Ordered Lists

DEFINITION An ***ordered list*** is a list in which each entry contains a key, such that the keys are in order. That is, if entry i comes before entry j in the list, then the key of entry i is less than or equal to the key of entry j.

- All List operations except insert and replace apply without modification to an ordered list.(除了insert和replace函数以外，其他的线性表操作对于有序表来说不需要修改。）
- Methods insert and replace must fail when they would otherwise disturb the order of a list.（当企图打乱有序表的有序性时，insert和replace操作会失败）

Ordered Lists

- We implement an ordered list as a class *derived* from a contiguous List. In this derived class, we shall override the methods insert and replace with new implementations.（将**ordered list**类定义成一个顺序表类**contiguous List**类的继承类，为此继承类重写其插入和替换操作）。

```
class Ordered_ list: public List<Record>{
public:
Ordered_list( );
Error_code insert(const Record &data);
Error_code insert(int position, const Record &data);
Error_code replace(int position, const Record &data);
};
```

Overloaded Insertion

（重载插入函数）

- We also overload the method insert so that it can be used with a single parameter. The position is automatically determined so as to keep the resulting list ordered:

Error_code Ordered_ list **::** insert(**const** Record &data)

> ***/* Post:*** If the Ordered list is not full, the function succeeds: The Record data is inserted into the list, following the last entry of the list with a strictly lesser key (or in the first list position if no list element has a lesser key).
>
> Else: the function fails with the diagnostic Error_ code overflow. **/*

Overloaded Insertion

```
{
int s = size( );
int position;
for (position = 0; position < s; position++) {
        Record list_data;
        retrieve(position, list_data);
        if (data >= list_data) break;
}
return List<Record> :: insert(position, data);
}
```

- The scope resolution is necessary, because we have overridden the original List insert with a new Ordered_ list method。

Overridden Insertion

（过载插入函数）

Error_code Ordered_ list :: insert(int position, const Record &data)

/* Post: If the Ordered list is not full, 0<= position<= n , where n is the number of entries in the list, and the Record data can be inserted at position in the list, without disturbing the list order, then the function succeeds: Any entry formerly in position and all later entries have their position numbers increased by 1 and data is inserted at position of the List . Else: the function fails with a diagnostic Error_code . */

Overridden Insertion

```
{
        Record list_data;
        if (position > 0) {
                retrieve(position - 1, list_data);
                if (data < list_data)
                return fail;
                }
        if (position < size( )) {
                retrieve(position, list_data);
                if (data > list_data)
                return fail;
        }
  return List<Record> :: insert(position, data);
}
```

Overridden Insertion

Note: The overridden method replaces a method of the base class by one with a matching name and parameter list. The overloaded method matches in name but has a different parameter list.

（注：**overridden method**:子类中重写了父类中的某个函数

overloaded method:与某个函数名相同，但参数不同）

Binary Search（二分查找）

- Idea: In searching an ordered list, first compare the target to the key in the center of the list. If it is smaller, restrict the search to the left half; otherwise restrict the search to the right half, and repeat. In this way, at each step we reduce the length of the list to be searched by half.（在查找时，将目标关键字与记录表的中间值去比较，如它小，则在左半边继续用同样的方法找，否则在右半边找，这样，每次待查表的长度都减少了一半。

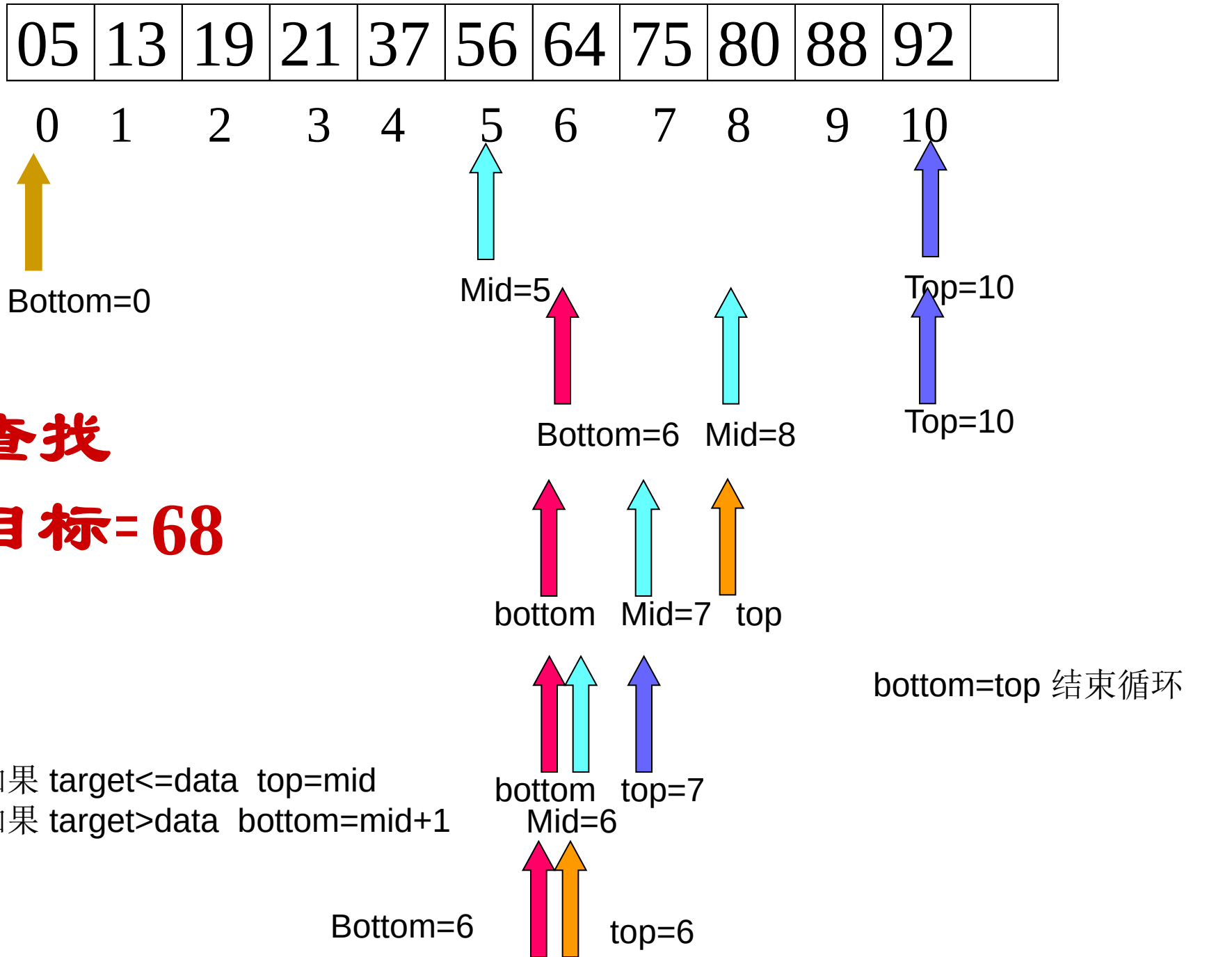
- Keep two indices, top and bottom, that will bracket the part of the list still to be searched.（用top和bottom来界定待查记录的范围）

> The target key, provided it is present, will be found between the indices bottom and top, inclusive.

Binary Search
（二分查找）

- Initialization: bottom = 0; top = the_list.size( ) - 1;
- Compare target with the Record at the midpoint,

  mid = (bottom+top)/2;
- Change the appropriate index top or bottom to restrict the search to the appropriate half of the list.
- Loop terminates when top<= bottom, if it has not terminated earlier by finding the target.
- Make progress toward termination by ensuring that the number of items remaining to be searched, top - bottom + 1,strictly decreases at each iteration of the process.


05 13 19 21 37 56 64 75 80 88 92
0 1 2 3 4 5 6 7 8 9 10
Bottom=0
Mid=5
Top=10
Bottom=6
Mid=8
Top=10
查找
目标=68
bottom
Mid=7
top
bottom=top 结束循环
如果 target<=data top=mid
如果 target>data bottom=mid+1
bottom
top=7
Mid=6
Bottom=6
top=6

Nonrecursive Binary Search（非递归查找）

```
Error_code binary_search_1 (const Ordered_list &the list,
const Key &target, int &position)
/* Post: If a Record in the list has Key equal to target , then
position locates one such entry and a code of success is returned.
Otherwise, not_present is returned and position is undefined.
Uses: Methods for classes List and Record . */
{
    Record data;
    int bottom = 0, top = the list.size( ) - 1;
```

Nonrecursive Binary Search
（非递归查找）

```
while (bottom < top) {
      int mid = (bottom + top)/2;
      the_list.retrieve(mid, data);
      if (target>data )
            bottom = mid +1;
      else
            top = mid;
     }
if (top < bottom) return not_present;
else {
      position = bottom;
      the_list.retrieve(bottom, data);
      if (data == target) return success;
      else return not_present;
}
```

Binary Search

- The Forgetful Version of Binary Search(二分查找的forgetful version)

Method: Forget the possibility that the Key target might be found quickly and continue, whether target has been found or not,

recursive Binary Search

（递归查找）

subdivide the list until what remains has length 1.

```
Error_code   recursive_binary_1(const Ordered_list &the_list,
const Key &target, int bottom, int top, int &position)
/* Pre: bottom and top define the range in the list to search
for the target .
Post: If a Record in the range of locations from bottom to top
in the list has key equal to target , then position locates one
such entry and a code of success is returned. Otherwise, the
Error code of not_ present is returned and position becomes
undefined.
Uses: recursive_binary_1 and methods of the classes List
and Record . */
{ Record data;
```

```
{
    Record data;
    if (top < bottom)
        return not_present; // List is empty.
        else { // List has exactly one entry.
            position = bottom;
            the_list.retrieve(bottom, data);
            if (data == target)
                return success;
            else
                return not_present; }    }
    else
    if (bottom < top) { // List has more than one entry.
        int mid = (bottom + top)/2;
        the_list.retrieve(mid, data);
        if (data < target) // Reduce to top half of list.
                return recursive_ binary_ 1(the_list, target,mid + 1, top, position)
        else // Reduce to bottom half of list.
                return recursive_binary_1(the_list, target,bottom, mid, position);
    }
```

0	1	1	1	4
0	1	2	3	4

寻找target=1的过程

Level0
（在run_recursive_binary_1中调用而进入）
recursive_binary_1(the_list,1,0,4,position)

Level1:
recursive_binary_1(the_list,1,0,2,position)

Level2:
recursive_binary_1(the_list,1,0,1,positio
n)

Level3:
recursive_binary_1(the_list,1,1,1,position)

```
if (bottom < top) {.
        int mid = (bottom + top)/2;
        the_list.retrieve(mid, data);
       if (data < target)
        return recursive_ binary_ 1(the_list,
      target,mid + 1, top, position);
       else
        return recursive_binary_1(the_list,
      target,bottom, mid, position);
}
else if (top < bottom)
         return not_present;
     else {
            position = bottom;
            the_list.retrieve(bottom, data);
             if (data == target)
                  return success;
             else
                   return not_present;
             }
```