# 习题6 排序

6-1 如果顺序表中的大部分数据元素按关键字值递增有序，则采用( D )算法进行升序排序，比较次数最少。

(A) 快速排序

(B) 归并排序

(C) 选择排序

(D) 插入排序

6-2 若一组记录的关键码为46, 79, 56, 38, 40, 84，则利用快速排序方法且以第一个记录为基准，得到的一次划分结果为(C )。

(A) 38，40，46，56，79，84

(B) 40，38，46，79，56，84

(C) 40，38，46，56，79，84

(D) 40，38，46，84，56，79

6-3 对数据36，12，57，86，9，25进行排序，如果前三趟的排序结果如下：

第1趟：12，36，57，9，25，86

第2趟：12，36，9，25，57，86

第3趟：12，9，25，36，57，86

则采用的排序方法是( B )。

(A) 插入排序

(B) 起泡排序

(C) 归并排序

(D) 快速排序

6-4设int r[9]={0, 25, 28, 13, 33, 56, 47, 19, 40}; 则调用F(r, 1, 8)之后，数组r[ ]中的数据元素存放顺序是( D )。

F(int r[ ], int s, int t)

{

for (int j=2\*s; j≤t; j\*=2)

{

if (r[j+1]<r[j]) ++j;

if (r[s]≤r[j]) break;

int x=r[s];

r[s]=r[j];

r[j]=x;

s = j;

}

}

(A) 0, 13, 19, 25, 28, 33, 40, 47, 56

(B) 56, 47, 40, 33, 28, 25, 19, 13, 0

(C) 0, 25, 28, 13, 33, 40, 47, 19, 56

(D) 0, 13, 28, 19, 33, 56, 47, 25, 40

6-5 在链式基数排序中，对关键字序列369, 367, 167, 239, 237, 138, 230, 139进行第1趟分配和收集后，得到的结果是( C )。

(A) 167, 138, 139, 239, 237, 230, 369, 367

(B) 239, 237, 138, 230, 139, 369, 367, 167

(C) 230, 367, 167, 237, 138, 369, 239, 139

(D) 138, 139, 167, 230, 237, 239, 367, 369

6-6 设int r[7]={5，2，6，4，1，7，3}; 则执行for ( i=0; i<7; i++) r[r[i]-1]=r[i]; 命令之后，数组r[7]中的数据元素存放顺序是( D )。

(A) 5，2，7，4，1，6，3

(B) 3，2，1，4，5，7，6

(C) 1，2，3，4，5，6，7

(D) A、B、C都不对

6-7 设计一种排序算法，对1000个[0, 10000]之间的各不相同的整数进行排序，要求比较次数和移动次数尽可能少。

// 获取数字的某位上的值

int getDigit(int number, int position) {

return (number / static\_cast<int>(pow(10, position))) % 10;

}

// 链式基数排序

void radixSort(vector<list<int>>& buckets, vector<int>& data, int maxDigits) {

for (int position = 0; position < maxDigits; ++position) {

for (int val : data) {

int digit = getDigit(val, position);

buckets[digit].push\_back(val);

}

// 合并各个分组

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

data.insert(data.end(), buckets[i].begin(), buckets[i].end());

buckets[i].clear();

}

}

}

6-8 设顺序表的结点结构为(Type Key; int Next)，其中，Key为关键字，Next为链表指针。试设计静态链表排序算法。

// 选择排序算法

void selectionSort(Node arr[], int length) {

for (int i = 0; i < length - 1; ++i) {

int minIndex = i;

for (int j = i + 1; j < length; ++j) {

if (arr[j].Key < arr[minIndex].Key) {

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

}

}

// 静态链表排序算法

void staticListSort(Node arr[], int length) {

selectionSort(arr, length);

// 重新调整指针，构建有序链表

for (int i = 0; i < length - 1; ++i) {

arr[i].Next = i + 1;

}

arr[length - 1].Next = -1; // 最后一个结点的Next设为-1，表示链表结束

}

6-9 假设n个部门名称的基本数据存储在字符数组name[N][31]中，0≤n≤N≤20。试设计一个起泡排序算法，将n个部门名称按字典序重新排列顺序。

#include <iostream>

#include <cstring>

using namespace std;

const int N = 20;

const int MAX\_LENGTH = 31;

// 交换两个字符串

void swapStrings(char str1[], char str2[]) {

char temp[MAX\_LENGTH];

strcpy(temp, str1);

strcpy(str1, str2);

strcpy(str2, temp);

}

// 起泡排序

void bubbleSort(char names[][MAX\_LENGTH], int n) {

for (int i = 0; i < n - 1; ++i) {

for (int j = 0; j < n - i - 1; ++j) {

// 使用strcmp比较字符串的字典序

if (strcmp(names[j], names[j + 1]) > 0) {

swapStrings(names[j], names[j + 1]);

}

}

}

}

int main() {

int n;

// 假设部门名称存储在字符数组name[N][31]中

char names[N][MAX\_LENGTH];

// 输入部门名称的个数n

cout << "Enter the number of department names (0 <= n <= 20): ";

cin >> n;

// 输入部门名称

cout << "Enter the department names:" << endl;

for (int i = 0; i < n; ++i) {

cin >> names[i];

}

// 执行起泡排序

bubbleSort(names, n);

// 输出排序后的部门名称

cout << "\nSorted Department Names:" << endl;

for (int i = 0; i < n; ++i) {

cout << names[i] << endl;

}

return 0;

}

6-10 假设采用链表存储类型：

typedef struct RNode

{

int key; //数据域(也是关键字域)

struct RNode \*next; //指针域

} RNode, \*RList;

typedef RList R[N]; //链表类型, 常变量N≥n

又设R[1..n]是[10, 999]之间的随机整数。试设计一个链表基数排序算法，将R[n]中的数从小到大排序。排序结果仍存放在R[n]中。

// 获取数字的某位上的值

int getDigit(int number, int position) {

return (number / static\_cast<int>(pow(10, position))) % 10;

}

// 将链表按照某一位的值进行分配

void distribute(RList& source, RList buckets, int position) {

RNode\* current = source;

while (current != nullptr) {

int digit = getDigit(current->key, position);

RNode\* temp = current->next;

current->next = buckets[digit];

buckets[digit] = current;

current = temp;

}

}

// 合并所有的链表

void collect(RList& source, RList buckets) {

for (int i = 0; i < N; ++i) {

RNode\* current = buckets[i];

while (current != nullptr) {

RNode\* temp = current->next;

current->next = source;

source = current;

current = temp;

}

}

}

// 基数排序

void radixSort(RList& r, int n) {

// 获取最大值的位数

int maxKey = r[0]->key;

for (int i = 1; i < n; ++i) {

if (r[i]->key > maxKey) {

maxKey = r[i]->key;

}

}

int maxDigits = static\_cast<int>(log10(maxKey) + 1);

// 创建桶

R buckets;

// 进行基数排序

for (int position = 0; position < maxDigits; ++position) {

// 分配

distribute(r, buckets, position);

// 合并

collect(r, buckets);

}

}

6-11 在下列排序算法中，时间复杂度最好的是( A )。

(A) 堆排序

(B) 插入排序

(C) 起泡排序

(D) 选择排序

6-12 根据建堆算法，将关键字序列5，7，10，8，6，4调整成一个大顶堆，最少的交换次数为( B )。

(A) 1

(B) 2

(C) 3

(D) 4

6-13 给定关键字序列503, 87, 512, 61, 908, 170, 897, 275, 653, 426。

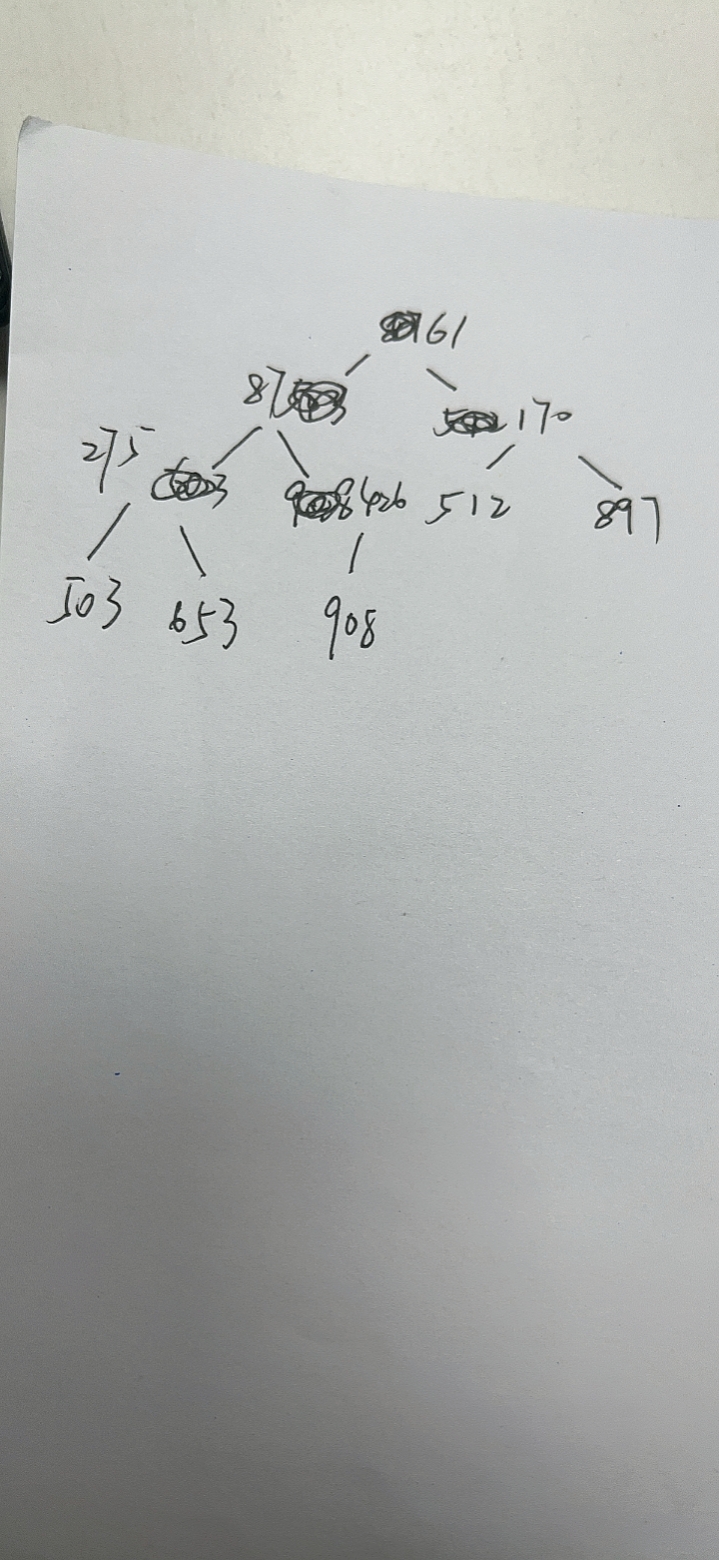
1. 以第一个关键字为枢轴，给出第1趟快速排序后的关键字序列;

第一躺快排后

426, 87,275,61,170,503,897,908,653,512

1. 给出根据堆排序算法建立的小顶堆序列。

61,87,170,275,426,512,897,503,653,908



6-14 设计基于顺序表存储结构的树形选择排序算法。

void buildTree(vector<int>& seq, int n) {

for (int i = n - 1; i > 0; --i) {

for (int j = (i + 1) / 2 - 1; j >= 0; --j) {

int leftChild = 2 \* j + 1;

int rightChild = 2 \* j + 2;

int minIndex = j;

if (leftChild < i && seq[leftChild] < seq[minIndex]) {

minIndex = leftChild;

}

if (rightChild < i && seq[rightChild] < seq[minIndex]) {

minIndex = rightChild;

}

if (minIndex != j) {

swap(seq[j], seq[minIndex]);

}

}

}

}

void treeSelectionSort(vector<int>& seq, int n) {

buildTree(seq, n);

for (int i = n - 1; i > 0; --i) {

swap(seq[0], seq[i]);

for (int j = (i + 1) / 2 - 1; j >= 0; --j) {

int leftChild = 2 \* j + 1;

int rightChild = 2 \* j + 2;

int minIndex = j;

if (leftChild < i && seq[leftChild] < seq[minIndex]) {

minIndex = leftChild;

}

if (rightChild < i && seq[rightChild] < seq[minIndex]) {

minIndex = rightChild;

}

if (minIndex != j) {

swap(seq[j], seq[minIndex]);

}

}

}

}