1.

By the previous homework, we all know if all edge costs are positive and distinct then MST T is unique. Now Suppose T is MST with normal cost, T’ is MST with square cost. Since all edges cost are distinct, so all edge cost of new graph are also distinct.

Assume there is an edge e in T’ but not in T. If add that edge to T, that will form a cycle, absolutely there is a shortest path in that cycle to form a MST, that path is different between T and T’.

This is contradiction Because we all know any shortest path is the same in T, T’ from the previous hw.

So, T and T’ must be the same.

2.

|E| − |V | = 20, so we can conclude there are many cycles in the graph G.

We need to remove all cycles from G to form a MST. Create a hash table for the edge weight. Run BFS to find cycle, remove the highest weight edge (use hash table to look up the edge O(1)). Since there are at most 21 cycles in G (|E| − |V | = 20 and need more than |V | - 1 edges to form a cycle), repeat BFS exactly 21 times (not affect to time complexity), then we have a tree, this is MST we need. Time complex O(|V|).

3.

The best solution for this problem is Kadane algorithm, O(n) time. But to meet O(nlogn) time requirement, we can use divide and conquer. The main point here is to find maximum value of sub-array, then merge them to form a longest array → maximum sum.

Max(1..n) = max(1..n/2)+max(n/2+1..n) = max(1..n/4)+max(n/4+1..n/2)+max(n/2+1..3n/2+1)+max(3n/2+2..n) = …...

Calculate recursively like that until the single number, then we have max(1..n) is the largest sum of array. Time complexity: T(n) = 2T(n/2)+Cn => O(nlogn)

4.

We can use binary search to solve this problem. To find a median number at nth, if number nth in A < nth in B then obviously that number is in the right from nth of A and in the left from nth of B (and vice versa). Compare to the “half” number and continue to search, we will have medium number.

Time complexity for binary search is O(logn)

5.

Divide:

Split X(1..n) into X1(1..n/2) and X2(n/2+1..n), continue to do the same with X1, X2 as well.

Call Left array is leftarray, right array is rightarray, right is right index of array, left is left index of array, mid is in the middle of left and right.

Conquer:

Recursively calculate the majority in the leftarray and rightarray.

mid = left+ (right-left)/2

majorleft =Major(leftarray, left, mid)

majorright =Major(rightarray, mid+1, right)

Merge:

frequency function will be calculated by counting number of occurrences.

If majorleft == majorright then

return majorleft, exit

if frequency(array, left, right, majorleft) > frequency(array, left, right, majorright)

return frequency(array, left, right, majorleft), exit

else

return frequency(array, left, right, majorright), exit

Time complexity of divide: O(1)

conquer: 2T(n/2)

merge: O(n)+O(n)

Overall: T(n)=2T(n/2) + O(n)

=> O(nlogn)

6.

Since the table size is 2kx2k so that is possible to split table into 4 equal quadrants until get 2x2 size. Now, consider 4 equal quadrants, L-shaped can fill 3 squares into 2x2 size then remain an missing square, then place the middle square of another L-shaped in the previous missing square if possible. Recursive like that will remain only 1 missing square.

Time complexity: T(2k) = 4T(2k/2) + C

=> O(22k)

7.

polynomials with degree-bound n/2

A1(x) = a0 + a2x + a4x2 + ...+ an-2xn/2-1

A2(x) = a1 + a3x + a5x2 + ...+ an-1xn/2-1

Then we can recursively compute polynomials A0(x), A1(x)

merge results:

A(x) = A0(x2) + xA1(x2)

time complexity O(nlogn)