1.

```cpp
#include <iostream>
#include <vector>
using namespace std;
const int N = 12;
//정렬할 배열
vector<int> arr;
void bubbleSort(int n) {
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        bool flag = true;

        for (int j = 1; j < n - i; j++) {
            if (arr[j - 1] > arr[j]) {
                flag = false;
                swap(arr[j - 1], arr[j]);
            }
            cnt++;
        }

        if (flag)
            break;
    }
}
int main() {
    arr = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };

    bubbleSort(N);
    for (int i = 0; i < N; i++)
        cout << arr[i] << " " ;
}
```

2.

2-1) dp풀이

```
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int>> dp;
bool isInterleaving(string A,string B,string C) {
	int len_a = A.length(), len_b = B.length();
	//가장자리 초기화
	dp[0][0] = true;
	for (int i = 1; i <= len_a; i++)
		dp[i][0] = (A[i - 1] == C[i - 1]) ? dp[i - 1][0] : false;
	for (int i = 1; i <= len_b; i++)
		dp[0][i] = (B[i - 1] == C[i - 1]) ? dp[0][i - 1] : false;
	for (int i = 1; i <= len_a; i++) {
		for (int j = 1; j <= len_b; j++) {
			char curA = A[i - 1], curB = B[j - 1], curC = C[i + j - 1];
			if (curA == curC && curB != curC)
				dp[i][j] = dp[i - 1][j];
			else  if (curA != curC && curB == curC)
				dp[i][j] = dp[i][j - 1];
			else if (curA == curC && curB == curC)
				dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
			else // 둘 다 다를때
				dp[i][j] = false;

		}
	}
	return dp[len_a][len_b];
}
/*
dp[i][j] : 해당 길이까지의 두 부분 문자열(~A[i],~B[j])로 C를 만들 수 있는지 없는지 저장
핵심 ; A의 i번째 문자와 B의 j번째 문자와 C의 i + j - 1번째 문자를 비교한다.
1. A와 같다면 위쪽칸 dp[i-1][j]의 결과를 가져온다.
 왜냐하면, 윗쪽 칸은 lhs의 i번째 문자가 없을 때의 결과를 의미하는데 여기에 i번째 문자를 추가해준 것과 같기 때문에 결과를 그대로 따라가기 때문
2. B와 같다면 왼쪽칸 dp[i][j-1]의 결과를 가져온다.
B의 j번째 문자를 추가해준 것과 같기 때문이다.
3. A와 B와 모두 같다면
해당 문자가 A에서 왔는지 B에서 왔는지 알 수 없다.
따라서 둘 (위쪽, 왼쪽) 중 하나라도 true가 있다면 그대로 가져온다.
4. 둘다 같지 않다
해당 부분 문자열은 A[i]와 B[j]까지 의 조합으로는 아직 만들 수 없기 때문에 false처리한다.
*/
int main() {

		string A, B, C;
		cin >> A >> B >> C;
		int len_a = A.length(), len_b = B.length();
		dp.assign(len_a+1, vector<int>(len_b+1, 0));
		cout << isInterleaving(A, B, C);

}
```

2-2) 재귀 풀이

```
#include<iostream>
#include <vector>
#include<algorithm>
using namespace std;
string a;
string b;
string c;
bool dp[201][201];
bool isInterleaving(int x, int y)
{
	if (dp[x][y])
		return false;
	dp[x][y] = true;
	if (c[x + y] == '\0')
		return true;
	if (c[x + y] == a[x])
	{
		if (isInterleaving(x + 1, y))
			return true;
	}
	if (c[x + y] == b[y])
	{
		if (isInterleaving(x, y + 1))
			return true;
	}
	return false;
}
int main()
{
	cin >> a >> b >> c;
	cout << isInterleaving(0, 0);
}
```

3.

3-1)dp

```
#include <iostream>
#include <vector>
using namespace std;
# define INF 123456789
const int n = 6;
/*
dp[i][j] :  i 까지 j 번을 경유해서 갔을 경우 최소비용저장
```

```
*/
int getMin(vector<int> v) {
	int min=INF;
	for (int i = 0; i < v.size(); i++) {
		if (min > v[i])
			min = v[i]; // 최솟값 갱신
	}
	return min;
}
int main() {
	vector<int> air; // 항공편 비용
	vector<int> hotel; //숙박 비용
	vector<vector<int>> dp(n+1, vector<int>(n+1, INF));
	air = {0,1,3,6,11,17 };
	hotel = {0,2,5,1,5,0};
	for (int i = 0; i < n; i++) {
		dp[i][1] = air[i] + hotel[i]; // 한번에 이동하는 방법 초기화
	}
	for (int i = 1; i < n; i++) { // i : 도착지
		for (int j = 1; j <= i; j++) { // j : 경유 횟수
			cout << dp[i][j] << " ";
		}
		cout << "\n";
	}
	for (int i = 1; i < n; i++) { // i : 도착지
		for (int j = 1; j <= i; j++) { // j : 경유 횟수

			int min_value = INF;
			for (int k = 1; k <= i - 1; k++) { //k : 경유지
				// k까지 가는 경로 중 최솟값

				int cost = getMin(dp[k]) + air[i - k] + hotel[i];
				cout << cost<<"\n";
				min_value = min(min_value, cost);
			}
			dp[i][j] = min_value;
		}
	}
	cout << getMin(dp[n]);


}
```