Bài tập 1:

1.

```
class People {
        public:
        int age;
        string first_name, last_name;
};
bool compare_tuoi(People a, People b)
        return (a.age < b.age);
bool compare_ten(People a, People b)
        return (a.first_name > b.first_name);
bool compare_ho(People a, People b)
        return (a. last_name < b. last_name);
vector <People> v(n);
stable_sort(v.begin(), v.end(), compare_ho);
stable_sort(v.begin(), v.end(), compare_ten);
stable_sort(v.begin(), v.end(), compare_tuoi);
```

2.

```
class Fraction {
public:
   int tu;
   int mau;
```

```
};
void sort_fraction_list(vector<Fraction*> &v) {
	int n = v.size();
	vector <pair<double, Fraction*>> temp(n);
	// double d = v[i][j]->tu * 1.0 / v[i][j]->mau;
	// temp[i][j] = make_pair(d, v[i][j]);
	_assign1(temp, v);
	sort(temp.begin(), temp.end());
	// v[i][j] = temp[i][j].second;
	_assign2(v, temp);
}
```

Bài tập 2:

Trường hợp QuickSort thực thi với O(n^2):

Tất cả phần tử đều bằng nhau.

Phương án giải quyết:

Thêm điều kiện “Nếu mảng đồng nhất, xem như mảng đã sắp xếp”.

Bài tập 3:

```
vector <int> S(n);
vector <int> sorted_S(n);
sorted_S.assign(S.begin(), S.end()); // O(n)
sort(sorted_S.begin(), sorted_S.end()); // O(nlogn)
```

1.

```
// Tìm sorted_S // O(nlogn)
cout << sorted_S[0];
for (int i=1; i<n; i++) // n steps
        if (sorted_S[i] != sorted_S[i-1])
                cout << sorted_S[i];
```

Time complexity: O(nlogn + n) = O(nlogn)

2.

```
void BT3_2(int &a, int &b) {
        for (int i=0; i<n; i++) { //  <= n steps
                for (int j=i; j<n; j++) { //  <= n-i steps
                        int64_t tmp = S[i] + S[j];
                        if (tmp == v) {
                                a = S[i];
                                b = S[j];
                                return;
}}}
        cout << “Khong co a, b”;
}
```

Time complexity: O(n + n-1 + … + 1) = O(n^2)

3.

```
void BT3_3(int &a, int &b) {
        for (int i=0; i<n; i++) { //  <= n steps
                for (int j=i; j<n; j++) { //  <= n-i steps
                        int64_t tmp = S[i] + S[j];
                        if (tmp > v) {
                                cout << “Khong co a, b”;
                                return;
                        }
                        if (tmp == S) {
                                a = S[i];
                                b = S[j];
                                return;
}}}
        cout << “Khong co a, b”;
}
```

Time complexity: O(n + n-1 + … + 1) = O(n^2)

4.

```
// Tìm sorted_S // O(nlogn)
int i;
for (i=0; i<n; i++) // <= n steps
        if (sorted_S[i] >= a)
```

```
            break;
for (; i<n; i++) { <= n steps
        if (sorted_S[i]  <= b)
            cout << sorted_S[i];
        else
            break;
}
```

Time complexity: O(nlogn + n + n) = O(nlogn)

5.

```
// Tìm sorted_S // O(nlogn)
int ans = sorted_S[(n-1)/2];
```

Time complexity: O(nlogn)

6.

```
// Tìm sorted_S // O(nlogn)
int ans, dem=1;
ans = sorted_S[0];
for (int i=1; i<n; i++) { // <= n steps
        if (sorted_S[i] == sorted_S[i-1])
            dem++;
        else
            dem = 0;
```

```
        if (dem > n/2) {
                ans = sorted_S[i];
                break;
}
```
Time complexity: O(nlogn + n) = O(nlogn)

Bài tập 4:

1.

```
void clock_rotate(vector<vector<int>> &v) {
   int n = v.size();
for (int i=0; i<n-1-i; i++) // (n-1)/2 steps
   v[i].swap(v[n-1-i]); // O(C)
Transpose(v); // O(n^2)
}
```
Time complexity: O(n^2 + C*n) = O(n^2)

2.

```
void Transpose(vector<vector<int>> &v) {
   int n = v.size();
for (int i=1; i<n; i++) // n-1 steps
       for (int j=0; j<i; j++) // i steps
           swap(v[i][j], v[j][i]);
}
```

Time complexity: $O(1 + 2 + \ldots + n-1) = O(n^2)$