①

We devised a recursive algorithm. In each recursion, we take the current value modulo 2 to get a bit. Then we divide the current number by two and round to get a new number. We recursively call our algorithm on this new number, prefixing it with the result, and concatenating it with the bits we got as the return value. Regardless of the parity of the numbers we operate on, after we get the last bit, the remaining part is divisible by 2, which means that the binary representation of the remaining part will end with 0. We shift all the bits by one bit, so that the weight of each effective bit becomes half of the original, and because the last bit has been obtained by us, the overall information will not be lost, so We show the correctness of the algorithm.

```
void ConvertNumberToBinary(int number){
    int binary[32];
    int i = 0;
    while(number > 0){
        binary[i] = number % 2;
        number = number / 2;
        i++;
    }
    for(int j = i - 1; j >= 0; j--){
        printf("%d", binary[j]);
    }
}
```

We can use 1 bit to represent 1, 2 bits to represent 2, 3, 3 bits to represent 4, 5, 6, 7, and so on. Consider a sequence S(t) = 1 × 1 + 2 × 2 + 3 × 4 + ... + t × 2^{t-1} . We can calculate $S(t) = (t-1)2^t + 1$ by using 2S(t) − S(t) = S(t). We take t = logn, and we can get the cost of conversion when converting all n numbers into binary representation. This cost can be limited to O((logn − 1)n + 1) = O(nlogn).

②

m ≤ 3n − 6

directly by using the Euler inequality of the planar graph. We assume that the degree of all points in the graph is greater than or equal to 6, then we have

$$\sum_i d_i \geq 6n$$

According to the handshake theorem, we have

$$\sum_i d_i = 2m$$

Then we get that the number of all edges in the graph satisfies the inequality
m ≥ 3n

We found that there is no m that can satisfy the two inequalities we obtained at the same time, which means that our assumption cannot be satisfied. That is to say, for any planar graph, there must be nodes with degree less than or equal to five.

We use mathematical induction to prove this conclusion. We first prove the base case. When there is only one node in the whole graph, we randomly select a color to dye, then the whole graph can be 6-colored. Our inductive hypothesis is that for planar graphs with n or less nodes, we can find a 6-colored solution. Next we prove the inductive step. For a planar graph with n+1 nodes, we can find a node with degree less than or equal to five, and we use v_i to represent the entire node. Note that this planar graph can be 6-colored equivalent to the original graph minus v_i and can be 6-colored, because no matter what color the neighbors of v_i are colored, because it only has 5 neighbors, so we still have a way of coloring v_i that doesn't destroy the 6-colored nature of the graph. According to our inductive hypothesis, the original graph can be 6-colored after removing v_i, which means we have proved the inductive step. So we proved that we must be able to make a floor plan 6-colored.

③

```c
typedef struct node {
    bool message_sent;
    int message;
    int neighbor_count;
    node *neighbors[10];
} node;

void flooding(node *n, int message) {
    if (n->message_sent == 1) {
        return;
    }
    n->message_sent = 1;
    n->message = message;
    for (int i = 0; i < n->neighbor_count; i++) {
        flooding(n->neighbors[i], message);
    }
}
```

When a node sends a message, it will mark that it has sent the message. In the following process, this node will not send any more messages. In each round, we will have message transmission, so after one iteration, the number of nodes that can continue to send messages will decrease, and the number of nodes that can send messages in the entire graph is limited, so our algorithm will stop after a certain iteration. Because each node forwards the original message, the message does not change during the whole process, so the messages received by all nodes are the messages of the original node.

Note that each node will only send a message once, so when a node sends a message, the more messages it sends, the more messages the algorithm will generate in total. The structure with the largest node degree in the network is a complete graph. Consider a complete graph with n nodes. In the first round, nodes send information to other n-1 nodes, and in the second round, other n-1 nodes also send information to all their neighbors, and then the algorithm terminates. In this case, the total number of messages sent is $N = n \times (n - 1)$. This is the upper bound for the algorithm to send information.

④

We first pass the information of the node to the root node. After an internal node receives the messages from all the child nodes, it passes the largest of these messages to its parent node, and continues this process until it reaches the root node. Then the root node starts to pass down the maximum value it gets in the way of Flooding, so that all nodes in the entire tree will get this maximum value. Because the value of the global largest leaf node will not be dropped during the transfer process, this value can reach the root node, which means that our algorithm is correct.

```
void messagePass(node *root, int *message) {
    if (root->isLeaf) {
        *message = root->data;
        send(root->parent, message);
    }
    int leftMessage, rightMessage;
    messagePass(root->left, &leftMessage);
    messagePass(root->right, &rightMessage);
    *message = max(leftMessage, rightMessage);
    send(root->parent, message);
}
void flood(node *root) {
    send(root->left, root->data);
    send(root->right, root->data);
    flood(root->left);
    flood(root->right);
}
```

The number of rounds of the algorithm is twice the depth of the tree. For a complete binary tree with n nodes, the depth of the tree is [logn]. The number of rounds our algorithm performs is 2[logn].

Each edge will carry a message in the process of passing up and down, which means that the total number of messages is twice the number of edges, is 2m.

When we use binary to represent integers, larger numbers need to use more bits for storage. For a number of t digits, we need logt bits for storage. So the total number of bits required in the algorithm is $2m\log x_{vmax}$.

⑤

Our algorithm is correct on any rooted tree, and the number of messages passed and the number of bits are the same. But if our tree is a chain, it means that our tree has only two leaf nodes, and all other nodes have a degree of 2. In this case, the number of rounds of the algorithm will be much more, we need n/2 round passes the integers of the two leaf nodes to the middle, and then spend n/2 to broadcast the larger of the two numbers to the entire tree. The total number of rounds spent is n.

Proof by contradiction. We use $w_1, w_2, ..., w_m$ to denote the weights of the upper edges of the graph. Not generally we can assume $w_1 < w_2 < ... < w_m$. We assume that there are two different minimum spanning trees, then there must be at least one edge with different weights between the two minimum spanning trees, assuming w_i , w_j . Before this, the edges of the minimum spanning trees of the two graphs are the same, and in this step we selected the edge with weight wi in the first minimum spanning tree, which means that the weight of w_i Less than the weight of the other remaining unselected edges, of course including w_j , we get $w_i < w_j$. Symmetrically, we can also get $w_j < w_i$, which leads to a contradiction.

⑥

If Robot R knows the specific value of p, because of the principle of the shortest straight line between two points, R directly uses the maximum speed to go to p, and then uses the maximum speed to go to 1. The time spent on the first step is $\sqrt{(\frac{1}{2}-p)^2+(\frac{1}{2})^2}$ and the time spent on the second part is 1 − p, the total time spent by the entire algorithm is $\sqrt{(\frac{1}{2}-p)^2+(\frac{1}{2})^2}+1-p$

Strategy-1 takes a constant time to send the treasure to the end. Robot R first goes to the origin and then directly to the end. In the process of R going to the end point, if you encounter r, take the treasure with you. The total time spent by this strategy is $\frac{\sqrt{2}}{2}+1$. We can calculate the competitive ratio $c_r=\frac{\frac{\sqrt{2}}{2}+1}{\sqrt{(\frac{1}{2}-p)^2+(\frac{1}{2})^2}+1-p}$

In strategy-2, R first goes to the midpoint, and then looks for r forward. If p is less than a quarter, then R will find r at p, otherwise R will find r at one quarter. The overall strategy of this strategy is the time spent is min(2(1−p),3/2). We can calculate the competitive ratio $c_r=\frac{min(2(1-p),\frac{3}{2})}{\sqrt{(\frac{1}{2}-p)^2+(\frac{1}{2})^2}+1-p}$