Họ tên: Nguyễn Trung Dũng

MSSV: 19120486

Bài 1:

- Thao tác cơ sở: `a[pivot] ⇆ a[i]`
- Gọi $T(n)$ là số phép tính cần thực hiện với mảng n phần tử, ta có:

$$T(0) = 0, T(1) = 2, \dots$$

Mỗi vòng lặp thì thực hiện thao tác cơ sở 2 lần và thực hiện $T(n-1)$ 1 lần, lặp n lần

$$\Rightarrow T(n) = nT(n-1) + 2n$$

- Dùng hàm sinh, ta tìm được nghiệm của hệ thức đệ quy trên:

$$\begin{aligned} T(n) &= 2.n! \sum_{i=0}^{n-1} \frac{1}{i!} \\ &= 2.n! \left(\frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{(n-1)!} \right) \\ &= 2\lfloor \mathrm{n!\, e} - 1 \rfloor \approx \Theta(\mathrm{n!}) \end{aligned}$$

Bài 2:

- Giải thuật tìm dạng thức liên kết:

```
// kiểm tra dạng fully connected mesh, tất cả các đỉnh phải có bậc = n-1
checkMesh(g[1..n][1..n]) {
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            if (i != j && g[i][j] == false)
                return false;
        }
    }
    return true;
}
// kiểm tra dạng star, duy nhất 1 đỉnh có bậc n-1, tất cả các đỉnh còn lại bậc 1
checkStar(g[1..n][1..n]) {
    if (n == 1)
        return true;
    countHub = 0;
    for (i = 1; i <= n; i++) {
        deg = 0;                        // tính bậc của đỉnh i
        for (j = 1; j <= n; j++) {
            if (g[i][j] == true)
                deg++;
        }
        if (deg != 1 && deg != n-1)  // nếu tồn tại bậc khác 1 và khác n-1 thì
            return false;            // trái với điều kiện

        if (deg == n-1)              // đếm số lượng đỉnh có bậc n-1
            countHub++;
    }
    if (countHub != 1)               // nếu số lượng đỉnh bậc n-1 khác 1 thì trái
        return false;                // với điều kiện
    return true;
}
// kiểm tra dạng vòng, g chứa ít nhất 3 đỉnh, phải liên thông và tất cả các đỉnh
có bậc 2
checkRing(g[1..n][1..n]) {
    if (n < 3)
        return false;
    v[] = false for (i from 1 to n);
    duyet(g, v);                 // duyệt qua g 1 lần, có thể dùng bfs hoặc dfs

    for (i = 1; i <= n; i++)
        if (v[i] == false)       // nếu có đỉnh chưa duyệt thì g không liên thông
            return false;

    for (i = 1; i <= n; i++) {
        deg = 0;                 // tính bậc của đỉnh i
        for (j = 1; j <= n; j++) {
            if (g[i][j] == true)
                deg++;
        }
        if (deg != 2)            // nếu tồn tại đỉnh bậc khác 2 thì trái với điều
            return false;        // kiện
    }
    return true;
}
```

```
if (checkMesh(g))
    print(“G có dạng lưới đầy đủ”);
if (checkStar(g))
    print(“G có dạng sao”);
if (checkRing(g))
    print(“G có dạng vòng”);
```

- Các thao tác cơ sở đã được tô màu đỏ. Cả 3 thuật toán kiểm tra đều có độ phức tạp $\Theta(n^2)$

Bài 3:

- Giải thuật tìm 2 tập có tổng bằng nhau
- Ý tưởng: chọn bằng cách phát sinh dãy nhị phân, nếu một trạng thái đã thỏa điều kiện có tổng bằng một nửa tổng tất cả phần tử thì in ra và kết thúc chương trình.
- Mã giả:

```
partition(cur, a[1..n], list[]) {
    // nếu con trỏ cur nằm ngoài khoảng cần tính hoặc tổng các phần tử trong list
    lớn hơn sum(a)/2
    // trả về false
    if (cur > n+1 || sum(list) > sum(a)/2)
          return false;

    // nếu tổng trong list = tổng các số còn lại
    // in ra, trả về true và kết thúc chương trình
    if (sum(list) == sum(a)/2) {
        print(list);        // in các số trong list
        print(a \ list);    // in các số còn lại
        return true;
    }

    // nếu con trỏ cur chưa chạy hết mảng: 2 trường hợp:
    // chọn a[cur] hoặc không chọn a[cur] và tiếp tục xét các phần tử còn lại
    else if (cur <= n) {
        // chọn a[cur] và tiếp tục xét; khi xét các phần tử còn lại mà tìm thấy
        1 tập list phù hợp thì trả về true và kết thúc chương trình (không cần
        xét trường hợp không chọn a[cur])
        list.push(a[cur]);
        if (partition(cur + 1, a, list))
              return true;

        // loại phần tử a[cur] và tiếp tục xét
        list.pop();
        return partition(cur + 1, a, list);
    }

    // nếu cur = n+1 (kết thúc việc xét) nhưng sum(list) != sum(a)/2
    return false;
}
// khởi tạo list rỗng
list[] = none;
// kiểm tra tổng lẻ hoặc chia không được (nếu tổng chẵn và chia được thì chương
trình sẽ in ra 2 tập)
if (sum(a) mod 2 == 1 || !partition(1, a, list))
    print("Không có cách chia");
```

Bài 4:

a. Tổng của tất cả phần tử của ma trận:

$$S = 1 + 2 + 3 + \cdots + n^2$$

$$= \frac{n^2(n^2+1)}{2}$$

Mà mỗi dòng (cột) có tổng là t

$$\Rightarrow S = nt$$

$$\Rightarrow nt = \frac{n^2(n^2+1)}{2}$$

$$\Leftrightarrow \mathrm{t} = \frac{n(n^2+1)}{2}$$

b. Giải thuật phát sinh ma phương bậc n:

```
magicSquare(pivot, a[1..n][1..n]) {
    // nếu đã xét hết các phần tử trong một hoán vị thì kiểm tra nó có phải là
    một ma phương hay không
    // nếu phải thì in
    if (pivot == n^2)
        if (kiemTra(a)) // kiểm tra tổng các dòng/cột và 2 đường chéo bằng nhau
            print(a);

    // nếu chưa xét hết thì tiếp tục xét, lấy ý tưởng từ hàm Taohoanvi()
    // chuyển đổi pivot và i từ tọa độ logic sang tọa độ 2d
    else {
        pRow = (pivot - 1) div n + 1;
        pCol = (pivot - 1) mod n + 1;
        for(i = pivot; i <= n^2; i++){
            row = (i - 1) div n + 1;
            col = (i - 1) mod n + 1;

            a[pRow][pCol] ⇆ a[row][col];
            magicSquare(pivot + 1, a);
            a[pRow][pCol] ⇆ a[row][col];
        }
    }
}
// khởi tạo table a chứa nxn phần tử từ 1 tới n^2 theo thứ tự
a[][] = i for (i from 1 to n^2)
magicSquare(1, a)
```