1.

(1)

```
Node* search(Node* head, Node*& p, int K) {
    if (p == nullptr) p = head;
    if (p->key < K) {
        while (p != nullptr && p->key < K) {
            p = p->next;
        }
    }
    else if (p->key > K) {
        while (p != nullptr && p->key > K) {
            p = p->prev;
        }
    }
    if (p != nullptr && p->key == K) {
        return p;
    }
    return nullptr;
}
```

(2)
注意到:

$$ASP_{succ} = \frac{1}{n}\sum_{i\in[n]}\frac{1}{n}\sum_{j\in[n]}|i-j| = \frac{n}{3} - \frac{1}{3n}$$

2.

先对集合 S_2 进行排序，然后对于集合 S_1 中的每个元素，使用二分查找在排序后的 S_2 中检查该元素是否存在。若存在，则该元素是交集的一部分。

```
function heap_sort(S2):
    build_max_heap(S2)
    for i = length(S2) - 1 down to 1:
        swap S2[0] and S2[i]
        heapify(S2, 0, i)

function binary_search(S2, target):
    low = 0
    high = length(S2) - 1
    while low <= high:
        mid = (low + high) / 2
        if S2[mid] == target:
            return true
        else if S2[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return false

function intersection(S1, S2):
    heap_sort(S2)
    result = empty set  // 初始化交集结果集合
    // 第二步：对 S1 中的每个元素进行二分查找
    for each element x in S1:
        if binary_search(S2, x):
            result.add(x)
    return result
```

- 堆排序的时间复杂度为 $O(M \log M)$.
- 对于 S_1 中的每个元素，二分查找的时间复杂度为 $O(\log M)$.
- 因此，整个算法的总时间复杂度为：

$$O(M \log M + N \log M) = O(N \log \log N)$$

3.

(1)

HT	0	1	2	3	4	5	6	7	8	9
Key	21	14	3	5	20		9	37		

检索成功的平均长度:

$$\frac{1}{7}(1+2+1+1+1+2+3) \approx 1.57$$

(2)

HT	0	1	2	3	4	5	6	7	8	9
Key	21	5	3	14	20		9		37	

检索成功的平均长度:

$$\frac{1}{7}(1+2+1+1+1+2+1) \approx 1.29$$