EVEN2022- APS Theory
T1 Solution & Marking Scheme

1.a.

The algorithm suggests that $T(n) = T(i) + T(n-1-i) + 1$. By summing this relationship for all the possible random values $i = 0, 1, \ldots, n-1$, we obtain that in average $nT(n) = 2(T(0)+T(1)+\ldots+T(n-2)+T(n-1))+n$. Because $(n-1)T(n-1) = 2(T(0) + \ldots + T(n-2)) + (n-1)$, the basic recurrence is as follows: $nT(n) - (n-1)T(n-1) = 2T(n-1) + 1$, or $nT(n) = (n+1)T(n-1) + 1$, or $\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{1}{n(n+1)}$. The telescoping results in the following system of expressions:

$$\begin{array}{cccccc} \frac{T(n)}{n+1} & = & \frac{T(n-1)}{n} & + & \frac{1}{n(n+1)} \\ \frac{T(n-1)}{n} & = & \frac{T(n-2)}{n-1} & + & \frac{1}{(n-1)n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \frac{T(2)}{3} & = & \frac{T(1)}{2} & + & \frac{1}{2\cdot 3} \\ \frac{T(1)}{2} & = & \frac{T(0)}{1} & + & \frac{1}{1\cdot 2} \end{array}$$

Because $T(0) = 0$, the explicit expression for $T(n)$ is:

$$\frac{T(n)}{n+1} = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \cdots \frac{1}{(n-1)n} + \frac{1}{n(n+1)} = \frac{n}{n+1}$$

so that $T(n) = n$.

1.b.

The inner loop has linear complexity cn, but the next called method is of higher complexity $n \log n$. Because the outer loop is linear in n, the overall complexity of this piece of code is $n^2 \log n$.

MARKING: Award full marks to those who have shown the complete tree or substitutions else mark according to the attempt.

---

2 We divide the array into equal sub-arrays (left and right half). Solve the problem recursively for the two parts. Two cases happen: (i) In both the parts, there is no majority element: in this case it is easy to see that the original array does not have a majority element, (ii) In either of the two parts, the recursive call returns a majority element: let these be x and y (in case only one the recursive calls returns a majority element, then y will not be defined). Now compare x with all the elements of A to see if it appears more than half the time { if yes, x is the answer. Perform the same steps for y. If neither x nor y turn out to be a present more than n/2 times, then the original array cannot have a majority element. The complexity of algorithm comes to O(n log n) procedure GetMajorityElement(a[1...n])

Assumption Getfrequency is function f mentioned above

Input: Array a of objects
Output: Majority element of a

```
if n = 1:        return a[1]
k = floor( n /2 )
elemlsub = GetMajorityElement(a[1...k])
elemrsub = GetMajorityElement(a[k+1...n]
if elemlsub = elemrsub:
        return elemlsub
lcount = GetFrequency(a[1...n],elemlsub)
rcount = GetFrequency(a[1...n],elemrsub)
if lcount > k+1:
        return elemlsub
else if rcount > k+1:
        return elemrsub
else
        return NO-MAJORITY-ELEMENT
```

MARKING: It has to be solved using D&C based algorithm. You can refer to the above-mentioned solution for marking. If someone proposed any sorting-based solution award 2 marks.

3.a. Divide: O(n) • Combine: O(n log n) because we sort by y • However, we can: – Sort all points by y at the beginning – Divide preserves the y-order of points Then combine takes only O(n) • We get T(n)=2T(n/2)+O(n), so T(n)=O(n log n)

3.b. The same idea can be used as for the 2D problem. Sort the points by x, y, and z, giving three arrays X, Y, Z. Partition the points in X into two sets X_L and X_R based on their sorted x values. Recursively find the closest pair in each of X_L and X_R, and let ä be the closest distance of pairs either both in X_L or both in X_R. To deal with the pairs of points where one is in X_L and one is in X_R, we need a condition similar to what he had for the closest distance in 2D problem. In the 3D problem, we don't have a middle strip of width 2ä but rather we have a middle slab of width 2ä where the slab extends in directions y and z. But the same idea can be used as before. For each point p in the slab, there can be at most some bounded number of points in its neighborhood, since those points in the slab that are in X_L (or X_R) must be spaced at least ä apart. For each point p, **15 such points need to be checked.** Then finding the closest pair with one point in X_L and the other in X_R can be done in O(n) time. This algorithm gives the same recurrence for time complexity, and hence the 3D problem can be solved in O(n log n) time too.

MARKING: It is a very straightforward question if in part (b) student has not found the correct number of points then deduct 1 mark for the same. Rest you can refer to the above-mentioned solution.

==

4.

T[].dist	1	2	3	4	5	6	7
initialize	0	∞	∞	∞	∞	∞	∞
V = 1	0	20	∞	∞	∞	10	25
V = 6	0	20	30	35	40	10	25
V = 2	0	20	30	35	40	10	23
V = 7	0	20	25	35	40	10	23
V = 3	0	20	25	28	40	10	23
V = 4	0	20	25	28	32	10	23
T[].path (final value)	0	1	7	3	4	1	2

	Shortest Path from 1	Cost
1	1	0
2	1 → 2	20 = 20
3	1 → 2 → 7 → 3	20 + 3 + 2 = 25
4	1 → 2 → 7 → 3 → 4	20 + 3 + 2 + 3 = 28
5	1 → 2 → 7 → 3→ 4 → 5	20 + 3 + 2 + 3 + 4 = 32
6	1→ 6	10 = 10
7	1 → 2 → 7	20 + 3 = 23

	T[w].dist	T[v].dist + C[v][w]	T[v].dist + C[v][w] < T[w].dist ?
V = 6	W:3 = ∞ W:4 = ∞ W:5 = ∞	10 + 20 = 30 10 + 25 = 35 10 + 30 = 40	True True True
V =2	W:3 = 30 W:7 = 25	20 + 15 = 35 20 + 3 = 23	False True
V = 7	W:3 = 30	23 + 2 = 25	True
V = 3	W:4 = 35	25 + 3 = 28	True
V = 4	W:5 = 40	28 + 4 = 32	True

MARKING: Full marks will be awarded o the student who has shown all the steps for the following graph. Else deduct marks and award according to the attempt

==