**复习题第三题**

需要10次比较。前6次比较把7个元素分成两个等长的数组，然后再用四次比较把它拆成7个长度为1的数组，排序结束。

e.g. 4 2 1 3 6 5 7

**复习题第五题**

1. 直接选择排序
2. 直接插入排序或者冒泡排序
3. 堆排序或者快速排序
4. 堆排序
5. 堆排序

**双向的冒泡排序**

#include <vector>

#include <iostream>

#include <random>

// 交换元素

template <typename T>

inline void swap(T &x, T &y) {

T tmp = x;

x = y;

y = tmp;

}

// 双向的冒泡排序

template <typename T>

void double\_bubble\_sort(std::vector<T> &elements) {

// i 记录最前面的元素的序数， f 记录最后面元素的序数

int i(0), f(elements.size() - 1);

while (i != f) {

// 正向搜索

for (int index = i; index != f; ++index) {

if (elements[index] > elements[index + 1])

swap(elements[index], elements[index + 1]);

}

--f;

if (i == f)

break;

// 逆向搜索

for (int index = f; index != i; --index) {

if (elements[index] < elements[index - 1])

swap(elements[index], elements[index - 1]);

}

++i;

}

}

**单词的基数排序**

#include <string>

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

// 这个函数可以把小写字母转换成大写字母

void convert\_to\_capital(string &str) {

for (int i = 0; i < str.size(); ++i) {

if (islower(str[i])) {

str[i] = toupper(str[i]);

}

}

}

bool radix\_sort\_for\_text(vector<string> &str\_vec) {

// 记录单词的最大长度

int max\_length(0);

//转换为大写

for (int i = 0; i < str\_vec.size(); ++i) {

convert\_to\_capital(str\_vec[i]);

// 如果有非字母元素，返回 false

for (auto ch : str\_vec[i])

if (!isupper(ch))

return false;

}

// 找到最大长度

for (auto c : str\_vec)

max\_length = max\_length > c.size() ? max\_length : c.size();

// 把不足最大长度单词用空格补全

for (int i = 0; i < str\_vec.size(); ++i) {

int size = str\_vec[i].size();

for (int j = size; j < max\_length; ++j) {

str\_vec[i] += " ";

}

}

// 这是基数排序使用的队列

vector<queue<string> > queue\_vec;

queue\_vec.resize(27);

// 从最后一位开始排序

for (int i = max\_length - 1; i >= 0; --i) {

// 收集过程

for (auto c : str\_vec) {

if (isupper(c[i]))

queue\_vec[c[i] - 'A' + 1].push(c);

else

queue\_vec[0].push(c);

}

int index = 0;

// 分配过程

for (int i = 0; i < queue\_vec.size(); ++ i) {

while (!queue\_vec[i].empty()) {

str\_vec[index++] = queue\_vec[i].front();

queue\_vec[i].pop();

}

}

}

return true;

}