1. Pseudo Code

   My pseudo code is the same as the professor's pseudo code, I just followed what the lecture told and tried to implement it.

```
struct node
{
  int level;              // the node's level in the tree
  ordered_set path;
  number bound;
};

 void travel2 (int n,
               const number W[] [],
               ordered-set& opttour,
               number& minlength)
 {
   priority_queue_of_node PQ;
   node u, v;

   initialize(PQ);                         // Initialize PQ to be empty.
   v.level = 0;
   v.path = [1];                           // Make first vertex the
   v.bound = bound(v);                     // starting one.
   minlength = ∞;
   insert(PQ, v);
   while (! empty(PQ)){
     remove(PQ, v);                        // Remove node with best bound.
     if (v.bound < minlength){
       u.level = v.level + 1;              // Set u to a child of v.
       for (all i such that 2 ≤ i ≤ n && i is not in v.path){
         u.path = v.path;
         put i at the end of u.path;
         if (u.level == n - 2){            // Check if next vertex
           put index of only vertex        // completes a tour.
           not in u.path at the end of u.path;
           put 1 at the end of u.path;     // Make first vertex last one.
           if (length(u) < minlength){     // Function length computes the
               minlength = length(u);      // length of the tour.
               opttour = u.path;
           }
         }
         else{
         u.bound = bound(u);
         if (u.bound < minlength)
             insert(PQ, u);
        }
    }
```

2. Flow chart
   (1) Initialize an array for the tour, with all of the data being 0;
   (2) Initialize a visited array to store the visited, array[i] 's value will be the position in the tour.
   (3) Calculate the bound for the root
   (4) Push the root into the priority queue
   (5) Starting from the priority queue, visit all nodes using the adjacency matrix
   (6) Calculate the bound for each node we visit
   (7) Starting with the node with the smallest bound, we go deeper and repetitively calculate the bound.
   (8) Update the Min_path_cost when hitting a leaf
   (9) If another branch has higher bound than current Min_path_cost, we prune it
   (10) Repeat the process until obtaining the best tour.

3. Time complexity analysis

   Suppose we have N cities left unvisited (including the root), we have to create all possible extensions for the unvisited cites which are N-1 (excluding the root). Following this thought, the complexity for generating the permutation is

$$O((n-1)!),$$

   which is equal to

$$O(2^{(n-1)})$$

   However, for each node, we have to calculate the cost, following the method below.

```
int calcbound(node n){
    int canvisit[31];
    for(int i = 0; i<31; i++)
        canvisit[i]=1;
    vector<int>::iterator end = n.path.end();
    for(vector<int>::iterator it = n.path.begin(); it != end; ++it){
        canvisit[*it-1] = 0;
    }
    canvisit[nd.label-1] = 1;
    int min = INF;
    int bound = 0;
    for(int i = 0; i<N; i++){
        min = INF
        if(canvisit[i]==0)continue;
        for(int j = 0; j<N; j++){
            if((canvisit[j]==0 || j ==nd.label-1) &&j!=1 && nd.label != 1){}
            else if(w[i][j]<min)min = w[i][j];
        }
        if(min!=INF){
            bound += min;
        }
    }
    return bound;
}
```

   Since there is a double for loop inside the function, which iterates through the N cities. This generates a time complexity of

$$O(n^2)$$

   So, combining the above, the total time complexity should be

$$O(n^2 \times 2^n)$$