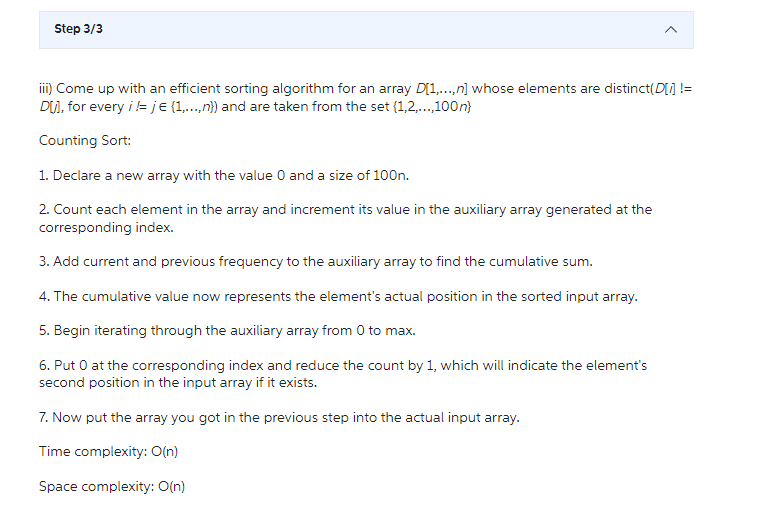
1. **Give an eﬀicient sorting algorithm for an array C[1,...,n] whose elements are taken from the set {1,2,3,4,5,6,7}.**
   1. The first step here is to define the size of a new array, which in this case is an array of size 7, given that the input are elements are taken from the set {1,2,3,4,5,6,7}
   2. Next, we iterate over the array, and check each element. For each element, we increment the count in our new array



1. **Give an eﬀicient sorting algorithm for an array D[1, ..., n] whose elements are distinct (D[i] ≠ D[j], for every i≠j ∈ {1, ..., n}) and are taken from the set {1, 2, ..., 2n}.**

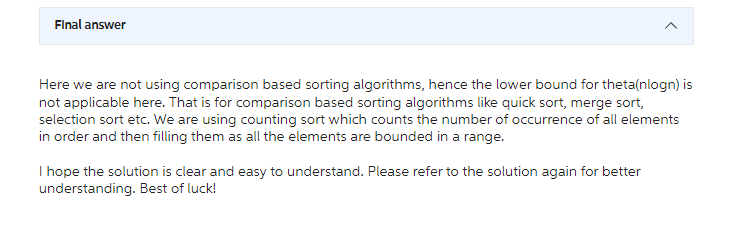
<https://www.chegg.com/homework-help/questions-and-answers/consider-problem-sorting-array-1--n-integers-presented-o-n-log-n-time-algorithm-class-also-q40321615>

counting Sort; on]
grep to define art [n] with Element [...]
slep-ai- define range Array as count on the
step-3;- for (i=0; i



**Problem 2 (10 points)**

In case you designed linear-time sorting algorithms for any subpart of problem 1, does it mean that the lower bound for sorting of Ω(n log n) is wrong? Explain.

-) Here we are not using
comparison sort&. 80 your answ01
is correct sent case for compari-
- son sort is ocnlogn).
Here we a

If you did not design a linear-time sorting algorithm for any subpart of problem 1, explain your lower bound for both subparts of problem 1.

**Problem 3 Closest Pair (10 points)**

We have learned the algorithm that solves the Closest pair problem in 2D in Θ(n log n) time. (Closest pair problem in 2D: Given n points in the 2D plane, find a pair with smallest Euclidean distance between them.)

Give an algorithm that solves the Closest pair problem in 3D in Θ(n log n) time. (Closest pair problem in 3D: Given n points in the 3D space, find a pair with smallest Euclidean distance between them.)

[**https://www.chegg.com/homework-help/questions-and-answers/give-algorithm-solves-closest-pair-problem-3d3d-theta-n-log-n-nlogn-time-closest-pair-prob-q98861485**](https://www.chegg.com/homework-help/questions-and-answers/give-algorithm-solves-closest-pair-problem-3d3d-theta-n-log-n-nlogn-time-closest-pair-prob-q98861485)

**Algorithm** For **3D-ClosestPair( )**

Given: n – number of points, p[1..n] – points array

**Data structures used by algorithm:**

d1[1..n] , d2[1..n], d3[1..n], d4[1..n], d5[1..n], d6[1..n] - distance arrays

sum[1..n] – sum array, index[1..n] – index array

***Step 1***.

a. Find point p1 such that its x coordinate is lower or equal to any other point in the array

of points p.

b. Find point p2 such that its x coordinate is higher or equal to any other point in the

array of points p.

c. Find point p3 such that its y coordinate is lower or equal to any other point in the array

of points p.

d. Find point p4 such that its y coordinate is higher or equal to any other point in the

array of points p.

e. Find point p5 such that its z coordinate is lower or equal to any other point in the array

of points p.

f. Find point p6 such that its z coordinate is higher or equal to any other point in the array

of points p.

***Step 2.***

a. Find distance of each point in p array from p1 and put its square in the d1 array.

For i=1..n, d1[i] = (distance between p[i] and p1)2

b. Find distance of each point in p array from p2 and put its square in the d2 array

For i=1..n, d2[i] = (distance between p[i] and p2)2

c. Find distance of each point in p array from p3 and put its square in the d3 array.

For i=1..n, d3[i] = (distance between p[i] and p3)2

d. Find distance of each point in p array from p4 and put its square in the d4 array.

For i=1..n, d4[i] = (distance between p[i] and p4)2

e. Find distance of each point in p array from p5 and put its square in the d5 array.

For i=1..n, d5[i] = (distance between p[i] and p5)2

f. Find distance of each point in p array from p6 and put its square in the d6 array.

For i=1..n, d6[i] = (distance between p[i] and p6)2

***Step 3***.

Calculate the sum array using the following formula:

For i=1..n,

sum[i] = 11\*d1[i] + 101\* d2[i] + 547\*d3[i] + 1009\*d4[i] + 5501\*d5[i] + 10007\*d6[i]

***Step 4***.

Initialise the index array to contain the indexes.

For i=1..n, index[i] = i

***Step 5.***

Mergesort the sum array. While mergesorting, if you exchange any 2 indices i and j of

sum array, be sure to exchange the corresponding entries i and j of index array.

***Step 6.***

For i=1..(n-1), Compare each point p[index[i]] to the 100 next points (if they exist)

ie. p[index[i+1]], p[index[i+2]]..p[index[i+100]]

if the 2 points being compared is the closest pair found so far, then store the 2 points.

***Step 7.*** Output the closest pair of points found.

***EXPLANATION :***

Assume p1, p2, p3, p4, p5, p6 are the extreme points found in step 1 of our 3D algorithm. Then the basic idea of our algorithm is that the closest pair of points should be almost equidistant from each of the 6 points.

So what our algorithm does is that it calculates the distance of each point from the 6 extreme points and puts its square in the corresponding d array. We wish to find (d1,d2,d3,d4,d5,d6) of a point x such that it almost equals (d1,d2,d3,d4,d5,d6) of a point y. The closer the match of the d’s, the closer the points are in 3D.

So what we do to find the closest match of (d1, d2, d3, d4, d5, d6) among all points in the d array, is that we multiply each by a prime number and add them to get the sum array. The closer the (d1, d2, d3, d4, d5, d6) of point x is to (d1, d2, d3, d4, d5, d6) of point y, the closer will be the sum numerically. Multiplying by prime numbers gives us a unique signature of each point in the sum array. Note that the prime numbers are all different from each other.

So then, we let the index array carry the index of the point corresponding to the sum array. We then mergesort the sum array, taking care to exchange corresponding entries in the index array when we exchange 2 elements of the sum array.

Now, we have the sorted sum array, and the points they represent are in the index array. Now, all we have to do is compare each point in the index array with 100 points that follow it. If the distance between 2 points being compared is the closest pair we have so far, it get stored. The closest pair of points is then output.