1 重點整理

1.1 注意事項

- 要二分搜的東西必須是具備**單調性**
- 單純的二分搜：$\log n$ 的測資範圍可以到 10^{18}，$n \log n$ 則可以到 5×10^5
- **範圍的初始左右界設定錯誤**
- 取 mid 的時候發生溢位
- 最後的 mid 不一定是答案
- 負數範圍上二分搜使用錯誤寫法

1.2 使用時機

- 尋找元素
- **二分搜答案**
- 最小化最大值/最大化最小值
- 第 k 大/小問題

1.3 二分搜工具

```
// 使用前請記得sort!!!
// 尋找是否有val，回傳true或false
cout << binary_search(v.begin(), v.end(), val) << endl;

// 尋找大於等於/大於val的第一個值，回傳此值的iterator
lower_bound(v.begin(), v.end(), val);
upper_bound(v.begin(), v.end(), val);
```

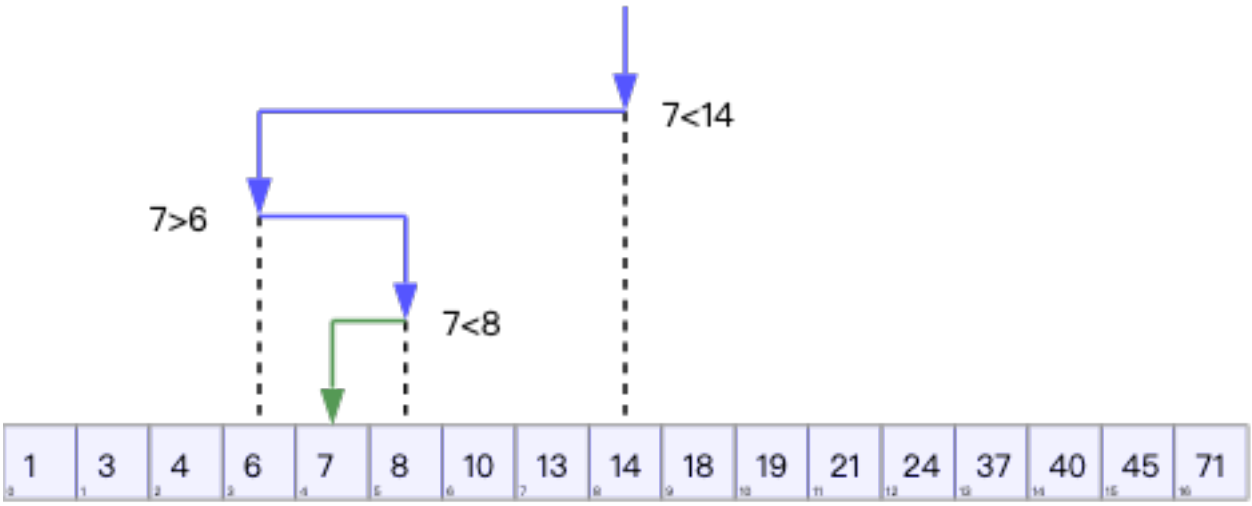
1.4 二分搜模板

```cpp
#include <bits/stdc++.h>
using namespace std;

bool check(int mid){
    // check function
}

int main(){
    // 請注意，這裡為[ll, rr);
    int ll=0, rr=n, mid;
    while (ll<rr){
        int mid=ll+(rr-ll)/2;
        if (check(mid)){
            rr=mid;
        }else{
            ll=mid+1;
        }
    }


    return 0;
}
```

2 二分搜簡介

2.1 概念

二分搜尋法（簡稱二分搜）的概念非常簡單，就如同遊戲「終極密碼」一樣，請參考以下範例



以上為最佳的遊玩策略，每次都選擇所有可能的數字的中位數
這就是二分搜的概念，此外，我們可以保證在此情況按照此策略可以在 4 次猜測以內找到答案，套用在程式的概念，就是可以保證時間複雜度是好的

總結：二分搜是一種搜尋技巧，可以在找到是否有一個值（以及他的位置）

2.2 複雜度分析

由於上面的概念很簡單，我們可以先做簡單的複雜度分析
由於每次都會減少一半的數值，直到只剩下一個可能的值，我們可以以此逆推

設元素的總數量為 n，每次都會減少一半的元素，可知每次查詢的時間複雜度為 $\log_2 n$ 可用 $O(\log n)$ 表示
可處理的上限的範圍可到 10^{18}，是個非常快速的搜尋法

3 0/1 二分搜

接下來是非常間單且經典的問題，會給予一個長度為 n 的陣列，並且陣列的前段皆為連續的 0，後段皆為連續的 1，目標是要尋找最左邊的 1

輸入：

```
7
0 0 0 1 1 1 1
```

輸出：

```
4
```

以上的例題實際上與前述的「終極密碼」非常相似，我們可以使用相同的規則，但是如果得到的值為 0，就當作數值太小，否則當作數值太大