**Descrição do Problema e da Solução**

A nossa solução proposta foi a utilização do algoritmo de Kruskal mas em vez de procurar a árvore geradora mínima procuramos a máxima, para tal utilizamos uma estrutura de dados disjoint-set, o que funciona da seguinte maneira: Ordenamos todas as arestas por ordem decrescente, uma a uma verificamos se os seus vertices se encontrão no mesmo conjunto e caso não se encontrem, estes são unidos (no inicio cada vértice é o seu próprio conjunto unitário) e é adicionado o peso da aresta ao total. No final acabamos com um só conjunto que contem todos os vértices e o total indica o custo de percorrer a árvore formada (que respreseta o valor máximo de trocas comerciais).

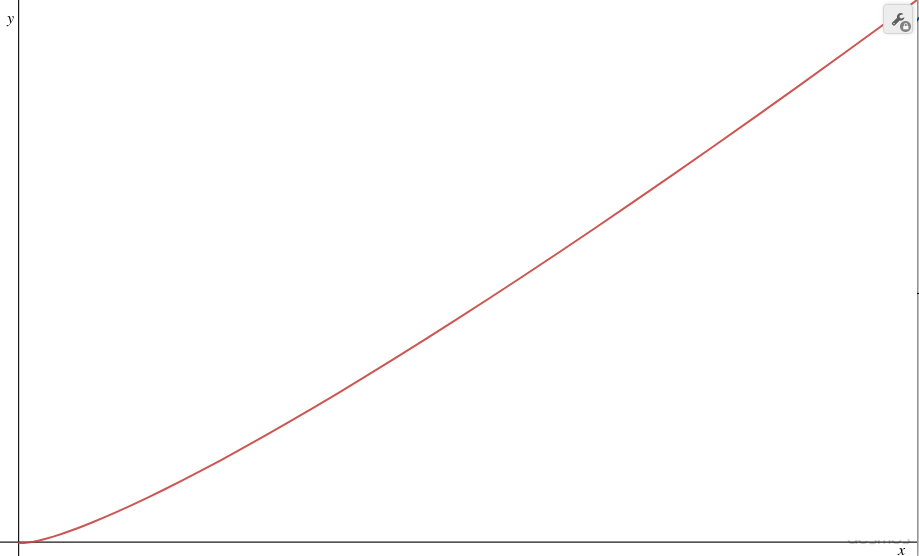
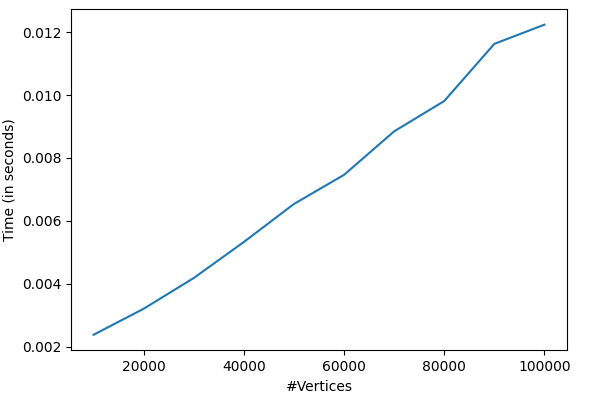
**Análise Teórica**

* Leitura dos dados de entrada: simples leitura do input e inicialização de variáveis, com 2 ciclos respetivamente a depender linearmente de V e E - O(V + E)
* Aplicação do algoritmo - O(E \* log(V)) + O (E \* log(E))
  + Ordenação de todos as arestas com a função sort (c++) - O(E \* log(E))
  + Cliclo a depender linearmente de E - O(E \* log(V))
    - Verificar se os vértices estão no mesmo conjunto (path compression) - O(log(V))
    - União dos conjuntos – O(1)

A complexidade total seria então: O(E \* log(V)) + O (E \* log(E)) como #E >= V - 1 é sempre verdade podemos dizer então que a Complexidade global da solução é O(E \* log(E))

**Avaliação Experimental dos Resultados**

O gráfico seguinte foi gerado com 10 testes diferentes em que o número de vertices é igual ao número de arestas aumentando sempre o número em 10000 (10000, 20000, 30000 ... 100000).

O gráfico que se segue é o gráfico da função n\*log(n), comparando o crescimento dos 2 gráficos verificamos que a implementação está de acordo com a análise da complexidade teórica.