UVA12676　Inverting Huffman

题目：

静态哈夫曼编码是一种主要用于文本压缩的编码算法。给定一个由 N 个不同字符组成的特定长度的文本，算法选择 N 个编码，每个不同的字符一个编码。使用这些编码压缩文本，当选择编码算法构建一个具有 N 个叶子的二叉树时，对于 N≥2，树的构建如下：

1. 对文本中的每个不同字符，构建一个仅包含单个结点的树，并为其分配权值，权值与文本中该字符出现的次数一致。
2. 构建一个包含上述 N 棵树的集合 s。
3. 当 s 包含多于一棵树时：
   （a） 选择最小的权值 t1∈s，并将其从 s 中删除；
   （b） 选择最小的权值 t2∈s，并将其从 s 中删除；
   （c） 构建一棵新树 t，t1 为其左子树，t2 为其右子树，并且 t 的权值为 t1、t2 权值之和；
   （d） 将 t 加入 s 集合。
4. 返回保留在 s 中的唯一一棵树。

对于文本“abracadabra”，由上述过程生成的树，可以像下图左侧，其中每个内部结点编辑有子树根的权值。请注意获得的树也可以像下图右侧或其它，因为在步骤 3（a）、3（b）中，结合可能包含几个权值最小的树。

对文本中的每个不同字符，其编码取决于最终树中从根到对应字符的叶子之间的路径，编码的长度是这条路径中的边数（与路径中的内部结点一致）。假设该算法构建的是左侧的树，“r”的代码长度为 3，“d”的代码长度为 4。

根据算法选择的 N 个代码的长度，找所有字符总数的最小值。

输入：

输入文件包含多个测试用例，每个测试用例如下所述：

第一行包含一个整数 N（2≤N≤50），表示文本中出现的不同字符数。

第二行包含 N 个整数 Li（1≤Li≤50，i=1,2,⋯,N）,表示由 huffman 算法生成的不同字符的编码长度。

假设至少存在一棵上述算法构建的树，可以生成具有给定长度的编码。

输出：

对每个测试用例，输出一行，表示所有字符总数的最小值。

题解：

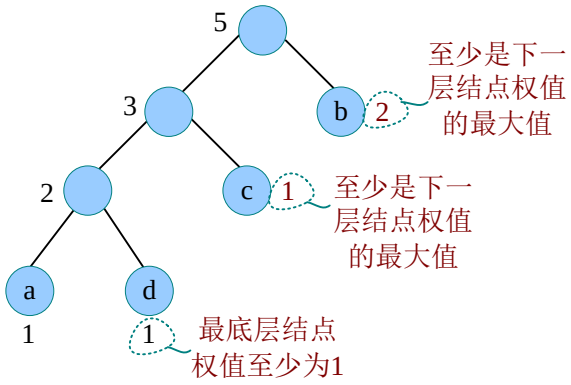
根据编码长度，推测最小字符数。

举例：

4

3 1 2 3

最长编码为 3，即最大深度为 3。



根据编码长度推测，该文本至少有个字符，其中 1 个 a，1 个 d，1 个 c，2 个 b，可能为“abbcd”。

算法设计：

（1） 每一层用一个深度数组 deep[]记录该层结点权值，该层每个结点的权值初始化为 0，等待推测权值；

（2） 根据输入的编码长度算出最大长度即最大深度 maxd。

（3） 从最大深度 maxd 向上计算并推测，直到树根。开始时 temp=1;

- i=maxd：第 i 层的结点权值如果为 0，则初始化为 temp(此时，temp=1)。对第 i 层从小到大排序，然后将第 i 层每两个合并，权值放入上一层（i-1 层）。更新 temp 为第 i 层排序后的最后一个元素（最大元素）。
- i=maxd-1：重复上述操作；
- i=0：结束。输出第 0 层第一个元素。

代码实现：

```
vector<long long>deep[maxn];
int main()
{
    int n,x;
    while(cin>>n)
    {
        for(int i=0;i<n;i++)
            deep[i].clear();
        int maxd=0;
        for(int i=0;i<n;i++)
        {
            cin>>x;
            deep[x].push_back(0);
            maxd=max(maxd,x);//求最大深度
        }
        long long temp=1;
        for(int i=maxd;i>0;i--)
        {
            for(int j=0;j<deep[i].size();j++)
                if(!deep[i][j])
                    deep[i][j]=temp;//将第 i 层最大的元素值赋值给 i-1 层没有权值的结点
            sort(deep[i].begin(),deep[i].end());//第 i 层排序
            for(int j=0;j<deep[i].size();j+=2)
```

```
                deep[i-1].push_back(deep[i][j]+deep[i][j+1]);//合并后放入上一层
                temp=*(deep[i].end()-1);//取第 i 层的最后一个元素，即第 i 层最大的元素
            }
            cout<<*deep[0].begin()<<endl;//输出树根的权值
        }
        return 0;
    }
```

特别注意：权值有可能很大，数组定义为 long long 类型，否则不通过。