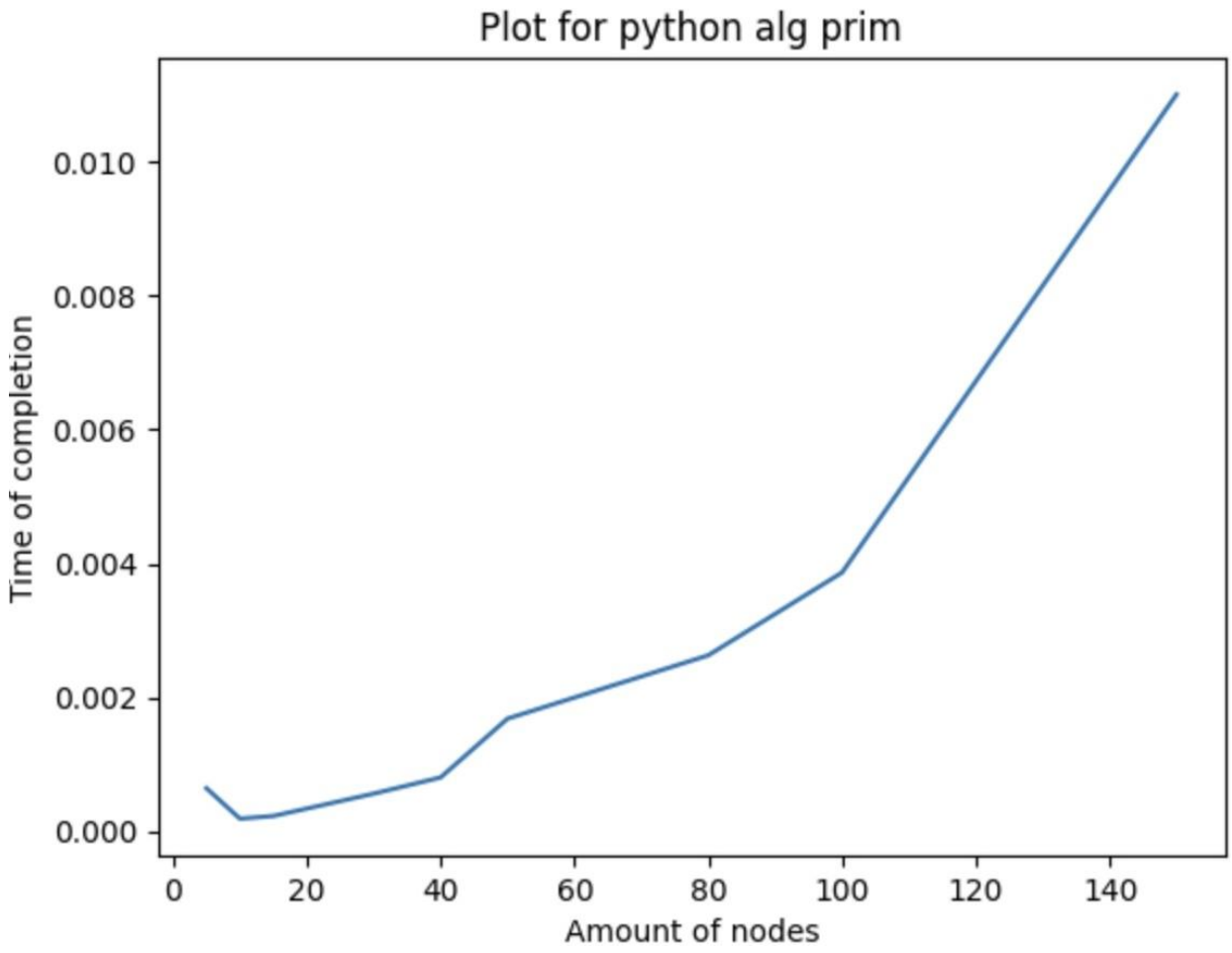
Дослідження роботи алгоритма Краскала та Пріма

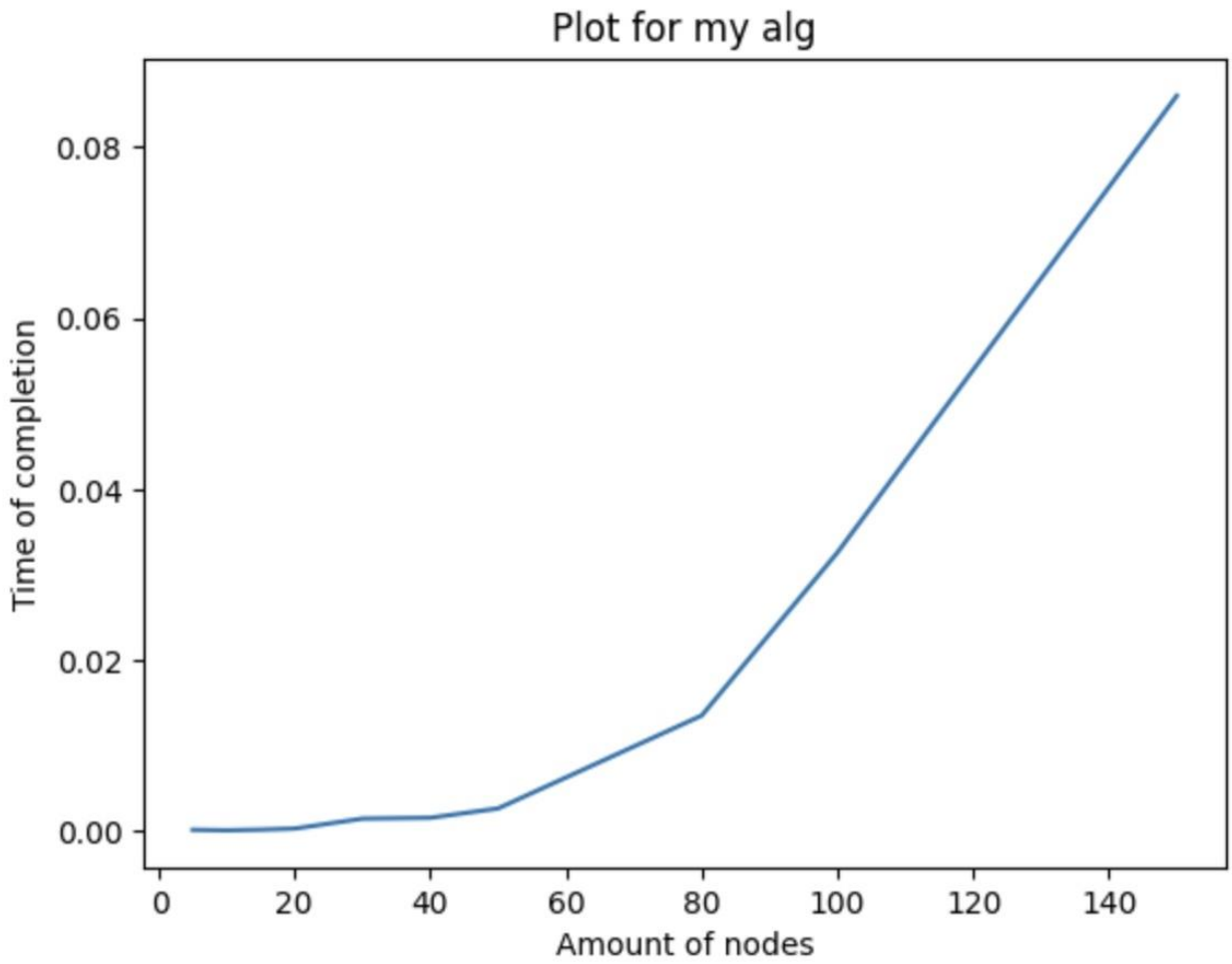
Шевчук Іван, Шинкаренко Іван

Наша задача була написати алгоритм Краскала та Флойда та порівняти з вбудованим модулем пайтона

1. Краскала

```python
def kruskal(graph):

    sets = [{node} for node in graph.nodes()]

    line_list = []
    weight = 0

    sorted_edges = sorted(graph.edges(data=True), key=lambda x: x[2]['weight'])

    for (u, v, w) in sorted_edges:
        u_set = next(s for s in sets if u in s)
        v_set = next(s for s in sets if v in s)

        if u_set != v_set:
            line_list.append((u, v))
            weight += w['weight']

            u_set.update(v_set)
            sets.remove(v_set)
    print(line_list, weight)
    return line_list, weight
```

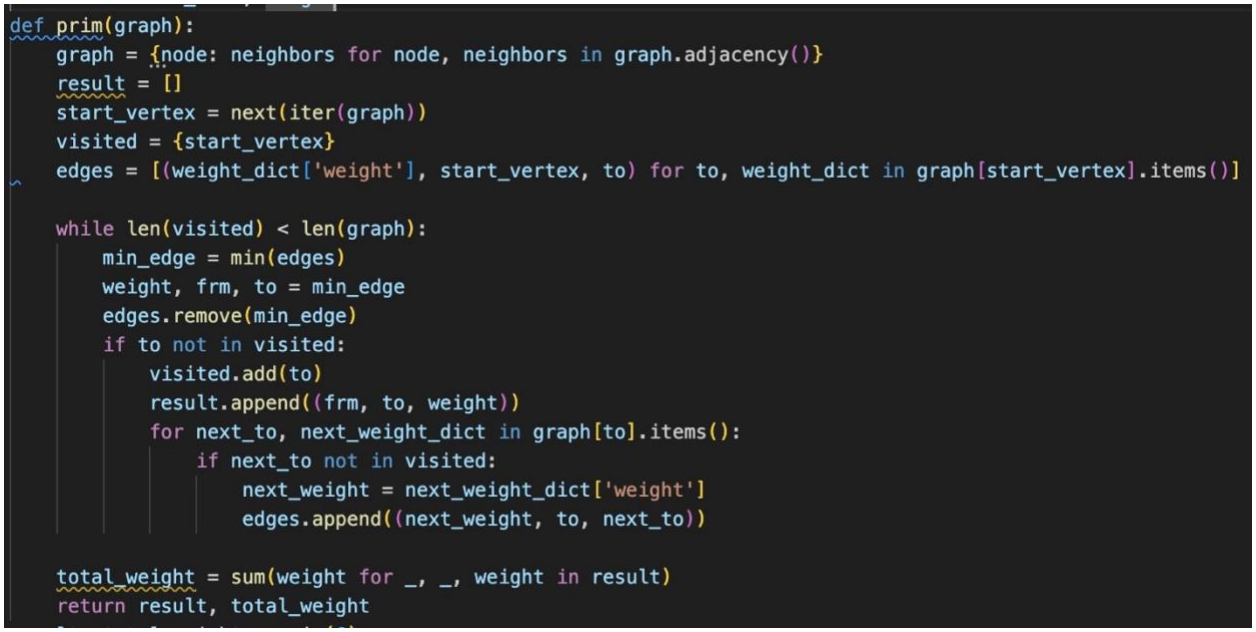

Plot for python alg prim
Time of completion
0.010
0.008
0.006
0.004
0.002
0.000
0
20
40
60
80
100
120
140
Amount of nodes



Суть алгоритму полягає у сортуванні ребер на початку, береться завжди ребро з найменшою вагою, перевіряється чи додавання ребра не спричинить додавання циклу, якщо ні то додаєм ребро у список ребер нашого каркасу

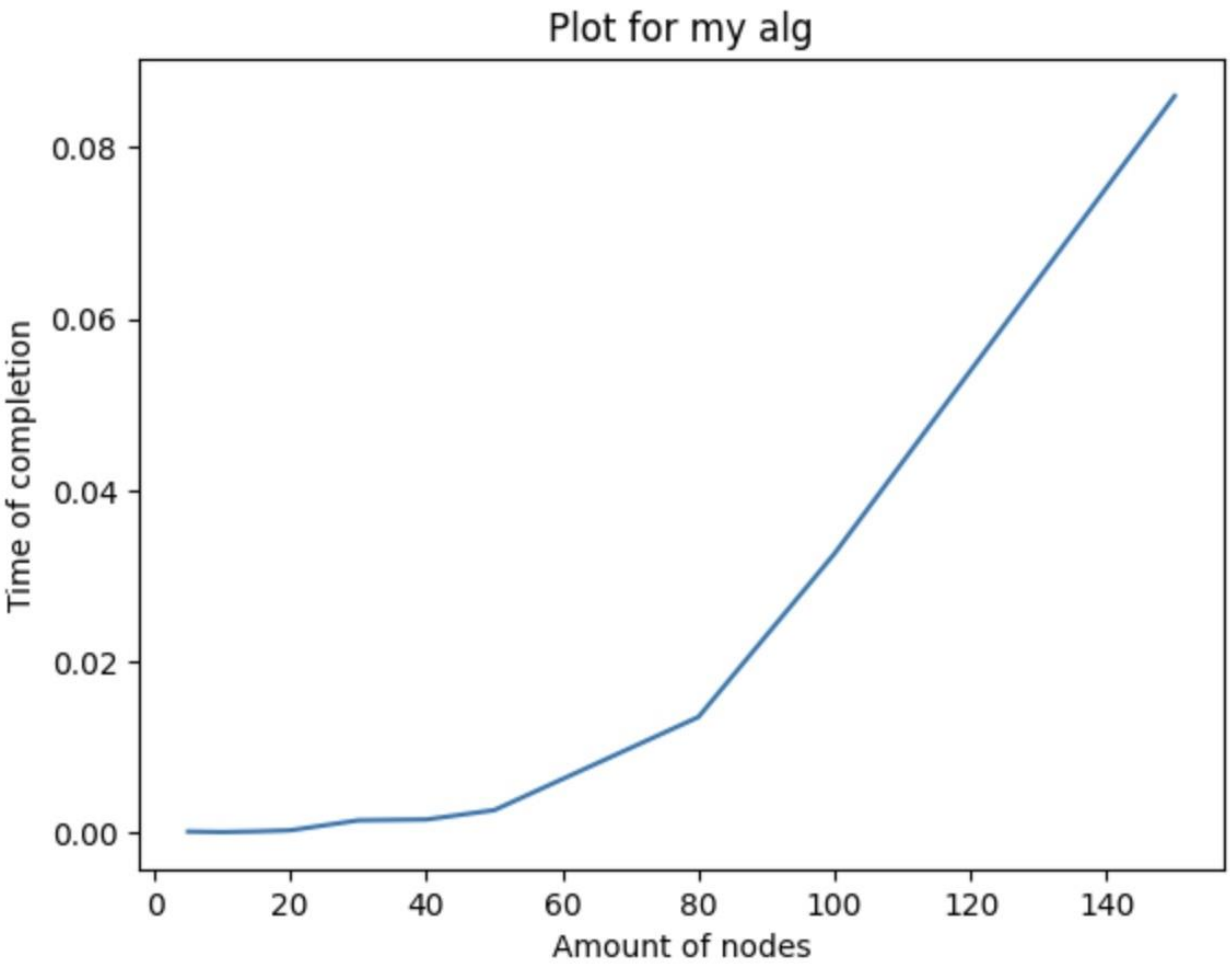
По графіку видно що наш алгоритм краще справляється з графіками з більшою кількістю ребер ніж вбудований

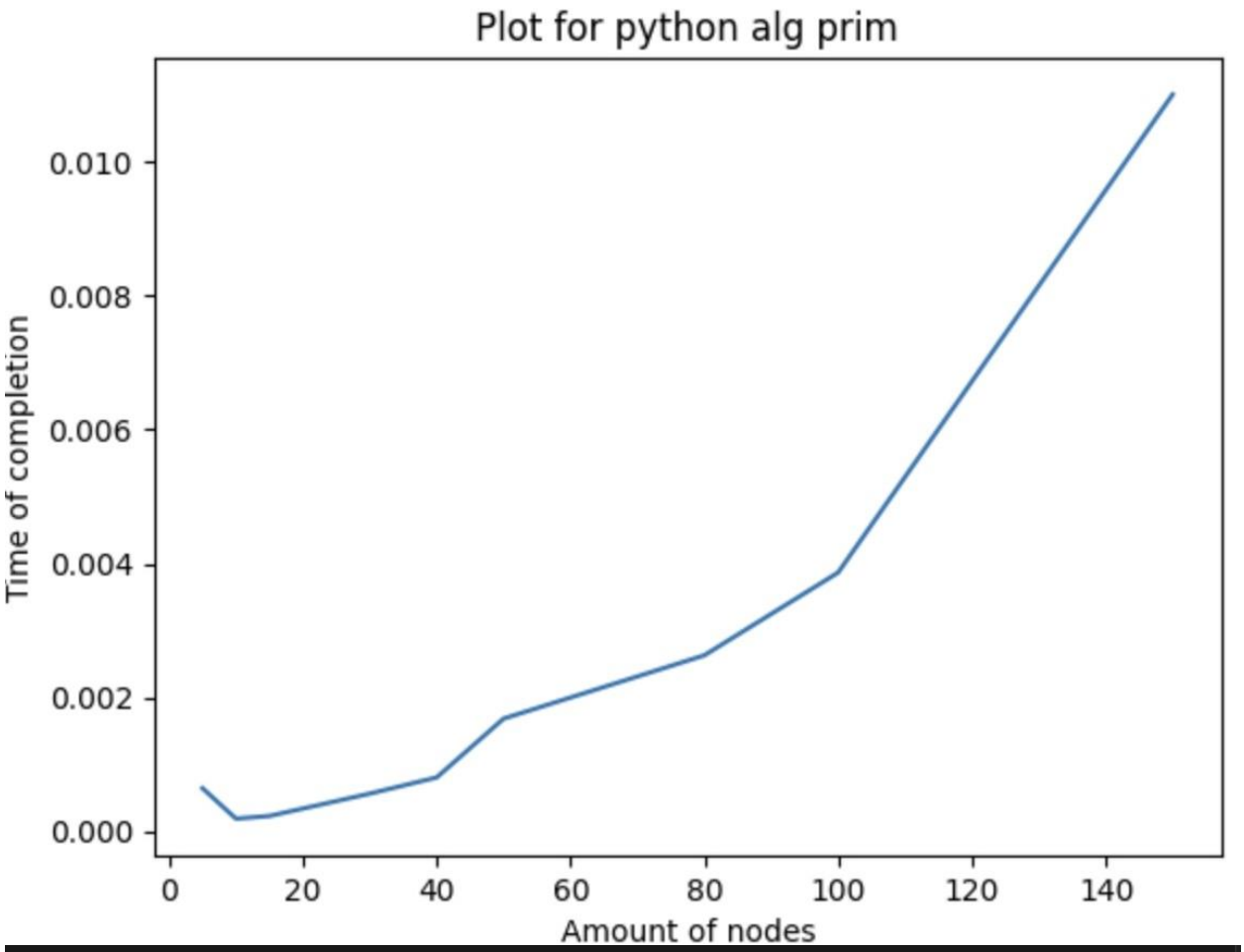
2.Прима



```python
def prim(graph):
    graph = {node: neighbors for node, neighbors in graph.adjacency()}
    result = []
    start_vertex = next(iter(graph))
    visited = {start_vertex}
    edges = [(weight_dict['weight'], start_vertex, to) for to, weight_dict in graph[start_vertex].items()]

    while len(visited) < len(graph):
        min_edge = min(edges)
        weight, frm, to = min_edge
        edges.remove(min_edge)
        if to not in visited:
            visited.add(to)
            result.append((frm, to, weight))
            for next_to, next_weight_dict in graph[to].items():
                if next_to not in visited:
                    next_weight = next_weight_dict['weight']
                    edges.append((next_weight, to, next_to))

    total_weight = sum(weight for _, _, weight in result)
    return result, total_weight
```

Алгоритм прима вимагає початкову задану вершину, в даному коді я вибираю першу веришу, у visited додаються вершини в які вже зайшли, саме від цих вершин ми в подальшому зможем додавати ребра. Також перевіряється чи немає вершини в яку ми йдемо уже в списку пройдених вершин, якщо є, берем інше ребро з найменшою вагою.


Plot for my alg
Time of completion
0.00
0.02
0.04
0.06
0.08
0
20
40
60
80
100
120
140
Amount of nodes




```
# plot for my alg prim
my_time = [0.000138582952786237, 8.237501606345177e-05, 0.0001573339686729014, 0.0002812910242937505, 0.0014451250317506492, 0.0015454098083078
nodes_amount = [5, 10, 15, 20, 30, 40, 50, 80, 100, 150]
xpoints = np.array(nodes_amount)
ypoints = np.array(my_time)

plt.plot(xpoints, ypoints)
plt.title("Plot for my alg")
plt.xlabel("Amount of nodes")
plt.ylabel("Time of completion")
plt.show()
```

Аналогічно до Краскала, вбудовані алгоритми Python справляються гірше з графами з більшою кількістю ребер