Problème BinSearch

Fichier d'entrée	`stdin`
Fichier de sortie	`stdout`

```
bool binary_search(int n, int p[], int target){
    int left = 1, right = n;
    while(left < right){
        int mid = (left + right) / 2;
        if(p[mid] == target)
            return true;
        else if(p[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    if(p[left] == target) return true;
    else return false;
}
```

On sait bien que lorsque le tableau `p` est trié, le code ci-dessus retourne `true` si et seulement si `p` contient `target`. Par contre, ce n'est pas forcément le cas si `p` n'est pas trié.

On vous donne un entier positif n et une séquence $b_1, \ldots, b_n \in \{\texttt{true}, \texttt{false}\}$. On vous garantit qu'il existe un entier strictement positif k tel que $n = 2^k - 1$.

Votre objectif est de produire un tableau p qui soit une permutation de $\{1, \ldots, n\}$ et qui respecte certaines conditions. Soit $S(p)$ le nombre d'indices $i \in \{1, \ldots, n\}$ pour lesquels `binary_search(n, p, i)` ne retourne **pas** b_i. Vous devez choisir p pour que $S(p)$ soit petit (comme précisé dans la section "Contraintes").

(Remarque : une permutation de $\{1, \ldots, n\}$ est une séquence de n entiers qui contient chaque entier entre 1 et n *exactement* une fois.)

Données d'entrée

Chaque fichier d'entrée contient plusieurs tests. La première ligne de l'entrée contient T, le nombre de tests de ce fichier. Les tests sont fournis ensuite, les uns après les autres.

La première ligne d'un test contient l'entier n. La deuxième ligne d'un test contient une chaîne de longueur n, qui ne contient que des caractères '0' et '1'. Ces caractères ne sont pas séparés par des espaces. Si le i^{e} caractère est '1', alors $b_i = \texttt{true}$, et si c'est '0', alors $b_i = \texttt{false}$.

Données de sortie

La sortie consiste en la réponse pour chacun des T tests. La réponse pour un test particulier consiste en la permutation p produite pour ce test.

Contraintes

- Notons $\sum n$ la somme de toutes les valeurs de n pour une entrée donnée.
- $1 \leq \sum n \leq 100\,000$.
- $1 \leq T \leq 7\,000$.
- $n = 2^k - 1$ pour un certain $k \in \mathbb{N}$, $k > 0$.
- Si $S(p) \leq 1$ pour tous les tests d'une sous-tâche, alors on vous donne 100% des points pour cette sous-tâche.
- Sinon, si $0 \leq S(p) \leq \lceil \log_2 n \rceil$ (i.e. $1 \leq 2^{S(p)} \leq n + 1$) pour tous les tests d'une sous-tâche, alors on vous donne 50% des points pour cette sous-tâche.

#	Points	Contraintes
1	3	$b_i = \texttt{true}$.
2	4	$b_i = \texttt{false}$.
3	16	$1 \leq n \leq 7$.
4	25	$1 \leq n \leq 15$.
5	22	$n = 2^{16} - 1$ et chaque b_i est choisi aléatoirement, de manière indépendante et uniforme parmi $\{\texttt{true}, \texttt{false}\}$.
6	30	Pas de contraintes particulières.

Exemples

Fichier d'entrée	Fichier de sortie
4 3 111 7 1111111 3 000 7 000000000	1 2 3 1 2 3 4 5 6 7 3 2 1 7 6 5 4 3 2 1
2 3 010 7 0010110	3 2 1 7 3 1 5 2 4 6

Explications

Exemple 1. Dans les deux premiers tests du premier exemple, on a $S(p) = 0$.

Dans le troisième test, on a $S(p) = 1$. En effet, `binary_search(n, p, 2)` retourne `true`, alors que $b_2 = \texttt{false}$.

Dans le quatrième test, on a $S(p) = 1$. En effet, `binary_search(n, p, 4)` retourne `true`, alors que $b_4 = \texttt{false}$.

Exemple 2. On a $S(p) = 0$ pour les deux tests.