严锦 1700011049

1.1

$$
\begin{gathered}
100n^2 < 2^n \\
\therefore -0.096704 < x < 0.103658 \ \vee \ x > 14.3247 \\
\therefore x_{\min} = 15
\end{gathered}
$$

1.2

$$
\begin{gathered}
\because f(n) < f(n) + g(n) \text{ and } g(n) < f(n) + g(n) \\
\therefore \max(f(n), g(n)) = \mathcal{O}(f(n) + g(n)) \\
\because f(n) + g(n) < 2\max(f(n), g(n)) \\
\therefore \max(f(n), g(n)) = \Omega(f(n) + g(n)) \\
\therefore \max(f(n), g(n)) = \Theta(f(n) + g(n))
\end{gathered}
$$

1.3

$$
\begin{aligned}
T_n &= T_{n-1} + n \\
T_{n-1} &= T_{n-2} + n - 1 \\
&\cdots \\
T_1 &= T_0 + 1
\end{aligned}
$$

$$
\begin{gathered}
\therefore T_n = T_0 + \sum_{i=1}^{n} i = T_0 + \frac{n(n+1)}{2} \\
\therefore T_n = \mathcal{O}(n^2)
\end{gathered}
$$

2.1

```cpp
template <class T>
struct node{
    T value;
    node* next;
}
bool findCircle(node* head){
    //快指针每次移动两步，慢指针每次移动一步
    node* p_fast = head;
    node* p_slow = head;
    while (p_fast){
        if (p_fast == p_slow){
            //若快慢指针相遇，则链表有环
```

```
            return true;
        }
        else {
            p_fast = p_fast->next;
            if (p_fast)
                p_fast = p_fast->next;
            else
                //若快指针走到尾节点，则链表无环
                return false;
            p_slow = p_slow->next;
        }
    }
    //若快指针走到尾节点，则链表无环
    return false;
}
```

时间复杂度

- 设链表总长度为m，最大环长度为n，则快指针进入环时，已移动$m-n$步。
- 在最坏的情况下，此时慢指针没有入环，且只移动了$\frac{m-n}{2}$步。
- 慢指针还需要$\frac{m-n}{2}$步才能入环，此时快指针已经移动了$\frac{m-n}{2}+(m-n)=\frac{3}{2}(m-n)$步。在最坏的情况下，快指针此时还没有在环内走完第一圈，与慢指针的距离为$n-\frac{m-n}{2}=\frac{3}{2}n-\frac{m}{2}$步。
- 快指针还需要$\frac{3}{2}n-\frac{m}{2}$即可追上慢指针，总步数为$(\frac{3}{2}n-\frac{m}{2})+\frac{3}{2}(m-n)=2m$，因此时间复杂度为$\mathcal{O}(m)$。

空间复杂度

只使用了两个指针，空间复杂度为$\mathcal{O}(1)$

2.2

```
#include <iostream>

using namespace std;

template<class T>
struct node {
    T item;
    node *next;
    node *jump;

    node(T v = -1) : item(v) {
        next = NULL;
        jump = NULL;
```

```
    }
};

template<class T>
class LinkList {
private:
    node<T> *head;
    node<T> *ptr;
    int size = 0;

private:
    void build_jump(int pos, int delta) {
        //二分法递归构造jump指针结构
        if (delta <= 0)
            return;

        int n_pos = pos + delta;
        node<T> *curr_ptr = head;
        node<T> *n_ptr = head;
        for (int i = 0; i < pos; ++i) {
            curr_ptr = curr_ptr->next;
        }
        if (curr_ptr->jump)
            return;
        for (int i = 0; i < n_pos; ++i) {
            n_ptr = n_ptr->next;
        }
        curr_ptr->jump = n_ptr;
        build_jump(n_pos, delta / 2);
        build_jump(pos + 1, delta / 2);
    }

    T find(int pos) {
        int delta = size / 2;
        node<T> *p = head->next;
        int curr_pos = 1;
        while (p && p->jump) {
            if (pos >= delta + curr_pos) {
                p = p->jump;
                curr_pos += delta;
            } else {
                p = p->next;
                curr_pos++;
            }

```

```
59              if (curr_pos == pos) {
60                  return p->item;
61              } else {
62                  delta /= 2;
63              }
64
65
66          }
67          return head->item;
68      }
69
70  public:
71      LinkList() {
72          head = new node<T>();
73          ptr = head;
74      }
75
76      void add_node(T item) {
77          ptr->next = new node<T>(item);
78          ptr = ptr->next;
79          size++;
80      }
81
82      void build() {
83          build_jump(1, size / 2);
84      }
85
86      T operator[](int pos) {
87          return find(pos);
88      }
89
90  };
91
92  int main() {
93      LinkList<int> lst;
94      for (int i = 1; i <= 50000; ++i)
95          lst.add_node(i);
96      lst.build();
97      cout << lst[30000] << endl;
98      return 0;
99  }
```

时间复杂度

- 设链表大小为size，则在最坏的情况下，查找时需要从第一个节点逐步跳转到下标增量(delta)为 $\frac{\text{size}}{2} \to \frac{\text{size}}{4} \to \frac{\text{size}}{8} \to \ldots$的各个jump节点，最多的跳转步数为$\log_2 \text{size}$，若出现跳转到next节点的情况，则跳转步数只会更少。因此时间复杂度为$\mathcal{O}(\log \text{size})$