Bài 1.

a) Thuật toán xác định bao lồi:

Bước 1: Sắp xếp các điểm theo thứ tự tăng dần của hoành độ (x). Nếu có các điểm có cùng hoành độ, sắp xếp chúng theo tung độ (y) tăng dần

Bước 2: Xây dựng bao lồi trên (hullUpper)

+ Khởi tạo danh sách rỗng cho hullUpper

+ Duyệt qua các điểm đã sắp xếp (từ 1 đến n). Thực hiện vòng while với mỗi điểm, kiểm tra xem điểm đang kiểm tra và 2 điểm cuối cùng có tạo thành một góc quay theo chiều ngược chiều kim đồng hồ (counter-clockwise, ccw) không. Nếu không, loại bỏ điểm cuối danh sách hull cho đến khi điểm đang kiểm tra và 2 điểm cuối cùng quay ngược chiều kim đồng hồ

+ Tiếp tục để thêm vào danh sách hullUpper

Bước 3: Xây dựng bao lồi dưới (hullLower)

+ Khởi tạo danh sách rỗng cho hullLower

+ Duyệt qua các điểm theo thứ tự ngược lại (từ n – 1 đến 1). Thực hiện tương tự như bước 2 để xây dựng chuỗi dưới

Bước 4: Kết hợp các điểm từ chuỗi trên và chuỗi dưới để tạo thành bao lồi. Lưu ý không lặp lại điểm đầu và điểm cuối, vì chúng đã có mặt trong cả chuỗi trên và chuỗi dưới

b) Chương trình cài đặt:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>
using namespace std;

struct Point {
    int x;
    int y;

    bool operator<(const Point& other) const {
        return x < other.x || (x == other.x && y < other.y);
    }
};

bool counterClockWise(const Point& A, const Point& B, const Point& C) {
    return (B.x * C.y - B.y * C.x) - A.x * (C.y - B.y) + A.y * (C.x - B.x) >=
0;
}

vector<Point> convexHull(vector<Point> points) {
```

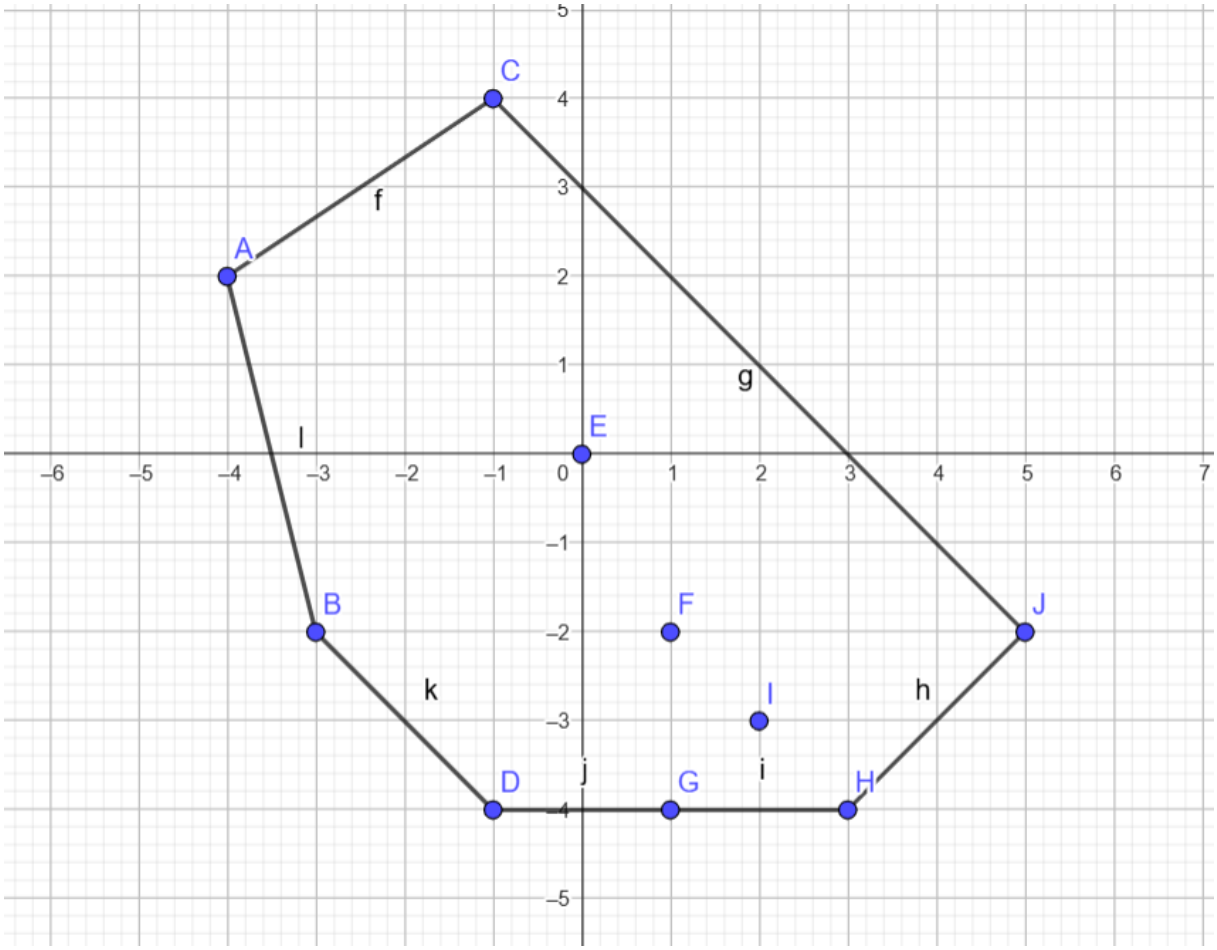
```
    int n = points.size();
    if (n < 3) return points;

    sort(points.begin(), points.end());

    vector<Point> hullUpper;

    for (int i = 0; i < n; i++) {
        while (hullUpper.size() >= 2
        && !counterClockWise(hullUpper[hullUpper.size() - 2],
                hullUpper[hullUpper.size() - 1], points[i])) {
            hullUpper.pop_back();
        }
        hullUpper.push_back(points[i]);
    }

    vector<Point> hullLower;

    for (int i = n - 2; i >= 0; i--) {
        while (hullLower.size() >= 2
        && !counterClockWise(hullLower[hullLower.size() - 2],
                hullLower[hullLower.size() - 1], points[i])) {
            hullLower.pop_back();
        }
        hullLower.push_back(points[i]);
    }


    hullUpper.insert(hullUpper.end(), hullLower.rbegin(), hullLower.rend());

    vector<Point> result;
    set<Point> seen;
    for (const Point& p : hullUpper) {
        if (seen.insert(p).second) {
            result.push_back(p);
        }
    }

    return result;
}

int main() {
    vector<Point> points = {{-4, 2}, {-3, -2}, {-1, 4}, {-1, -4}, {0, 0},
                            {1, -2}, {1, -4}, {2, -3}, {3, -4}, {5, -2}
};
    vector<Point> hull = convexHull(points);

    for (Point p : hull) {
        cout << "(" << p.x << ", " << p.y << ")" << endl;
    }
```

```
    return 0;
}
```

- Minh họa kết quả:

```
PS C:\Users\admin\Documents\Toan_ung_dung_cntt\Vector> g++ .\Bai1.cpp
PS C:\Users\admin\Documents\Toan_ung_dung_cntt\Vector> .\a.exe
(-4, 2)
(-3, -2)
(-1, -4)
(1, -4)
(3, -4)
(5, -2)
(-1, 4)
PS C:\Users\admin\Documents\Toan_ung_dung_cntt\Vector>
```

c) Vẽ bao lồi



Tính diện tích bao lồi:

```
double dienTichBaoLoi(vector<Point> hull) {
    if (hull.size() < 3) return 0;

    double area = 0.0;
    for (int i = 0; i < hull.size(); i++) {
        int next = (i + 1) % hull.size();;
        area += (hull[i].x * hull[next].y) - (hull[next].x * hull[i].y);
```

```
    }
    return abs(area) / 2.0;
}
```

Kết quả: 43

Bài 2.

a) Mô tả thuật toán:

Bước 1: Chia n điểm thành hai nửa bằng cách sắp xếp chúng theo tọa độ x. Điểm giữa sẽ được chọn làm điểm phân chia

Bước 2: Gọi một hàm đệ quy để tìm khoảng cách nhỏ nhất trong nửa bên trái của điểm (từ đầu đến điểm giữa) và một hàm đệ quy khác để tìm khoảng cách nhỏ nhất trong nửa bên phải (từ điểm giữa đến cuối). Lưu lại khoảng cách nhỏ nhất tìm được từ nửa trái và nửa phải

Bước 3: Tính khoảng cách giữa các điểm gần biên

+ Tạo một danh sách các điểm nằm trong khoảng cách nhỏ hơn một giá trị minDistance (khoảng cách nhỏ nhất đã tìm được) từ đường phân chia. Điều này có nghĩa là chỉ giữ lại các điểm mà tọa độ x của chúng nằm trong khoảng giới hạn nhất định quanh điểm giữa

Bước 4: Kiểm tra khoảng cách trong danh sách gần biên

+ Duyệt qua từng cặp điểm trong danh sách đã tạo ra ở bước trước. Tính khoảng cách giữa các cặp điểm này và so sánh với khoảng cách nhỏ nhất đã lưu

+ Chỉ kiểm tra các điểm mà sự chênh lệch về tọa độ y giữa chúng là nhỏ hơn khoảng cách nhỏ nhất đã lưu

Bước 5: Nếu khoảng cách giữa bất kỳ cặp điểm nào trong danh sách này nhỏ hơn khoảng cách nhỏ nhất đã lưu, hãy cập nhật giá trị khoảng cách nhỏ nhất.

Thuật toán này sử dụng phương pháp chia để trị để giảm thiểu số phép tính cần thiết. Bằng cách tận dụng đặc điểm của khoảng cách Manhattan và chỉ kiểm tra các điểm gần đường phân chia, thuật toán có độ phức tạp thời gian là O(n log n)

b) Cài đặt thuật toán:

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

using namespace std;

struct Point {
```

```
    int x;
    int y;
    int z;

    Point(int x, int y, int z) : x(x), y(y), z(z) {}
    bool operator<(const Point& other) const {
        return x < other.x || (x == other.x && y < other.y) || (x == other.x
&& y == other.y && z < other.z);
    }
};

int khoangCachManhattan(const Point& a, const Point& b) {
    return abs(a.x - b.x) + abs(a.y - b.y) + abs(a.z - b.z);
}

int khoangCachNhoNhatTraiVaPhai(vector<Point> &points, int left, int right) {
    if (right - left <= 1) {
        return INT_MAX;
    }

    int mid = left + (right - left) / 2;
    int minLeft = khoangCachNhoNhatTraiVaPhai(points, left, mid);
    int minRight = khoangCachNhoNhatTraiVaPhai(points, mid, right);

    int minDistance = min(minLeft, minRight);

    vector<Point> strip;
    for (int i = left; i < right; i++) {
        if (abs(points[i].x - points[mid].x) < minDistance) {
            strip.push_back(points[i]);
        }
    }

    for (int i = 0; i < strip.size(); i++) {
        for (int j = i + 1; j < strip.size(); j++) {
            if (abs(strip[j].y - strip[i].y) >= minDistance) {
                break;
            }

            minDistance = min(minDistance, khoangCachManhattan(strip[i],
strip[j]));
        }
    }

    return minDistance;
}

int khoangCachNhoNhat(vector<Point> &points) {
    sort(points.begin(), points.end());

    return khoangCachNhoNhatTraiVaPhai(points, 0, points.size());
```

```
}

int main() {
    vector<Point> points = {
        Point(1, 2, 3),
        Point(4, 5, 6),
        Point(7, 8, 9),
        Point(1, 1, 1),
        Point(7, 2, 3),
        Point(2, 2, 2),
        Point(3, 3, 3),
    };
    cout << khoangCachNhoNhat(points) << endl;

    return 0;
}
```

c) Kết quả: 2