Part a:

a.) When n=4 and d=0, considering space as \_ , the output for above program will be -

n=4 begins

\_n=2 begins

\_\_n=1 begins

\_\_n=1 ends

\_\_hi

\_\_n=1 begins

\_\_n=1 ends

\_n=2 ends

\_hi

\_n=2 begins

\_\_n=1 begins

\_\_n=1 ends

\_\_hi

\_\_n=1 begins

\_\_n=1 ends

\_n=2 ends

n=4 ends

b.)

Recurrence Formula for T(n) is

T(n) = T(n/2) + T(n/2) + O(1)

T(n) = 2T(n/2) + O(1)

c.)

T(n) = T(n/2) + 1

n=2k where k=log n

T (2k) = T(2k/2) + 1

T(2k) = T(2k-1) + 1

= (T(2k-2) +1) + 1

= T(2k-2) + 2

= T(2k-3) + 3

T(2k) = T(2k-k) + k

= T(20) + k

= T(1) + k

If T(1) =1 then,

T(2k) = 1+k

T(n) = log n +1

T(n) = O(log n)

Thus, 2 \* log n steps or lines.

Part b:

a.) For the array of n elements the minimum number of bad pairs will be zero, when the array is sorted and the maximum number of bad pairs will be (n\*(n-1))/2, when the array is sorted in descending order and for every i,j in array a, a[i]>a[j]+2.

b.) The straightforward approach to find bad pairs is using two nested loops, and checking for every pair (i,j) when a[i] > a[j] + 2, then the pair is a bad pair. The time complexity for this approach is O(n^2).

Pseudocode:

bad\_pair=0;

for(i=0 to i<n)

for(j=0 to j<n)

if(a[i]>a[j]+2)

Bad\_pair++

End if

End for

End for

Here Bad\_pair is the counter for counting the bad pairs in each iteration.

c.) Since both the first half and second half of the array are sorted in ascending order, to find the total number of bad pairs

1. Divide the array into two halves by calculating the middle value, the middle will be the size of the array n divided by 2(mid=n/2).
2. Start iterating to the array, i points to zero and j points to mid+1, naming left and right subarrays respectively
3. Comparing the values of arr[i] and arr[j], if arr[i] > arr[j]+2 then there will be mid - i bad pairs since both halves of the array are sorted, so the elements left in left half of the array,arr[i+1],arr[i+2].... arr[mid] will be greater than arr[j]+2.

The time complexity of the above algorithm is O(n), as only one iteration is required to compare the left and right halves.

Pseudo code:

mid=n/2

i=0,j=mid+1;

bad\_pair=0;

while(i to mid and j to n)

if(arr[i]<arr[j])

i++

end if

else

if(arr[i]>arr[j]+2)

bad\_pair+=mid-i;

end if

j++

end else

end while

d.) The divide and conquer algorithm can be used for counting all the bad pairs, in divide and conquer technique a problem is divided to many subproblems,when the results from the subproblem are ready, then the results are combined to reach the main problem.

Algorithm:

1. Repeatedly divide the array into two halves until the size of the array is one.

2. Now a function is created that will count the number of bad pairs when two halves of the array are merged to single array, here merging the array is important so the we can compare and count the bad pairs for other halves of the array as well, and then comparing the values of arr[i] and arr[j], if arr[i] > arr[j]+2 then there will be mid - i bad pairs since both halves of the array are sorted, so the elements that are left in the left half of the array, arr[i+1],arr[i+2].... arr[mid] will be greater than arr[j]+2.

3. A recursive function is created to divide the array into two parts, and sum up the total number of bad pairs in first halves of the array and the second halves of the array and the bad pairs calculated by merging the array into one.

4. The base case will be when the size of the array is one.

Pseudo code:

merge(arr,left,mid,right)

{

i=left,j=mid+1;

bad\_pair=0;

array temp[right-left+1]

k=0

while(i to mid and j to n)

if(arr[i]<arr[j])

temp[k++]=arr[i]

i++

end if

else

if(arr[i]>arr[j]+2)

bad\_pair+=mid-i;

end if

temp[k++]=arr[j]

j++

end else

end while

for(i=left to left+right)

arr[i]=temp[i-left]

end for

}

getBadPair(arr[],i,j){

if(i>j)

return;

end if

mid=(i+j)/2

getBadPair(arr,i,mid);

getBadPair(arr,mid+1,j);

merge(arr,i,mid,j);

}

e. Alice’s argument is true as the time complexity of merge sort algorithm is O( n log n) and counting the bad pairs using the approach of divide and conquer will also take at least O(n logn) time, since the algorithm is using the very of similar approach as merge sort of dividing the array into two halves till the size of the array is one and then merging the two arrays and counting the bad pairs for each comparison. After merging the arrays on each recursive call, we will get the fully sorted array.

f.

T(n) = T(n/2) + T(n/2) + n

n/2 - for recursive function and n for merging it.

T(n) = 2 T(n/2) + n — 1

Substituting the value of n in equation 1

T(n/2) = 2T(n/4) + (n/2) — 2

Substituting the value of equation 2 in equation 1

T(n) = 2{2T (n/4) + n/2} + n

T(n) = 22 T(n/22) + 2n — 3

Substitute n/4 in place of n in equation 1

T(n/4) = 2T (n/8) + n/4 — 4

T(n) = 22 {2T (n/8) + n/4} + 2n

T(n) = 23 T(n/23) + 3n

T(n) = 24 T(n/24) + 4n

T(n) = 2 T (n/2i + in) T(1) = 1, let n/2i = 1, n= 2i

Log n = log2 x = i logx 2

log n = i => T(n) = nT(1) + n log2 n

T(n) = n + n log2 n

= O( n log2 n)