《数据结构与算法》实验报告

实验题目	排序与二叉树实践		
实验时间	2023 年 11 月 17 日 9:00-12:00	实验地点	DS3402
实验成绩		实验性质	☐验证性 √设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

实验目的

1. 掌握二叉树、二叉查找树以及堆的基本原理
2. 训练使用二叉树与堆基本操作，通过编程解决不同难度问题的实践能力

二、实验项目内容

注：每道题按下面的格式分别描述

实验题目 1：

题目内容：快速排序的过程

给定 n 个整型元素，利用快速排序算法对其进行非递减排序，请输出每一趟 Partition 的结果。每次选择所处理的区间的第一个元素作为基准元素。

代码实现：

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
using namespace std;
int n;
```

报告创建时间：2023/11/25

```
void mysort(int* a, int l, int r);
int partition(int* a, int l, int r);
void swap1(int& a, int& b);
void myprint(int* a);

int main() {
	cin >> n;
	int* a = new int[n];
	for (int i = 0; i < n; i++) {
		cin >> a[i];
	}
	mysort(a, 0, n - 1);
}
void mysort(int* a, int l, int r) {
	//cout << "mysort a" << l << r << endl;
	if (l < r) {
		int i = partition(a, l, r);
		myprint(a);
		if (i > l)
			mysort(a, l, i - 1);
		if (i < r)
			mysort(a, i + 1, r);

	}
	/*
	else if (l == r - 1 && (a[r] < a[l])) {
		swap1(a[r], a[l]);
		myprint(a);
	}*/
	else if(l==r) myprint(a);
}
```

```
int partition(int* a, int l, int r) {
	int i = l + 1, j = r;
	while (i < j) {
		while (a[i] < a[l] && i < r)i = i + 1;
		while (a[l] < a[j] && j > l)j = j - 1;
		if (i < j) {
			swap1(a[i], a[j]);
			i = i + 1;
			j = j - 1;
		}
	}
	swap1(a[j], a[l]);
	//cout << "partition"<<j << endl;
	return j;
}

void swap1(int& a, int& b) {
	int t = a;
	a = b;
	b = t;
}

void myprint(int*a) {
	for (int i = 0; i < n; i++) {
		cout << a[i] << " ";
	}
	cout << endl;
}
```

实验题目 2：

题目内容：数据结构考题-二叉树的遍历（中序）

以二叉链表作存储结构，建立一棵二叉树， 输出该二叉树的中序遍历序列。

二叉链表的类型描述：

```
typedef char ElemType;
typedef    struct    BiNode
{    ElemType    data;
      struct    BiNode     *lchild,*rchild;
}BiNode,*BiTree;
```

代码实现：

```
#include<iostream>
#include<string>
using namespace std;
typedef char ElemType;
typedef    struct    BiNode
{
      ElemType    data;
      struct    BiNode* lchild, * rchild;
}BiNode, * BiTree;

BiTree mycreat(string s, int& k);
void inorder(BiTree Tree);

int main() {
      string s;
      getline(cin, s);
      int k = 0;
      BiTree Tree = mycreat(s, k);
      inorder(Tree);
}

BiTree mycreat(string s, int& k) {
```

```
    BiTree Tree = NULL;
    int n = s.length();
    if (k < n) {
        char c = s[k];
        k++;
        if (c != '#') {
            Tree = new BiNode();
            Tree->data = c;
            Tree->lchild = mycreat(s, k);
            Tree->rchild = mycreat(s, k);
        }
    }
    return Tree;
}

void inorder(BiTree Tree) {
    if (Tree == NULL) return;
    inorder(Tree->lchild);
    cout << Tree->data;
    inorder(Tree->rchild);
}
```

实验题目 3：

题目内容：最大三角形

有一个游戏，玩法是在一堆长度不一的小棍中找出三根棍子，拼出一个周长最大的三角形。有什么策略能快速的找到三根小棍么？

输入格式:

在一行中给出小棍的个数 N，另一行中分别给出 N 个小棍的长度，之间用空格隔开。

输出格式:

如果小棍的数量小于 3，则输出小棍的个数不能组成三角形；如果找到最大的

三角形，则输出最大三角形的周长是?，并在下一行中输出组成最大三角形的三条边是?,?,?，三条边之间用英文逗号隔开并从小到大输出；如果没有找到，则输出没有找到能组成三角形的小棍。

代码实现：

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

bool canbuild(int* a);
bool cmp(int x, int y) { return x > y; }

int main() {
	int n;
	cin >> n;
	if (n < 3) {
		cout << "小棍的个数不能组成三角形";
		return 0;
	}
	int* p = new int[n];
	for (int i = 0; i < n; i++) {
		cin >> p[i];
	}
	sort(p, p + n, cmp);
	int a[3] = { p[0],p[1],p[2] };
	bool flag = 0;
	int k = 2;
	while (!flag && k < n) {
		if (canbuild(a)) { flag = 1; }
		else {
```

```
                k++;
                if (k < n) {
                    a[0] = a[1];
                    a[1] = a[2];
                    a[2] = p[k];
                }
            }
        }
        if (flag) {
            cout << "最大三角形的周长是" << a[0] + a[1] + a[2] <<
                "\n 组成最大三角形的三条边是" << a[2] << "," << a[1] << ","
<< a[0];
        }
        else
            cout << "没有找到能组成三角形的小棍";
}

bool canbuild(int* a) {
        if (a[0] < a[1] + a[2])return true;
        else return false;
}
```

实验题目 4：

题目内容：最大三角形

根据维基百科的定义：

插入排序是迭代算法，逐一获得输入数据，逐步产生有序的输出序列。每步迭代中，算法从输入序列中取出一元素，将之插入有序序列中正确的位置。如此迭代直到全部元素有序。

归并排序进行如下迭代操作：首先将原始序列看成 N 个只包含 1 个元素的有序子序列，然后每次迭代归并两个相邻的有序子序列，直到最后只剩下 1 个有序的序列。

现给定原始序列和由某排序算法产生的中间序列，请你判断该算法究竟是哪种

排序算法？

输入格式：

输入在第一行给出正整数 N (≤100)；随后一行给出原始序列的 N 个整数；最后一行给出由某排序算法产生的中间序列。这里假设排序的目标序列是升序。数字间以空格分隔。

输出格式：

首先在第 1 行中输出 Insertion Sort 表示插入排序、或 Merge Sort 表示归并排序；然后在第 2 行中输出用该排序算法再迭代一轮的结果序列。题目保证每组测试的结果是唯一的。数字间以空格分隔，且行首尾不得有多余空格。

代码实现：

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

bool issame(int* a,int* b,int l,int r);
bool insort(int* a, int l, int r);
void insertsort(int* a, int num);
void mergesort(int *a);
void myprint(int *a);
void twowaymerge(int* p, int lx, int rx, int ly, int ry);
int n;

int main() {
	cin >> n;
	bool insert = 0;
	int insertnum;
	int* std = new int[n];
```

```
        for (int i = 0; i < n; i++) {
            cin >> std[i];
        }
        int* a = new int[n];
        for (int i = 0; i < n; i++) {
            cin >> a[i];
        }
        for (int i = 0; i < n; i++) {
            if (issame(a, std, i, n - 1)&&insort(a,0,i-1)) {
                insert = 1;
                insertnum = i;
                break;
            }
        }
        if (insert) {
            cout << "Insertion Sort\n";
            insertsort(a, insertnum);
        }
        else {
            cout << "Merge Sort\n";
            mergesort(a);
        }
    }


    bool issame(int* a,int *b, int l, int r) {
        for (int i = l; i <= r; i++) {
            if (a[i] != b[i])return false;
        }
```

```
        return true;
    }

    void insertsort(int *a, int num) {

        int t = a[num];
        if (t < a[num - 1]) {
            a[num] = a[num - 1];
            bool sw = 0;
            for (int i = num - 1; i > 0; i--) {
                if (t < a[i])a[i] = a[i - 1];
                else {
                    a[i+1] = t;
                    sw = 1;
                    break;
                }
            }
            if (!sw)a[0] = t;
        }
        myprint(a);
    }

    void mergesort(int* a) {
        bool flag = 0;
        int* a0 = a;
        int len = 1;
        while (len < n) {
            int lx = 0;
            while (lx < n-1-len) {
```

```
                int rx = lx + len - 1;
                int ly = rx + 1;
                int ry = min(ly + len-1, n-1);
                twowaymerge(a, lx, rx, ly, ry);

                lx = ry + 1;
            }
            if (flag) { myprint(a); return; }
            if (issame(a, a0, 0, n - 1))flag = 1;
            len *= 2;
        }
    }

    void myprint(int* a) {
        cout << a[0];
        for (int i = 1; i < n; i++) {
            cout << " " << a[i];
        }
    }

    void twowaymerge(int* p, int lx, int rx, int ly, int ry) {
        int n0 = ry + ly - rx - lx + 2;
        int* p1 = new int[n0];
        int m = lx, n = ly;
        int i = 0;
        while (m <= rx && n <= ry) {
            if (p[m] <= p[n]) {
                p1[i] = p[m];
                i++;
```

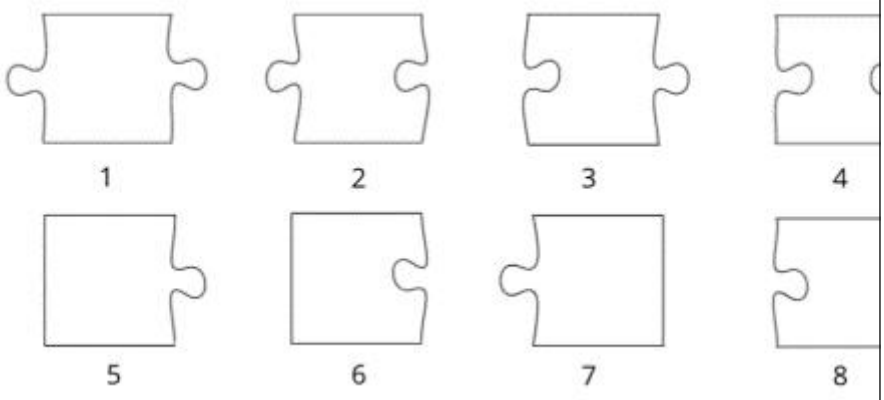
```
            m++;
        }
        else {
            p1[i] = p[n];
            i++;
            n++;
        }
    }
    if (m <= rx) {
        while (i < n0) { p1[i] = p[m]; m++; i++; }
    }
    if (n <= ry) {
        while (i < n0) { p1[i] = p[n]; n++; i++; }
    }
    int k = 0;
    for (int i = lx; i < ry + 1; i++) {
        p[i] = p1[k++];
    }
    delete[] p1;
}

bool insort(int* a, int l, int r) {
    for (int i = l; i < r; i++) {
        if (a[i] > a[i + 1])return false;
    }
    return true;
}
```

实验题目 5：BFS

题目内容：快速排序的过程

小费边得到了一个由 *N* 块组成的一维拼图。他很快意识到每件作品都属于以下类型之一：



此外，我们知道，在这 *N* 个部件中，有一个部件属于类型 5 或类型 6（左边框），另一个部件属于类型 7 或类型 8（右边框）。*Fabian* 希望将所有的棋子排成一行，这样第一个（最左边的）棋子是 5 型或 6 型，最后一个（最右边的）棋子是 7 型或 8 型。当且仅当两块相邻的边缘形状不同时，两块可以相邻放置，即一块有凸起（也称为 *outie* 或 *tab*），另一块有孔（也称为 *innie* 或 *blank*）。

简单地解决这个难题对费边来说太容易了，所以他决定在每一块上写一个唯一的正整数。现在，他感兴趣的是寻找拼图游戏的最小词典解决方案。如果在第一个位置（从左到左）*i* 处，答案 *A* 被认为比答案 *B* 小，而在第一个位置（从左到左）*i* 处，答案 *A* 与答案 *B* 不同，则答案 *A* 认为 *A* 中第 *i* 个拼图上的数字小于 *B* 中第 *i* 个拼图上的数字。

注意：拼图不能旋转

输入格式:

第一行包含一个整数 *N*（2≤*N*≤105），根据任务描述。
下一行包含两个整数 *Xi* （1≤*Xi* ≤8）和 *Ai* （1≤*Ai* ≤109）代表第 *i* 件作品的类型和费边在上面写的数字。所有的 *Ai* 都会不同。

输出格式:

如果费边不能解决拼图游戏，你应该输出−1。
否则，您应该在拼图的字典最小解中输出写在拼图块上的数字。

得分:

在总共值 5 分的测试用例中，它将保持 *N*≤4.
在价值额外 5 分的测试案例中，它将保持 *N*≤10

在价值额外 10 分的测试用例中，类型 2 和类型 3 不会出现在输入中。
在价值额外 20 分的测试用例中，最多有一件 1 型或 4 型。
如果对于存在谜题解决方案的某个测试用例，您输出正确解决的谜题，但您的解决方案不是最小的，那么您将获得该测试用例预期分数的 40%。

代码实现：

```
#include<iostream>
using namespace std;
int main(){

    cout<<"-1";
}
```

三、思考题

遍历二叉树是按一定的顺序依次访问树中各结点并输出结点存放的数据。二叉树有四种遍历方式：前序、中序、后序遍历即层序遍历。假设二叉树各结点的数据互异。如果出现如下的情况，即对二叉树用两种不同的方式遍历，生成的结点序列却相同，则该二叉树具有什么特征？

1）前序遍历与中序遍历相同
所有结点均无左子树
2）后序遍历与中序遍历相同
所有结点均无右子树
3）前序遍历与后序遍历相同
该二叉树为空树只由一个根节点构成
4）前序遍历与层序遍历相同
该二叉树的所有结点中最多有一个结点有右子树，且该右子树对应的节点在最低层