Laboratory work # 3

Student: *Zhanghao Chen*
Student ID: 21321205
Timus Name: *BennyChan*

Mail: 1824362950@qq.com / czhczf2003@foxmail.com

(Two different domain names of the same email)

Problem # 1207 *Median on the Plane*

Screenshot from Timus:

10244934	19:14:09 16 Apr 2023	BennyChan	1207. Median on the Plane	G++ 9.2 x64	Accepted		0.046	608 KB

Explanation of algorithm:

In this program, I first enter three integers n, x, and y. Then, generate a matrix M of $2^n \times 2^n$ by calling the paveSquare function. Among them, the paveSquare function takes a recursive way to continuously divide the matrix until the matrix size is 2×2. The function findFake is used to find the fake cell of each matrix block, where, x and y specify the upper-left position of the matrix block currently being processed, and *fakex* and *fakey* specify the position of the deleted cell, and the value of each position in the matrix is successively a multiple of 3. The resulting matrix M is output at the end of the program.

Computational complexity of algorithm:

$$O(N \times \log_2 N)$$

Source code:

```
#include <algorithm>
#include <iostream>

using namespace std;
double values[10000];
int main() {
    int n;
    cin >> n;
    int X[n], Y[n];
    for (int i = 0; i < n; i++) {
```

```
        cin >> X[i] >> Y[i];
    }
    int min_x = X[0];
    int min_index = 0;
    for (int i = 1; i < n; i++) {
        if (X[i] < min_x || (X[i] == min_x && Y[i] < Y[min_index])) {
            min_x = X[i];
            min_index = i;
        }
    }


    for (int i = 0; i < n; i++) {
        if (i != min_index) {
            values[i] = (Y[i] - Y[min_index]) * 1.0 / (X[i] - X[min_index]);
        } else {
            values[min_index] = -2e7;
        }
    }
    double copy_values[n];
    for(int i = 0; i < n; i++){
        copy_values[i] = values[i];
    }
    sort(values, values + n);
    for (int i = 0; i < n; i++) {
        if (values[n / 2] - copy_values[i]) {
            cout << min_index + 1 << " " << i + 1 << endl;
            return 0;
        }
    }
}
```

Problem # 1604 *Country of Fools*

Screenshot from Timus:

10249880	16:33:44 21 Apr 2023	BennyChan	1604. Country of Fools	G++ 9.2 x64	Accepted		0.031	304 KB

Explanation of algorithm:

This code implements an algorithm that converts a sequence of positive integers into a particular form. The main idea is to use a priority_queue defined by a lambda function to maintain a collection of nodes as well as define a priority queue pq. By using a loop to take the two smallest nodes a and b from the priority queue pq, print out their ids, and rejoin the queue by subtracting their values. Until the queue is empty, the desired sequence is obtained.

Computational complexity of algorithm:

$$O(N \times \log_2 N)$$

Source code:

```
#include <iostream>
#include <queue>
```

```cpp
#include <algorithm>
#include <functional>

using namespace std;

struct Node {
    int val;
    int id;
    Node(int val, int id) : val(val), id(id) {}
};

int main() {
    int n;
    cin >> n;

    auto cmp = [](Node a, Node b) { return a.val < b.val; };
    priority_queue<Node, vector<Node>, decltype(cmp)> pq(cmp);

    for (int i = 1; i <= n; i++) {
        int num;
        cin >> num;
        pq.push(Node(num, i));
    }

    while (!pq.empty()) {
        Node a = pq.top();
        pq.pop();
        if (pq.empty()) {
            for (int i = 0; i < a.val; i++) {
                cout << a.id << " ";
            }
            break;
        }
        Node b = pq.top();
        pq.pop();
        cout << a.id << " " << b.id << " ";
        if (a.val > 1) {
            pq.push(Node(a.val - 1, a.id));
        }
        if (b.val > 1) {
            pq.push(Node(b.val - 1, b.id));
        }
    }
    return 0;
}
```

Problem # 1444 *Elephpotamus*

Screenshot from Timus:

10249908	16:45:14 21 Apr 2023	BennyChan	1444. Elephpotamus	G++ 9.2 x64	Accepted		0.125	772 KB

Explanation of algorithm:

The problem can be regarded as a convex hull algorithm. Firstly, using a structure called point to store the horizontal and vertical coordinates of each point and the number of that point. Among them, the number is useful in the final output of the point on the convex hull to output the number of the point. Then, define a comparison function *cmp* used for

sorting, and its main function is to sort by the angle between the line of this point and the first point and the *x* axis. Where, if the line between the two points and the angle between the *x* axis are the same, the two points are sorted in order of their distance from the first point from nearest to far. This ensures that points at the same angle are sorted from nearest to far. This sorting method ensures that the points are sorted counterclockwise. In the main function, you first enter the number of points and the horizontal and vertical coordinates of each point and store the number of each point in the structure. Then, shift each point to the first point and sort according to the function *cmp*. The sorted array is the point sorted counterclockwise. Next, find the starting point on the convex hull. Initialize *f* to 0, then iterate through the sorted points from front to back, finding the first point that turns counterclockwise and assigning *f* as its index. Finally, the point on the convex hull is printed. Print the number of points first, then the number of the first point and the numbers of the other points.

Computational complexity of algorithm:

$$O(N \times \log_2 N)$$

Source code:

```
#include <iostream>
#include <algorithm>
#include <cmath>

using namespace std;
struct point {
    int x, y;
    int i;
} p[30002];
point c;

int comp(point& a, point& b) {
    if(a.x * b.y == a.y * b.x && a.x * b.x + a.y * b.y >= 0)
        return a.x * a.x + a.y * a.y < b.x * b.x + b.y * b.y;
    return atan2(a.y, a.x) < atan2(b.y, b.x);
}

int main() {
    int n;
    cin >> n;

    for(int i = 0; i < n; i++) {
        int x, y;
        cin >> p[i].x >> p[i].y;
        p[i].i = i;
    }

    for(int i = n-1; i >= 0; i--)
        p[i].x -= p[0].x, p[i].y -= p[0].y;

```

```
    sort(p, p+n, comp);

    int f = 0;
    for(int i = 0; i < n-1; i++) {
        point p0 = p[0], p1 = p[i], p2 = p[i+1];
        int d1x = p1.x - p0.x, d2y = p2.y  - p0.y, d1y = p1.y -
  p0.y, d2x = p2.x - p0.x;
        int x = d1x * d2y - d1y * d2x, d = d1x * d2x + d1y * d2y;
        if(x < 0 || x == 0 && d < 0) {
            f = i;
            break;
        }
    }
    cout << n << endl;
    cout << (p[0].i+1) << endl;
    for(int i = 0; i < n-1; i++)
        cout << (p[(f+i)%(n-1)+1].i+1) << endl;
}
```