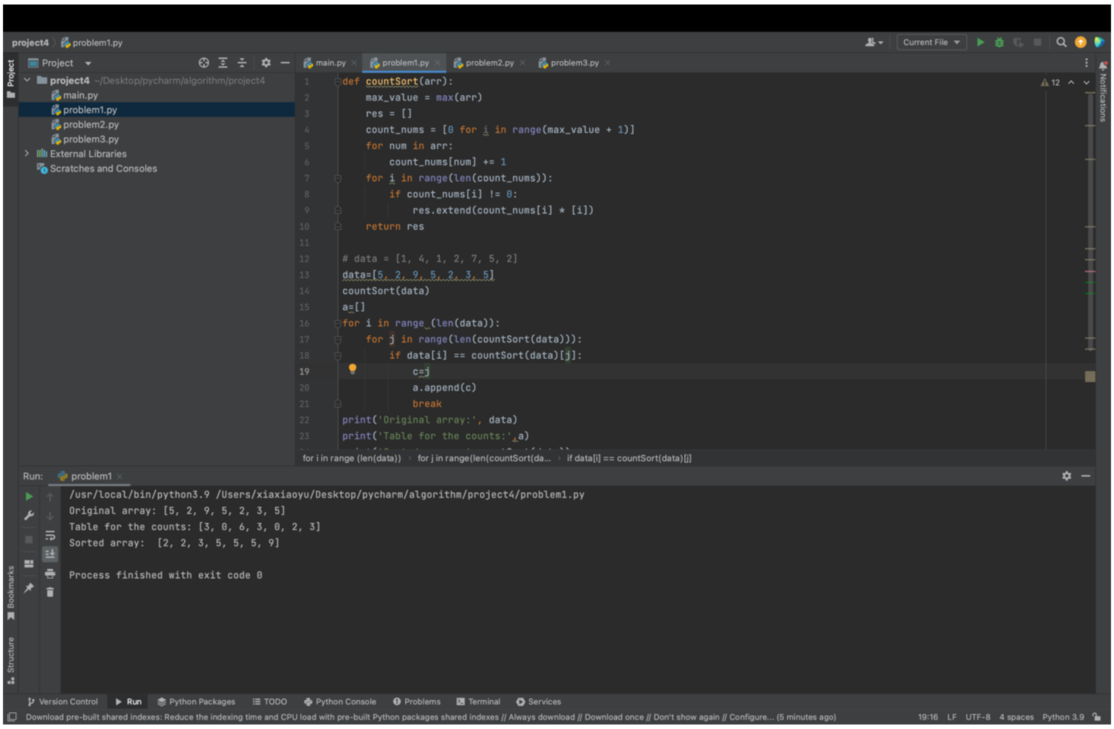
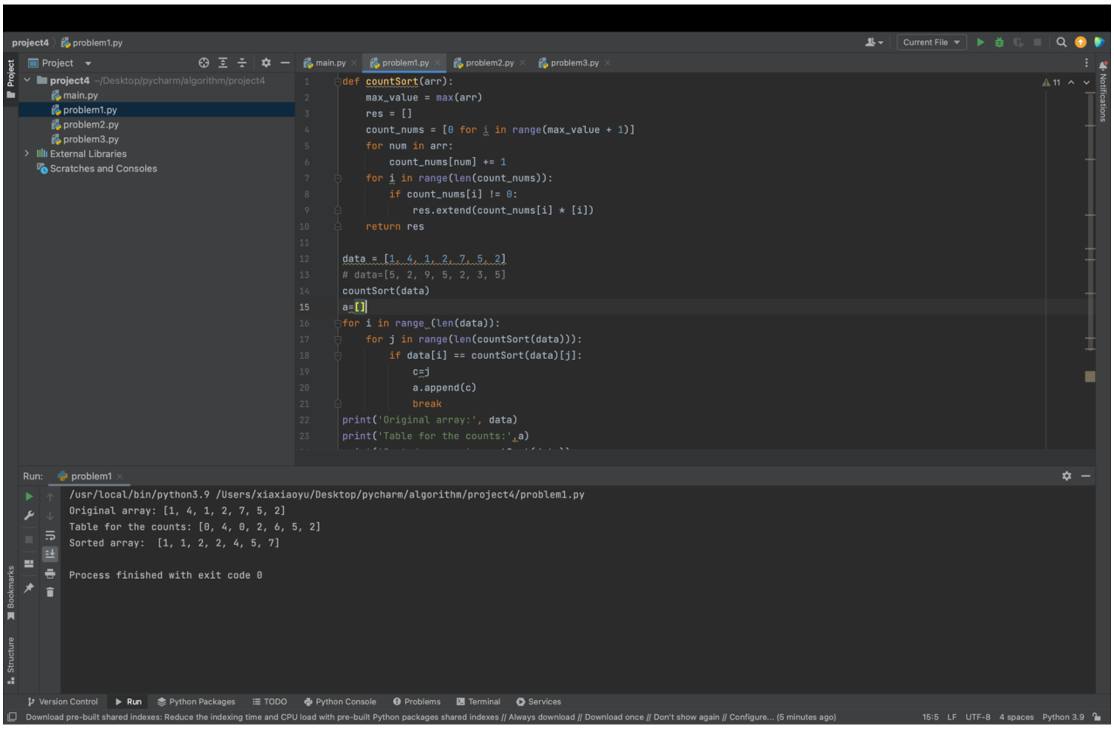
# Problem 1:

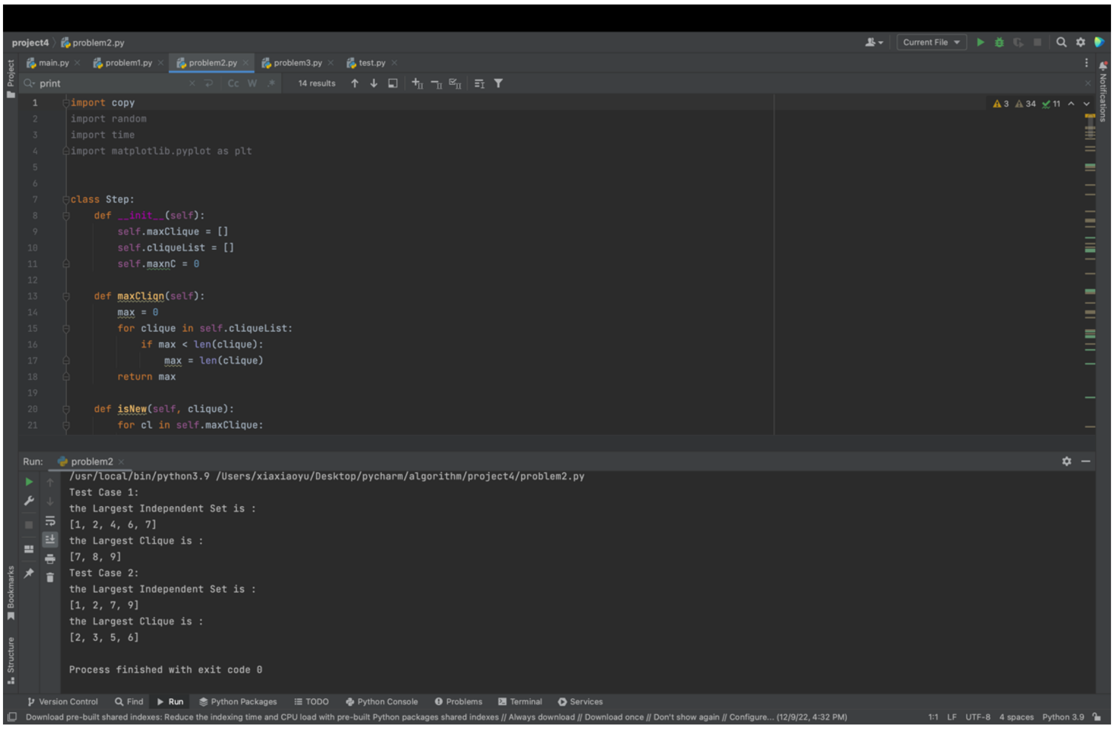
## Case1:




## Case2:



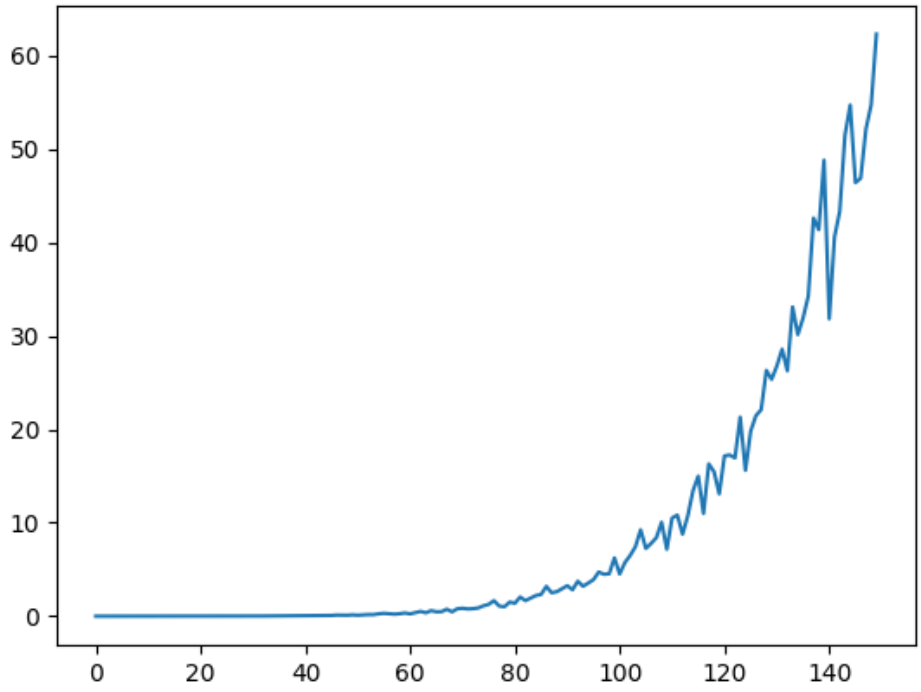
# Problem2:




## Time complexity for both algorithms : $n^4$

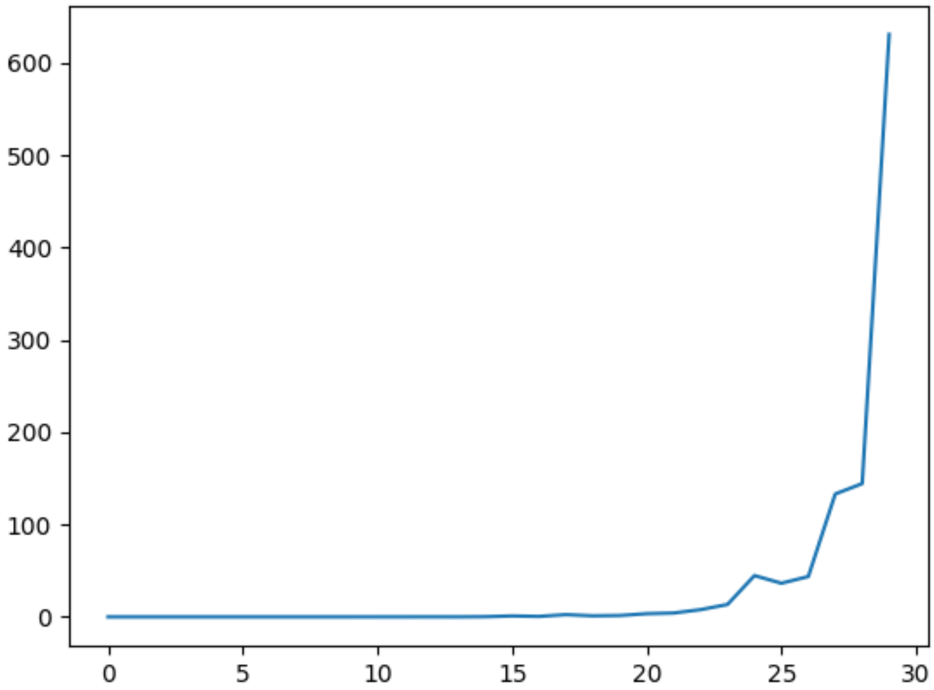
## The largest independent set:

```python
def process(self):

    n = len(self.V)
    solutions = {}
    for i in range(0, n):
        solutions[i] = Step()
    for v in self.V:
        a = []
        a.append(v)
        solutions[0].cliqueList.append(a)
        solutions[0].updateMaxClique()

    for i in range(1, n):
        # cliqList= solutions[i-1].maxClique
        preData = solutions[i - 1]
        cliqList = preData.maxClique
        preMax = preData.maxnC
        for clique in cliqList:
            for v in self.V:
                tempclique = copy.deepcopy(clique)
                if not v in tempclique:
                    # if s.isConnectedAll(tempclique,v):
                    if self.isConnectedOne(tempclique, v):
                        tempclique.append(v)
                        solutions[i].cliqueList.append(tempclique)
        solutions[i].updateMaxClique()
        if not len(solutions[i].maxClique):
            break
```



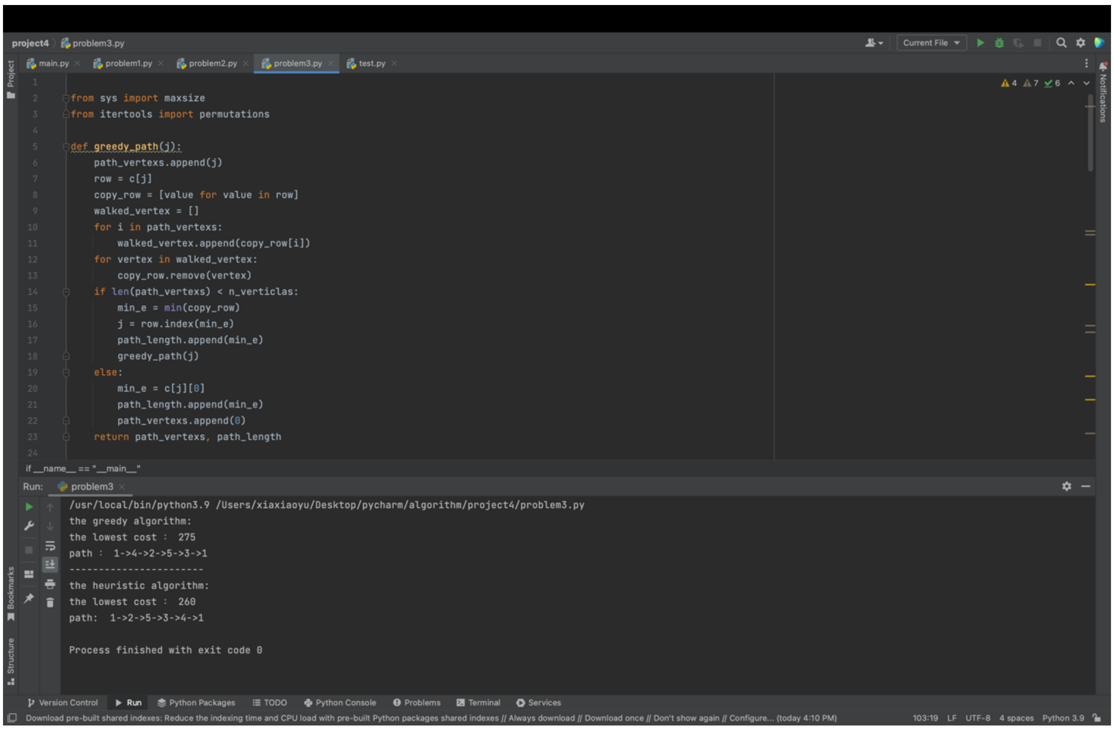

## The largest clique:

```
def process(self):
    n = len(self.V)
    solutions = {}
    for i in range(0, n):
        solutions[i] = Step()
    for v in self.V:
        a = []
        a.append(v)
        solutions[0].cliqueList.append(a)
        solutions[0].updateMaxClique()

    for i in range(1, n):
        # cliqList= solutions[i-1].maxClique
        preData = solutions[i - 1]
        cliqList = preData.maxClique
        preMax = preData.maxnC
        for clique in cliqList:
            for v in self.V:
                tempclique = copy.deepcopy(clique)
                if not v in tempclique:
                    if self.isConnectedAll(tempclique, v):
                        tempclique.append(v)
                        solutions[i].cliqueList.append(tempclique)
        solutions[i].updateMaxClique()
        if not len(solutions[i].maxClique):
            break
```

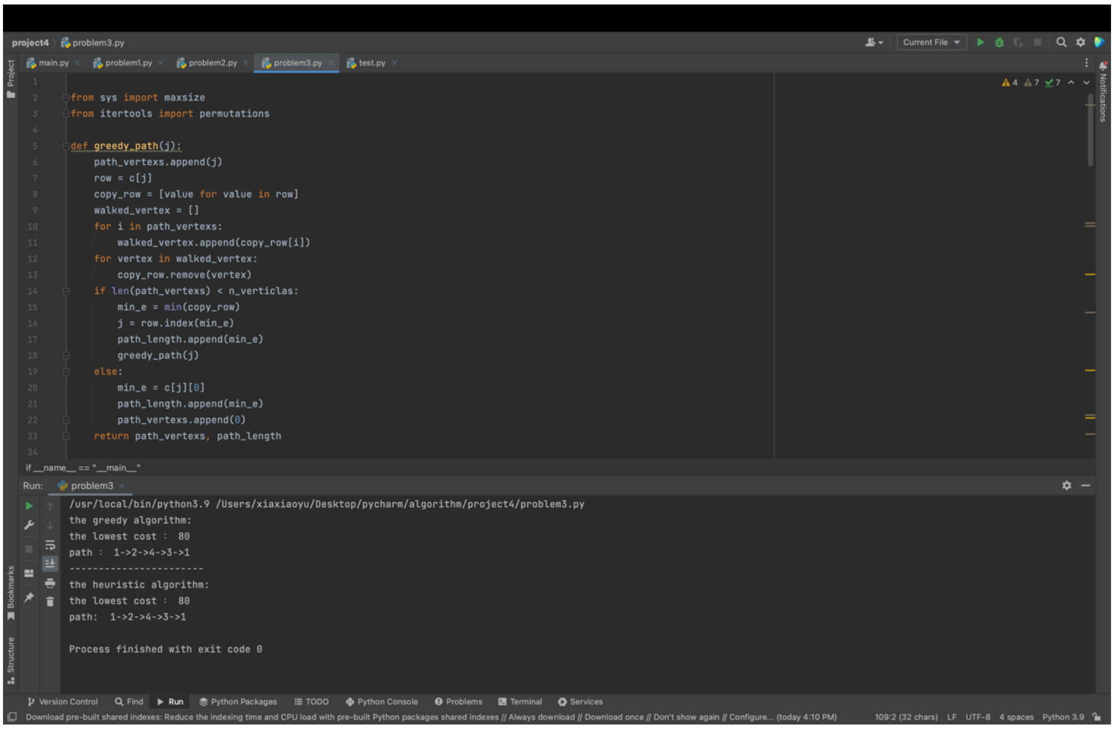



# Problem3:

## Case1:



## Case2:




**compare** the results of A and B and discuss how good the results of B are by calculating: in the first test case, the calculating result is 260 / 275 = 0.945. In the second test case, the result is 80 / 80 = 1. There is a difference in the results in the first test case because the TSP problem is an NP-hard problem so we cannot promise that every different algorithm, which is used in the same graph, could get the same answer.

**the time complexity of each using Big O：**

greedy algorithm: $O(n^2)$:

```python
def greedy_path(j):
    path_vertexs.append(j)
    row = c[j]
    copy_row = [value for value in row]
    walked_vertex = []
    for i in path_vertexs:
        walked_vertex.append(copy_row[i])
    for vertex in walked_vertex:
        copy_row.remove(vertex)
    if len(path_vertexs) < n_verticlas:
        min_e = min(copy_row)
        j = row.index(min_e)
        path_length.append(min_e)
        greedy_path(j)
    else:
        min_e = c[j][0]
        path_length.append(min_e)
        path_vertexs.append(0)
    return path_vertexs, path_length
```

A heuristic algorithm: $O(n^2)$:

```python
def heuristic(graph, s):
    # store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    min_path = maxsize
    next_permutation = permutations(vertex)
    for i in next_permutation:
        # store current Path weight(cost)
        current_pathweight = 0
        # compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        # update minimum
        min_path = min(min_path, current_pathweight)
    return min_path
```