二分查找

1. 一般而言，对于包含n个元素的列表，用二分查找最多需要$\log_2 n$步，而简单查找最多需要n步。
2. 缺陷:仅当列表是有序的时候，二分查找才管用。例如，电话簿中的名字是按字母顺序排列的，因此可以使用二分查找来查找名字。
3. C++代码实现如下:

```
//二分查找
#include <iostream>
using namespace std;
int binary_finding(double *arr,int size,double target_)
{
    int low = 0;
    int high = size - 1;
    int mid = (low + high) / 2;
    double guess = arr[mid];
    while (low <= high)
    {
        if (guess == target_)
        {
            cout << "找到了目标元素" << target_ << "该元素在原数组中对应的下标为" <<
mid << endl;
            return mid;
        }
        else if (guess > target_)
        {
            high = mid - 1;
        }
        else if (guess < mid)
        {
            low = mid + 1;
        }
        cout << "没有找到目标元素" << endl;
        return 0;
    }
}
int main()
{
    int n;
    cout << "请输入待查找的数组元素的个数" << endl;
    cin >> n;
    double* num = new double[n];
    cout << "请依次输入待查找数组的各元素信息(相邻两个元素的输入用空格或者换行符隔开)"
<< endl;
    for (int i = 0; i < n; i++)
    {
        cin >> *(num + i);
    }
```

```
    double target;
    cout << "请输入待查找的元素的值" << endl;
    cin >> target;
    binary_finding(num, n,target);


    delete[]num;
}
```

4. Python代码实现如下:

```python
def binary_search(list, item):
    low = 0
    high = len(list)-1
    while low<=high:
        mid = (low+high)//2
        guess = list[mid]
        if guess == item:
            return mid
        if guess > item:
            high = mid-1
        else:
            low = mid+1
    return None
my_list = [1,3,5,7,9]
print (binary_search(my_list,3))
print (binary_search(my_list,-1))
```

practices

1.1

Question:

假设有一个包含128个名字的有序列表，你要使用二分查找在其中查找一个名字，请 问最多需要几步才能找到?

Answer:

需要$\log_2 128 = 7$步.

1.2

Question:

上面列表的长度翻倍后，最多需要几步?

Answer:

8步.

上述列表长度翻倍128×2 = 256

$\log_2 256 = 8$ 故需要8步

5. 运行时间的计算:

> 回到前面的二分查找。使用它可节省多少时间呢？简单查找逐个地检查数字，如果列表包含100个数字，最多需要猜100次。如果列表包含40亿个数字，最多需要猜40亿次。换言之，最多需要猜测的次数与列表长度相同，这被称为线性时间（linear time）. 二分查找则不同。如果列表包含100个元素，最多要猜7次；如果列表包含40亿个数字，最多需猜32次。

comment:对于$\log_2 n$ 不为整数的情形,在计算运行次数时向上取整

e.g.$\log_2 100$不为整数，$\log_2 64 = 6$ < $\log_2 100$ < $\log_2 128 = 7$ 故我们认为对本列表的查找次数为7次.

大O表示法

一些常见的大O运行时间:

1.O(log n)，也叫对数时间，这样的算法包括二分查找。

2.O(n)，也叫线性时间，这样的算法包括简单查找。

3.O(n * log n)，这样的算法包括第4章将介绍的快速排序——一种速度较快的排序算法。

4.O(n^2)，这样的算法包括第2章将介绍的选择排序——一种速度较慢的排序算法。

5.O(n!)，这样的算法包括接下来将介绍的旅行商问题的解决方案——一种非常慢的算法。

comment:

1. 本书使用大O表示法（稍后介绍）讨论运行时间时，log指的都是$\log_2$。
2. 大O表示法指出了最糟情况下的运行时间:

> 假设你使用简单查找在电话簿中找人。你知道，简单查找的运行时间为O(n)，这意味着在最 糟情况下，必须查看电话簿中的每个条目。如果要查找的是Adit——电话簿中的第一个人，一次 就能找到，无需查看每个条目。考虑到一次就找到了Adit，请问这种算法的运行时间是O(n)还是 O(1)呢？ 简单查找的运行时间总是为O(n)。查找Adit时，一次就找到了，这是最佳的情形，但大O表 示法说的是最糟的情形。因此，你可以说，在最糟情况下，必须查看电话簿中的每个条目，对应 的运行时间为O(n)。这是一个保证——你知道简单查找的运行时间不可能超过O(n)。

3. 算法的速度指的并非时间，而是操作数的增速
4. 谈论算法的速度时，我们说的是随着输入的增加，其运行时间将以什么样的速度增加。
5. 算法的运行时间用大O表示法表示。
6. O(log n)比O(n)快，当需要搜索的元素越多时，前者比后者快得越多。

旅行商问题

> 有一位旅行商,他需要前往5个城市。这位旅行商（姑且称之为Opus吧）要前往这5个城市，同时要确保旅程最短。为此，可考虑前往这些城市的各种可能顺序。对于每种顺序，他都计算总旅程，再挑选出旅程最短的路线。5个城市有120种不同的排列方式。因此，在涉及5个城市时，解决这个问题需要执行120次操作。涉及6个城市时，需要执行720次操作（有720种不同的排列方式）。涉及7个城市时，需要执行5040次操作！ 推而广之，涉及n个城市时，需要执行n!（n的阶乘）次操作才能计算出结果。因此运行时间为O(n!)，即阶乘时间。除非涉及的城市数很少，否则需要执行非常多的操作。如果涉及的城市数超过100，根本就不能在合理的时间内计算出结果——等你计算出结果，太阳都没了。 这种算法很糟糕！Opus应使用别的算法，可他别无选择。这是计算机科学领域待解的问题之一。对于这个问题，目前还没有找到更快的算法，有些很聪明的人认为这个问题根本就没有更巧妙的算法。面对这个问题，我们能做的只是去找出近似答案。

comment:

1. 二分查找的速度比简单查找快得多。
2. O(log n)比O(n)快。需要搜索的元素越多，前者比后者就快得越多.
3. 算法运行时间并不以秒为单位。
4. 算法运行时间是从其增速的角度度量的。
5. 算法运行时间用大O表示法表示。