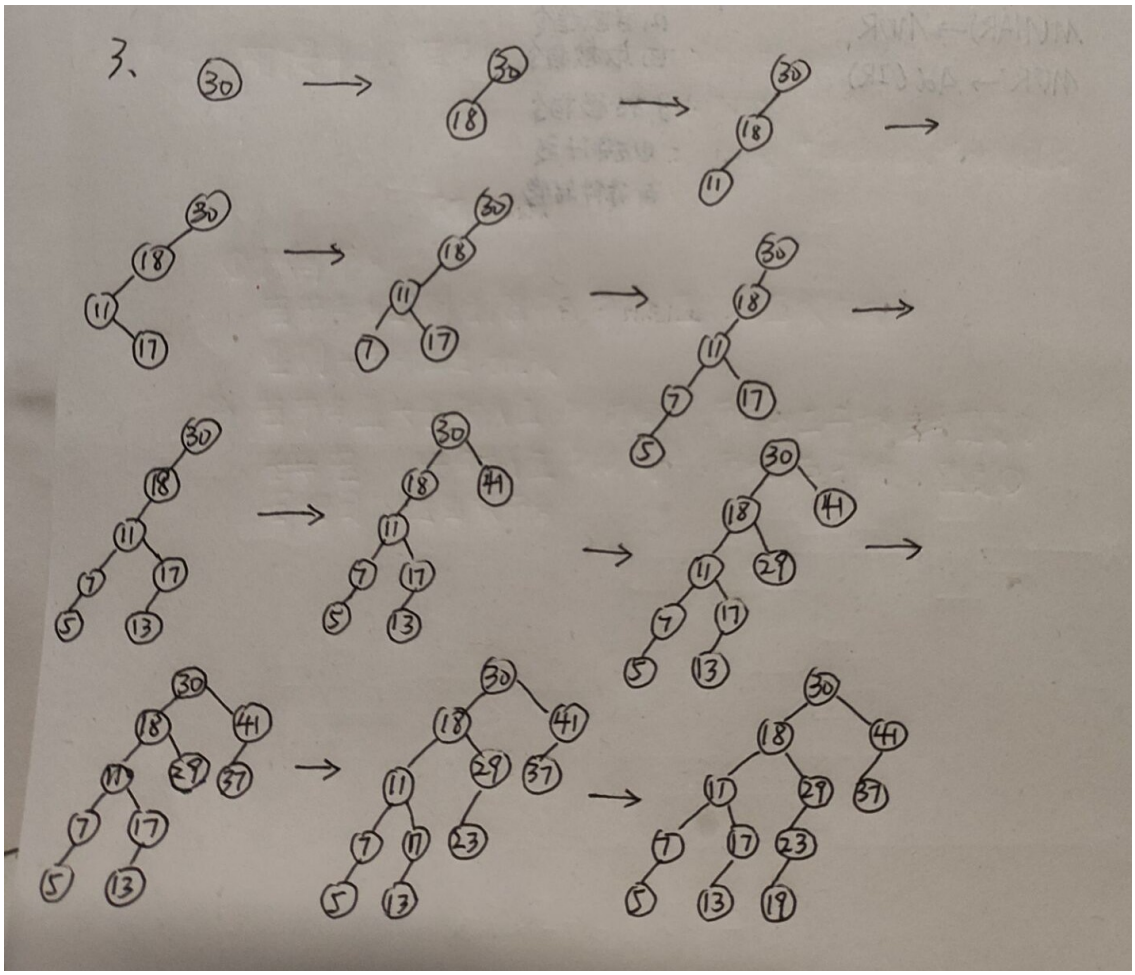
| 序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 值 | 1 | 3 | 9 | 12 | 32 | 41 | 45 | 62 | 75 | 77 | 82 | 95 | 100 |

1. 4次
    1. low = 1, high = 13, mid = (1+13)/2 = 7 ,  45 < 82
    2. low = 8, high = 13, mid = (8+13)/2 = 10 , 77 < 82
    3. low = 11, high = 13, mid = (11+13)/2 = 12 , 95> 82
    4. low = 11, high = 11, mid = (11+11)/2 = 11 , 82> 82  <=== 找到

| 地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 值 | | | | | 15 | 38 | 61 | 84 | | | |

2. 8
    1. $H(49) = 5$  X
    2. $5 + 1 = 6$  X
    3. $5 + 2 = 7$  X
    4. $5 + 3 = 8$  放入数据

3. 

4. 函数接受参数，数组 r、 数组 r 的长度n、 查找 元素 k。用二分查找， 查找k。 如果找到 k ，返回k的下标；如果没有找到，返回 -1
5. 如下

```
//A
q = p->lchild； 或者 q = q->lchild
//B
q = p->rchild 或者  q = q->rchild
//C
root = q
//D
p->lchild = q
//E
p->rchild = q
```

6. 算法如下：

   定义：

```
//具有如下的数据结构
typedef struct node{
    int data;
    struct node *lchild, *rchild;
}node;

//传入参数
储存数据的数组： int data[];
数组长度： int n;
```

   算法：

   1. 若 `n == 0` ，则返回 `NULL`
   2. 为`root`申请空间，并令`root->data = data[0]`
   3. 声明 `node *q,*p ; int i = 1`
   4. 令`q = NULL ; p = root`
   5. 如果 `p != NULL` , 则进入6, 否则进入7
   6. 若 `data[i] < p->data`, 就令 `q = p; p = q->lchild` ；否则令 `q = p; p = q->rchild`。回到第 5 步。
   7. 为`p`申请空间，并令`p->data = data[i]`
   8. 若 `p->data < q ->data`, 则令`q->lchild = p` , 否则令 `q->rchild = p`
   9. 令 `i++` ， 若 `i < n` ，则回到4，否则进入10。
   10. 返回 `root`

   或者

   1. 若 `n == 0` ，则返回 `NULL`
   2. 为`root`申请空间，并令`root->data = data[0]`
   3. 从data的第二个元素开始，依次插入 `root`中
   4. 返回 `root`

代码

```
typedef struct node{
    int data;
    struct node *lchild, *rchild;
}node;

node* createNode(int data){
    node *re = (node *)malloc(sizeof(node));
    re->data = data[0];
    re->lchild = NULL;
    re->rchild = NULL;
    return re;
}

node *createBT(int data[], int n)
{
    if (!n)
        return NULL;

    //创建根节点
    node *root = createNode(data[0]);
    //将后面的元素不停的插入树中
    //等价于
    // for (int i = 1; i < n; i++)
    // {
    //     root = insert(root, data[i]);
    // }
    node *q, *p;
    for (int i = 1; i < n; i++)
    {
        q = NULL;
        p = root;
        while (p != NULL)
        {
            if (data[i] < p->data)
            {
                q = p;
                p = q->lchild;
            }
            else
            {
                q = p;
                p = q->rchild;
            }
        }
        p = createNode(data[i]);
        if (p->data < q->data)
            q->lchild = p;
        else
            q->rchild = p;
    }
    return root;
}
```