# 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(  **A**  )。

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。

// 获取数字的某一位上的值

int getDigit(int number, int place) {

    while (place > 0) {

        number /= 10;

        place--;

    }

    return number % 10;

}

// 基数排序

void radixSort(vector<int>& arr) {

    const int maxDigits = 5;  // 最大位数，假设数字范围为[0, 10000]

    const int base = 10;      // 基数

    // 创建10个桶

    vector<vector<int>> buckets(base);

    // 进行maxDigits次分配和收集

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

        // 分配到桶中

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

            int bucketIndex = getDigit(arr[i], digit);

            buckets[bucketIndex].push\_back(arr[i]);

        }

        // 从桶中收集

        arr.clear();

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

            arr.insert(arr.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。

struct Node {

    int Key;

    int Next;

};

// 初始化静态链表

void InitList(Node\* list) {

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

        list[i].Next = i + 1; // 设置链表的Next指针，最后一个结点的Next为0表示链表结束

    }

    list[MAXSIZE - 1].Next = 0;

}

// 输出静态链表

void PrintList(Node\* list) {

    int index = list[0].Next; // 从第一个结点开始遍历

    while (index != 0) {

        cout << list[index].Key << " ";

        index = list[index].Next;

    }

    cout << endl;

}

// 静态链表排序算法

void StaticListSort(Node\* list) {

    int i, j, p, q;

    for (i = list[0].Next; i != 0; i = list[i].Next) {

        for (j = list[i].Next; j != 0; j = list[j].Next) {

            if (list[i].Key > list[j].Key) {

                // 交换结点的Key值

                swap(list[i].Key, list[j].Key);

                // 调整Next指针

                p = 0;

                q = list[0].Next;

                while (list[q].Key < list[j].Key) {

                    p = q;

                    q = list[q].Next;

                }

                list[p].Next = j;

                list[i].Next = list[j].Next;

                list[j].Next = q;

            }

        }

    }

}

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。

void bubbleSort(char names[N][31], int n) {

    for (int i = 0; i < n - 1; i++) {

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

            // 使用strcmp比较两个字符串的大小

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

                // 交换两个字符串

                char temp[31];

                strcpy(temp, names[j]);

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

                strcpy(names[j + 1], temp);

            }

        }

    }

}

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。

typedef RList R[10]; // 链表数组，用于存放基数排序过程中的桶

// 初始化链表

void InitList(RList& L) {

    L = nullptr;

}

// 将结点插入链表头部

void InsertToHead(RList& L, int key) {

    RNode\* newNode = new RNode;

    newNode->key = key;

    newNode->next = L;

    L = newNode;

}

// 输出链表

void PrintList(RList L) {

    RNode\* current = L;

    while (current != nullptr) {

        cout << current->key << " ";

        current = current->next;

    }

    cout << endl;

}

// 基数排序算法

void RadixSort(RList R[], int n) {

    const int numDigits = 3; // 假设数字范围在[10, 999]之间，最多3位数字

    for (int digit = 1; digit <= numDigits; digit++) {

        // 将数据分配到桶中

        for (int i = 0; i < n; i++) {

            int bucketIndex = (R[i]->key / (int)pow(10, digit - 1)) % 10;

            InsertToHead(R[bucketIndex], R[i]->key);

        }

        // 从桶中收集数据

        int index = 0;

        for (int i = 0; i < 10; i++) {

            while (R[i] != nullptr) {

                RNode\* temp = R[i];

                R[i] = R[i]->next;

                RNode\* current = new RNode;

                current->key = temp->key;

                current->next = nullptr;

                delete temp;

                R[index++] = current;

            }

        }

    }

}

6-11 在下列排序算法中，时间复杂度最好的是( **B** )。

(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;

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

(2)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。

// 交换数组中的两个元素

void swap(vector<int>& arr, int i, int j) {

    int temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

}

// 将无序序列构建成二叉树

void buildHeap(vector<int>& arr, int n, int i) {

    int smallest = i; // 初始化根节点为最小值

    int leftChild = 2 \* i + 1; // 左子节点索引

    int rightChild = 2 \* i + 2; // 右子节点索引

    // 比较根节点和左子节点

    if (leftChild < n && arr[leftChild] < arr[smallest]) {

        smallest = leftChild;

    }

    // 比较根节点和右子节点

    if (rightChild < n && arr[rightChild] < arr[smallest]) {

        smallest = rightChild;

    }

    // 如果根节点不是最小值，则交换并递归调整子树

    if (smallest != i) {

        swap(arr, i, smallest);

        buildHeap(arr, n, smallest);

    }

}

// 树形选择排序算法

void treeSort(vector<int>& arr, int n) {

    // 构建最小堆

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

        buildHeap(arr, n, i);

    }

    // 选择最小值，交换根节点和最后一个节点

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

        swap(arr, 0, i);

        // 重新构建最小堆

        buildHeap(arr, i, 0);

    }

}