Tugas Kecil II Strategi Algoritma

**Laporan Penyelesaian Topological Sort dengan Decrease and Conquer**

Oleh:

Kinantan Arya Bagaspati

13519044



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

1. **Algoritma Decrease and Conquer**

Algoritma decrease and conquer merupakan salah satu variasi dari sebuah algoritma yang lebih generalnya yakni divide and conquer. Sesuai namanya, algoritma decrease and conquer memiliki 2 bagian utama yakni:

* Decrease : Mereduksi persoalan menjadi persoalan yang lebih kecil
* Conquer : Memproses satu upa persoalan secara rekursif

Berbeda dengan divide and conquer, tidak ada tahap combine dalam algoritma decrease and conquer, karena tidak terjadi pembagian persoalan, melainkan hanya pengecilan ukuran persoalan yang sudah ada.

Pengecilan persoalan ini juga dapat dibagi lagi menjadi 3 varian, yakni:

* Decrease by a constant

Ukuran persoalan dikurangi sebesar sebuah konstanta (biasanya bernilai 1) setiap pemanggilan metode decrease dalam algoritma. Contoh: Topological sort, Insertion sort, Selection sort.

* Decrease by a constant factor

Ukuran persoalan dibagi dengan sebuah konstanta setiap pemanggilan metode decrease dalam algoritma. Contoh: Binary search (konstanta 2), Pencarian koin palsu (konstanta 3, karena dapat dibagi 3 kelompok dan timbang 2 kelompok diantaraya)

* Decrease by a variable size

Pengurangan ukuran persoalan berbeda-beda tiap pemanggilan metode decrease-nya, bergantung pada sebuah variabel. Contoh: Interpolation search, Mencari nilai median.

Sebagaimana algoritma lanjutan lain pada umumnya, decrease and conquer hadir sebagai optimasi dari algoritma yang paling memakan waktu paling banyak yakni brute force. Namun optimasi ini tentunya bervariasi tergantung jenis persoalan yang diberikan. Decrease and conquer sangat bagus dipakai saat penyelesaian dari data yang menjadi persoalan memiliki metode yang serupa dengan penyelesaian sebagian dari data tersebut, terlebih lagi bila hasil dari sebagian dari data digunakan untuk mempermudah penyelesaian data awal. Decrease and conquer juga tidak selalu memberikan optimasi yang signifikan bila data yang digunakan merupakan worst case dan kebetulan metode pengecilan yang digunakan ialah decrease by a variable size, meski tentunya kemungkinan hal ini terjadi cukup kecil

Dalam menyelesaikan tugas kecil topological sort yang diberikan, saya menggunakan algoritma decrease and conquer yang mengutamakan kecepatan program. Terdapat satu hal utama yang berpengaruh secara signifikan pada waktu eksekusi program saya, yakni struktur data yang saya gunakan dalam menyimpan setiap pasangan mata kuliah {x, y} dengan x ialah prasyarat dari y. Daripada saya menyimpan array of vector prequisites[NMAX] yang perquisites[i] menyimpan semua ID mata kuliah yang menjadi prasyaratnya, lebih baik saya menyimpan array of vector preqOf[NMAX] yang preqOf[i] menyimpan semua ID matakuliah yang memuat mata kuliah ber-ID i sebagai prasyaratnya. Ini dilakukan agar memudahkan mengurangkan jumlah prasyarat mata kuliah yang bersangkutan setelah mata kuliah ber-ID i diambil. Selain itu digunakan pula struktur data queue bernama queueCourse dengan implementasi array sebagai tempat memasukkan mata kuliah yang siap untuk diambil karena jelas dapat diperlakukan secara First In First Out. Selanjutnya akan dijelaskan alur kerja program beserta kaitannya dengan decrease and conquer.

Pertama program membaca file masukan sebanyak 2 kali dengan prosedur mapping() dan setPreqOf(). Pada mapping(), program hanya membaca kode mata kuliah pertama setiap baris untuk memberi ID setiap string yang terbaca, dan kemudian memasukkannya dalam map bernama courseCodetoID dengan key string yang terbaca dan value ID string tersebut, serta dalam array of string IDtoCourseCode yang fungsinya sesuai nama arraynya. Pada setPreqOf(), program sudah memberikan ID pada setiap mata kuliah yang tercatat dalam file sehingga sudah dapat dimasukkan ke dalam array of vector preqOf sesuai yang dijelaskan di paragraph sebelumnya. Prosedur ini juga mengisi array nbPreq dengan banyaknya prasyarat tiap ID mata kuliah terkait, serta mengisi queue dengan ID mata kuliah yang sudah bisa diambil (nbPreq[ID] == 0).

Dari kedua prosedur di atas, semua data sudah siap untuk diproses. Selanjutnya program akan menjalankan prosedur solve() yang menyelesaikan persoalan topological sort. Program akan menyimpan nilai tail saat ini dalam tempTail, kemudian jelas bahwa semua mata kuliah yang ada dalam queue saat ini (dari index head hingga tempTail) tentunya dapat diambil saat semester ini. Oleh karena itu program akan melakukan pop() pada queue satu per satu hingga head = tempTail, dengan mengurangi jumlah prasyarat sejumlah 1 dari semua mata kuliah yang memuat mata kuliah dengan ID hasil pop tersebut sebagai prasyaratnya. Disinilah preqOf dirasa sebagai struktur data yang superior. Kemudian setelah head = tempTail, program akan mempunyai queue baru, sisa mata kuliah serta sisa pasangan mata kuliah dan prasyaratnya yang tentunya berkurang jumlahnya. Itulah bagian decrease dari topological sort, dilanjutkan dengan conquer karena metode penyelesaian sisa mata kuliah yang ada sama persis dengan metode awal.

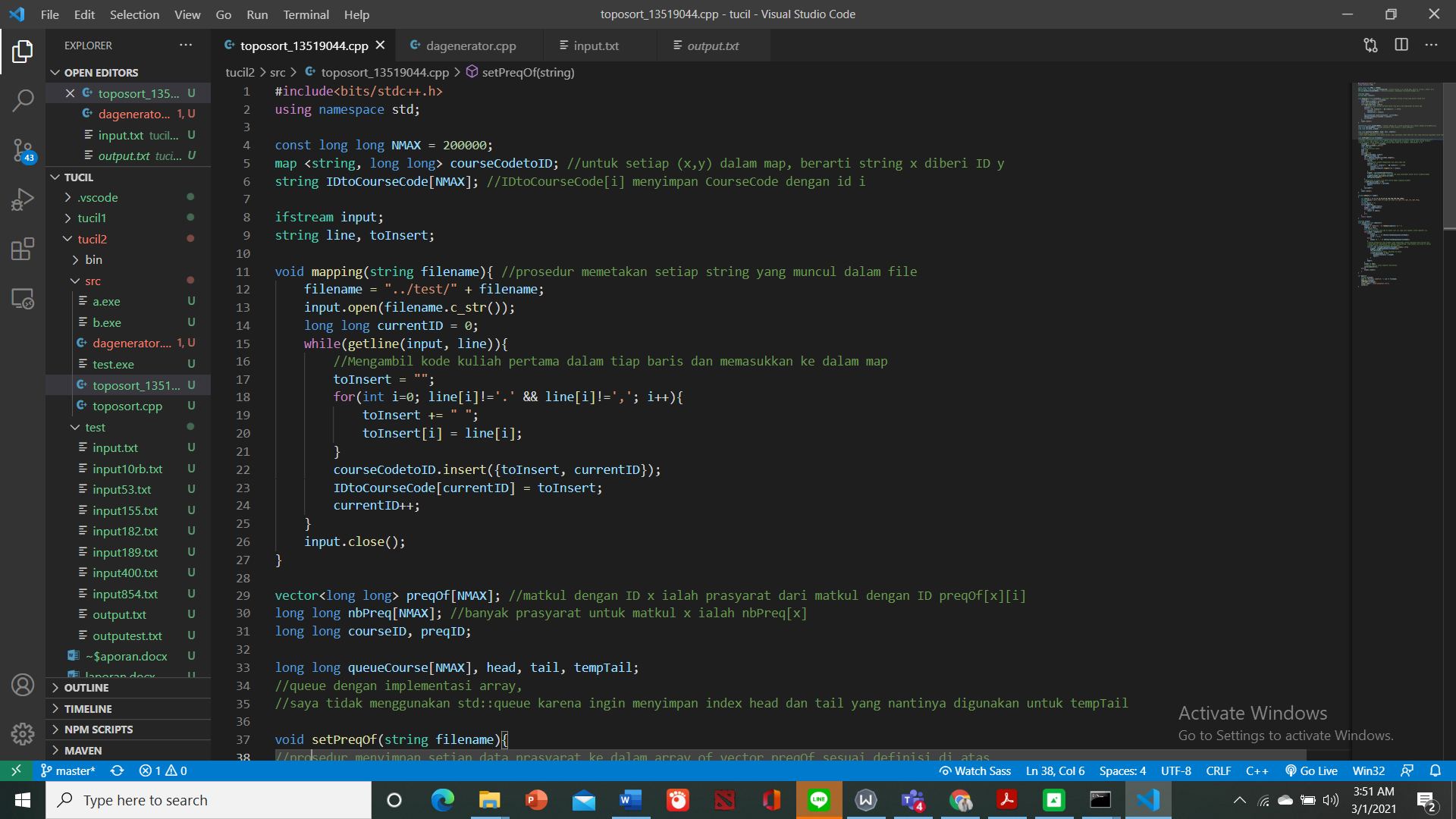
Analisis kompleksitas dari program topological sort ini cukup mudah. Misalkan V dan E berturut-turut merupakan banyak node (mata kuliah) dan banyak edge (relasi prasyarat) dari DAG (daftar prasyarat). Diawal program terdapat penyimpanan pasangan string dan IDnya dalam map yang jelas merupakan O(Vlog(V)). Kemudian pembacaan pada prosedur setPreqOf() juga masih perlu menggunakan map untuk mengidentifikasi ID dari string sehingga bernilai O(Elog(V)). Selanjutnya setiap pem-pop-an queue, dilakukan perubahan nilai array nbPreq sebanyak mata kuliah yang memiliki head queue sebagai prasyarat. Apabila semua operasi tersebut ditotal, jelas memiliki kompleksitas O(E). Dengan menjumlahkan semua kompleksitas diperoleh kompleksitas total ialah O((V+E)logV).

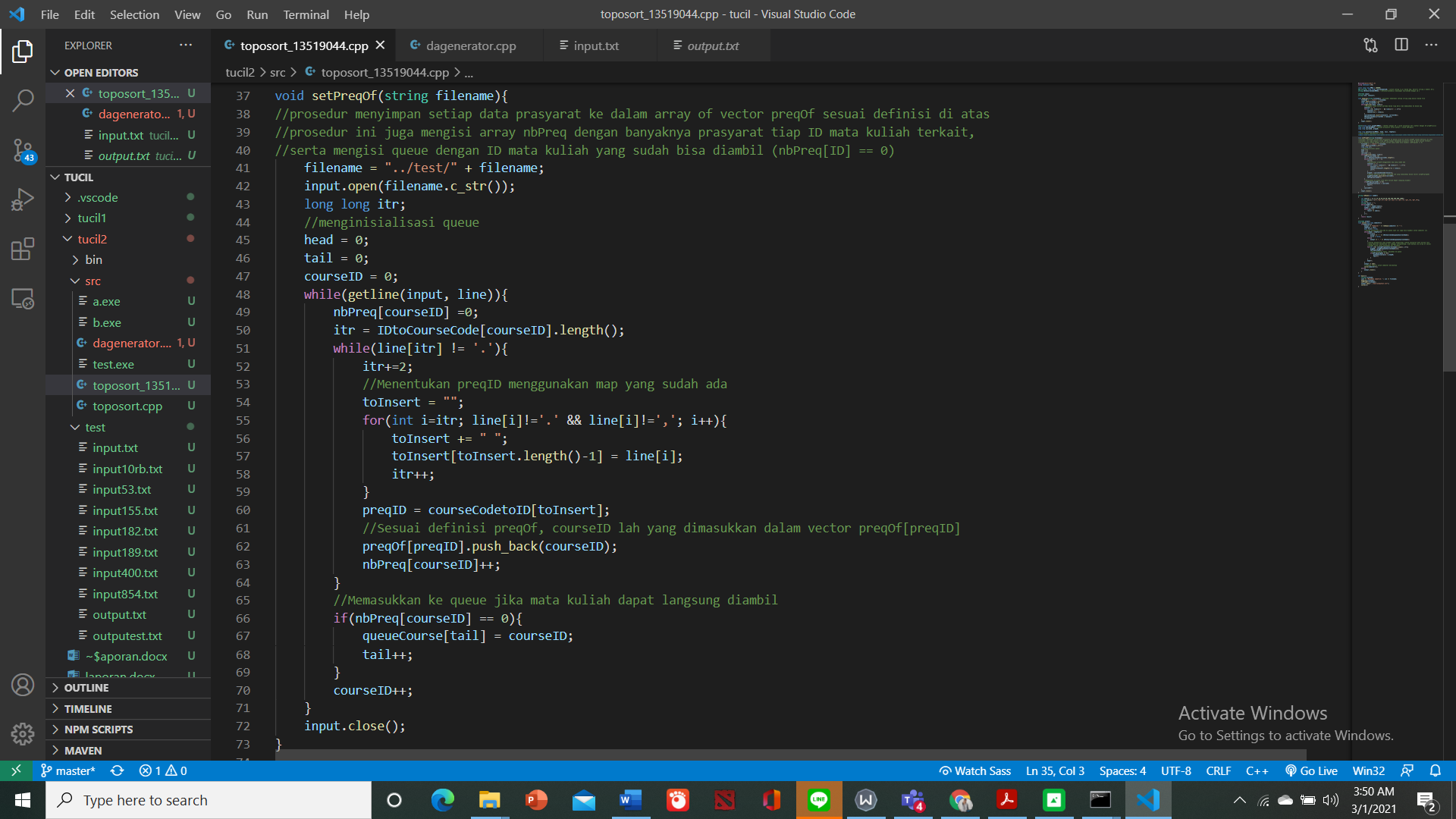
Selain topological sort, saya juga mengimplementasikan test case generator yang mengenerate directed acyclic graph yang menandakan pasangan mata kuliah dan prasyaratnya dan kemudian menuliskannya dalam file sesuai format yang dibutuhkan untuk tucil ini. Program ini bernama DAGgenerator.cpp yang meminta input:

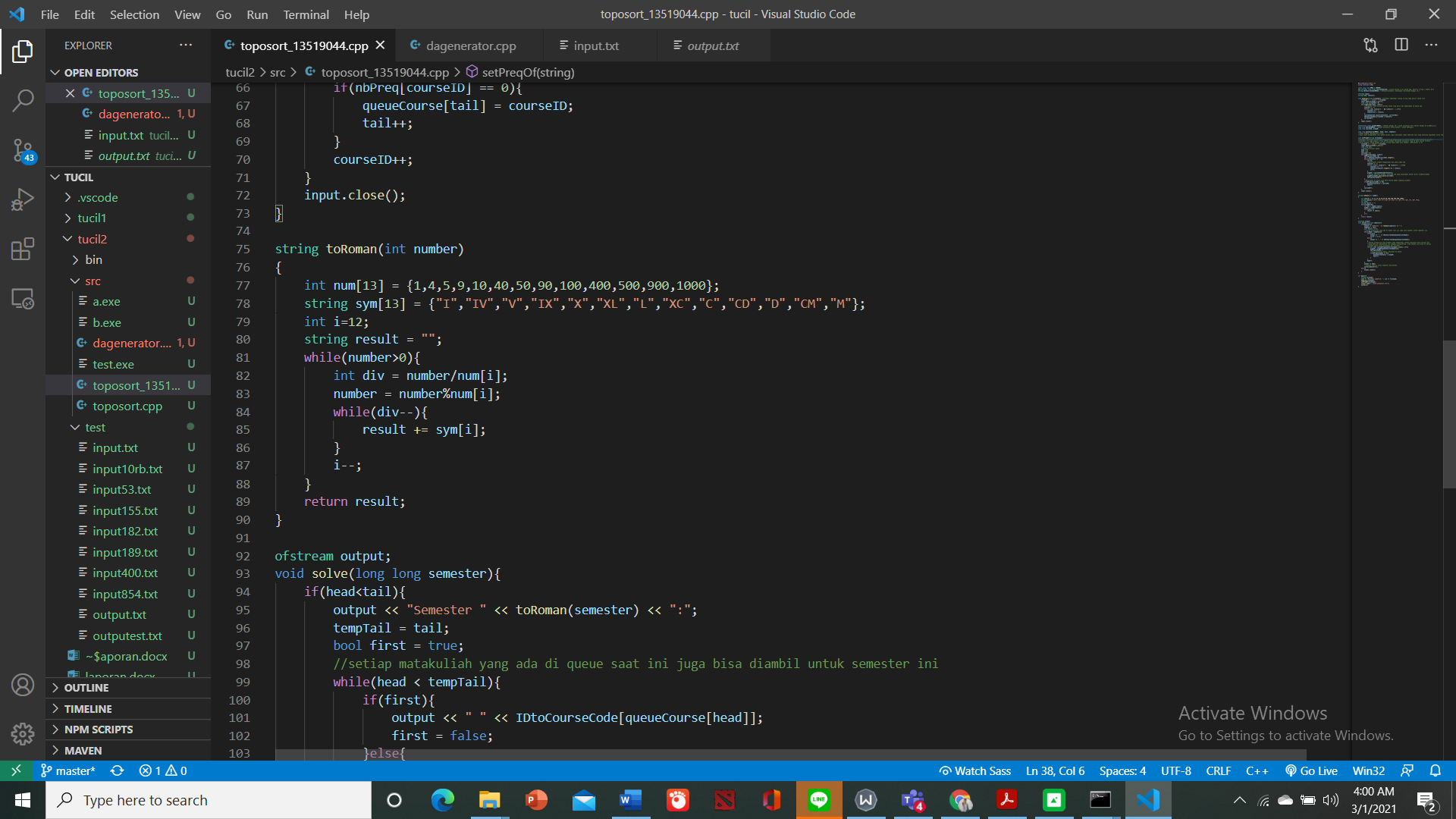
* majorCode sebagai kode jurusan
* nbCourse sebagai banyak mata kuliah yang ingin dihasilkan
* MaxSemester yang berarti DAG yang dihasilkan pasti memiliki solusi untuk banyaknya semester sama dengan maxSemester
* Density yang menandakan jumlah edge yang mungkin dalam graph tersebut. Apabila density bernilai 0 maka tidak ada edge, sementara bila bernilai 100000 maka setiap edge yang mungkin dibentuk akan terbentuk
* Pseudorandomizer yakni agar matakuliah yang dihasilkan tidak urut, setiap ID matakuliah dikalikan dengan nilai pseudorandomizer ini kemudian dimodkan dengan 100000. Pseudorandomizer harus relative prima dengan 10

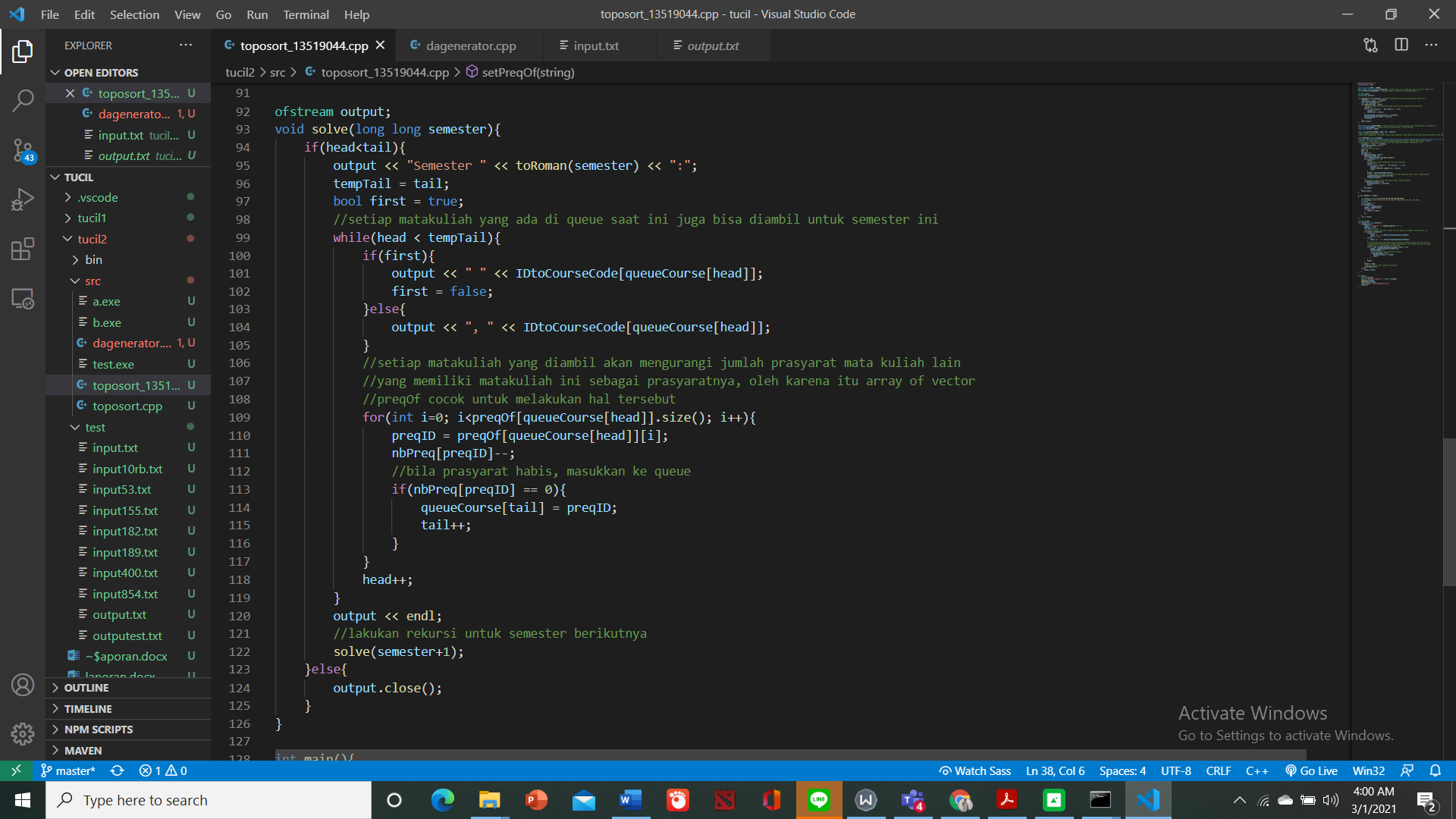
Algoritma yang digunakan generator saya cukup simpel, pertama saya menginisialisasi array of vector preqs yang menyimpan prasyarat setiap ID mata kuliah. Kemudian setelah mendapat banyak mata kuliah dan jumlah semester, program akan melakukan ramdomisasi untuk menghasilkan barisan partition[maxSemester+1] bilangan terurut menaik yang akan berperan menjadi partisi, sehingga node dengan ID diantara partition[i] dan partition [i+1] tidak boleh memiliki prasyarat node dengan ID diatas partition[i+1] namun boleh memiliki prasyarat node dengan ID dibawah partition[i]. Setiap edge yang mungkin ada pun ditambahkan dengan probabilitas sesuai dengan density/100000. Kemudian program menuliskan setiap ID yang sudah dipseudirandomize ditambahkan dengan kode kuliah di depannya dalam file input{nbCourse}.txt.

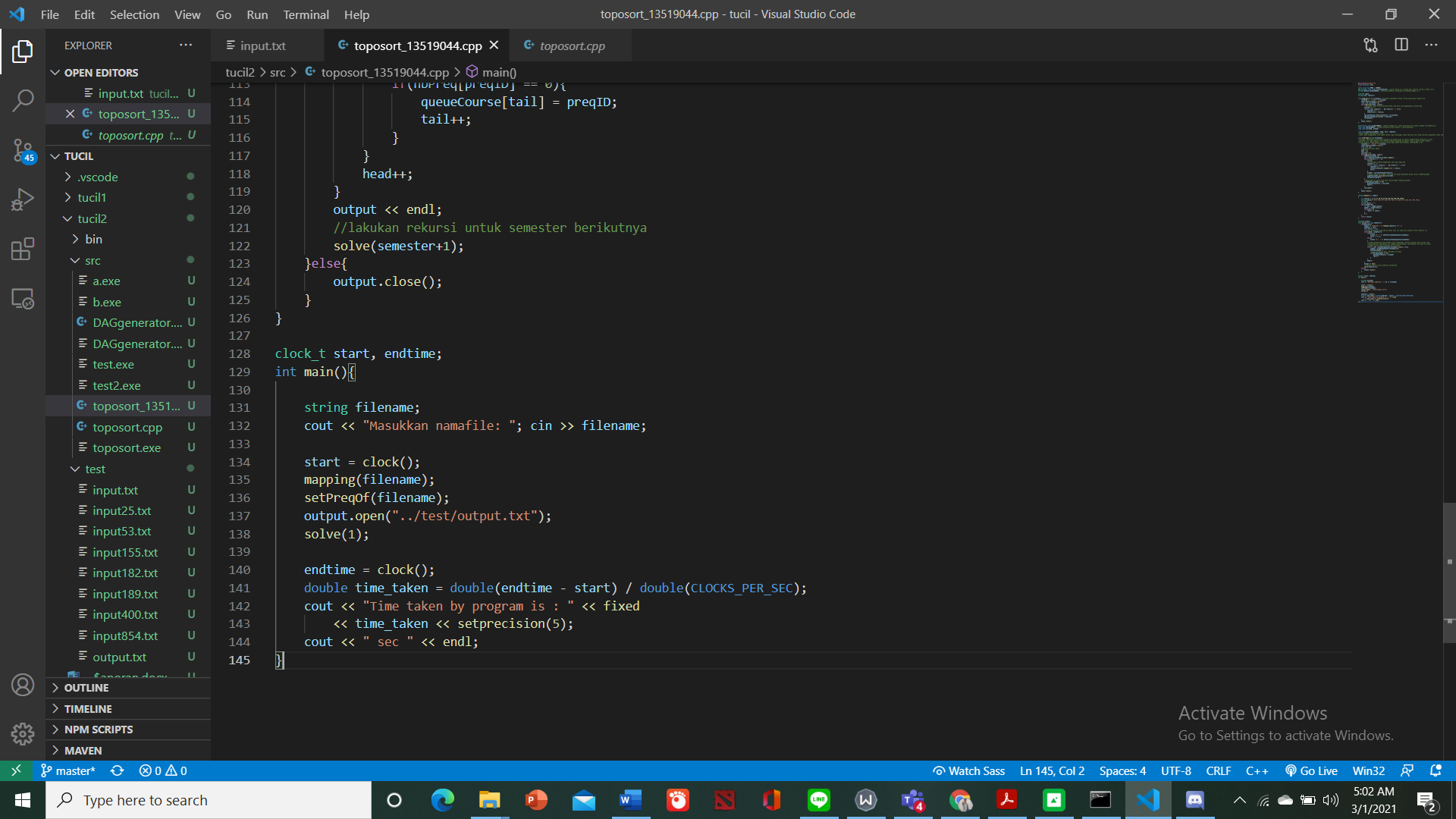
1. **Source Program**
2. toposort\_13519044.cpp



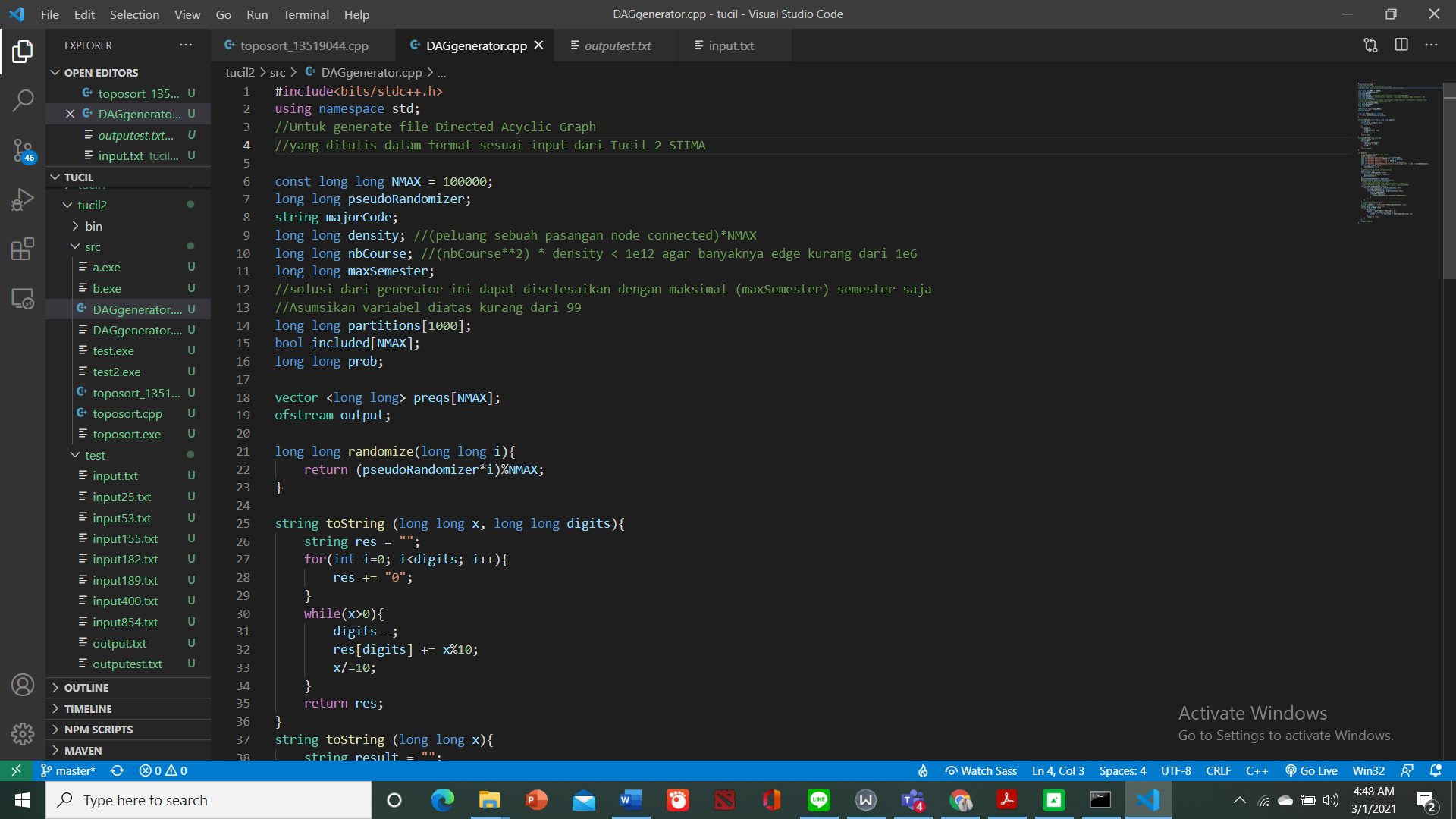


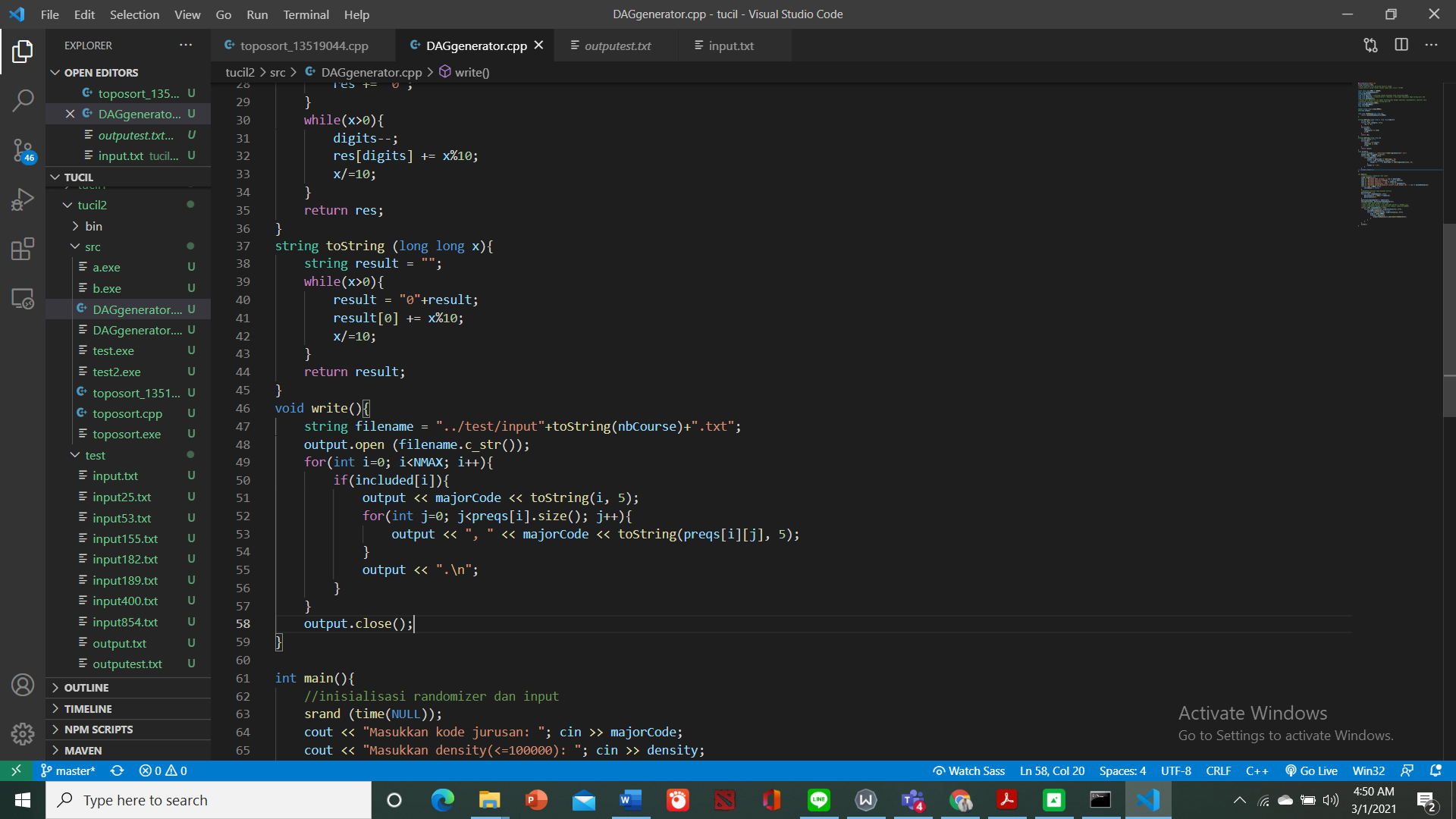


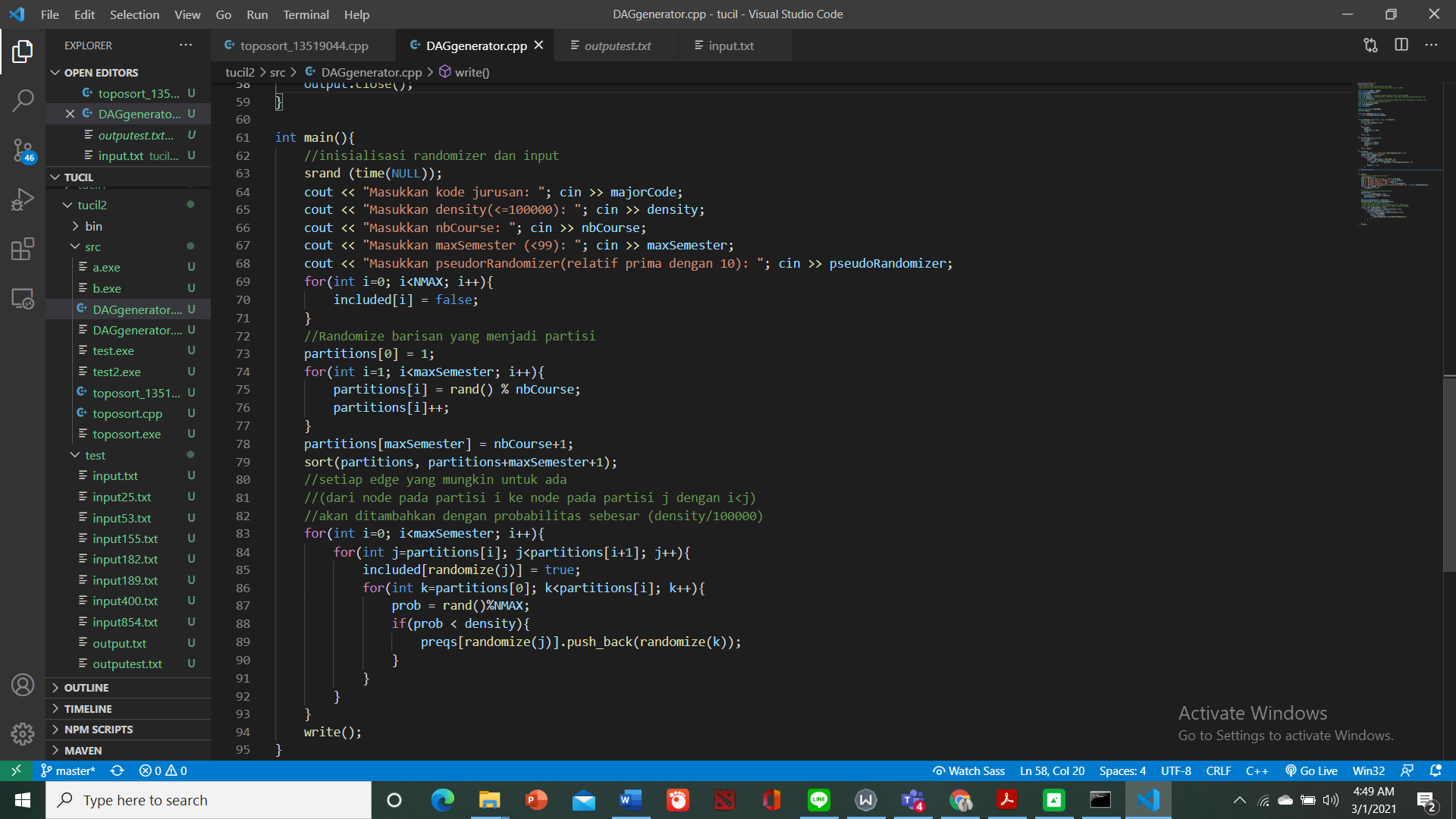




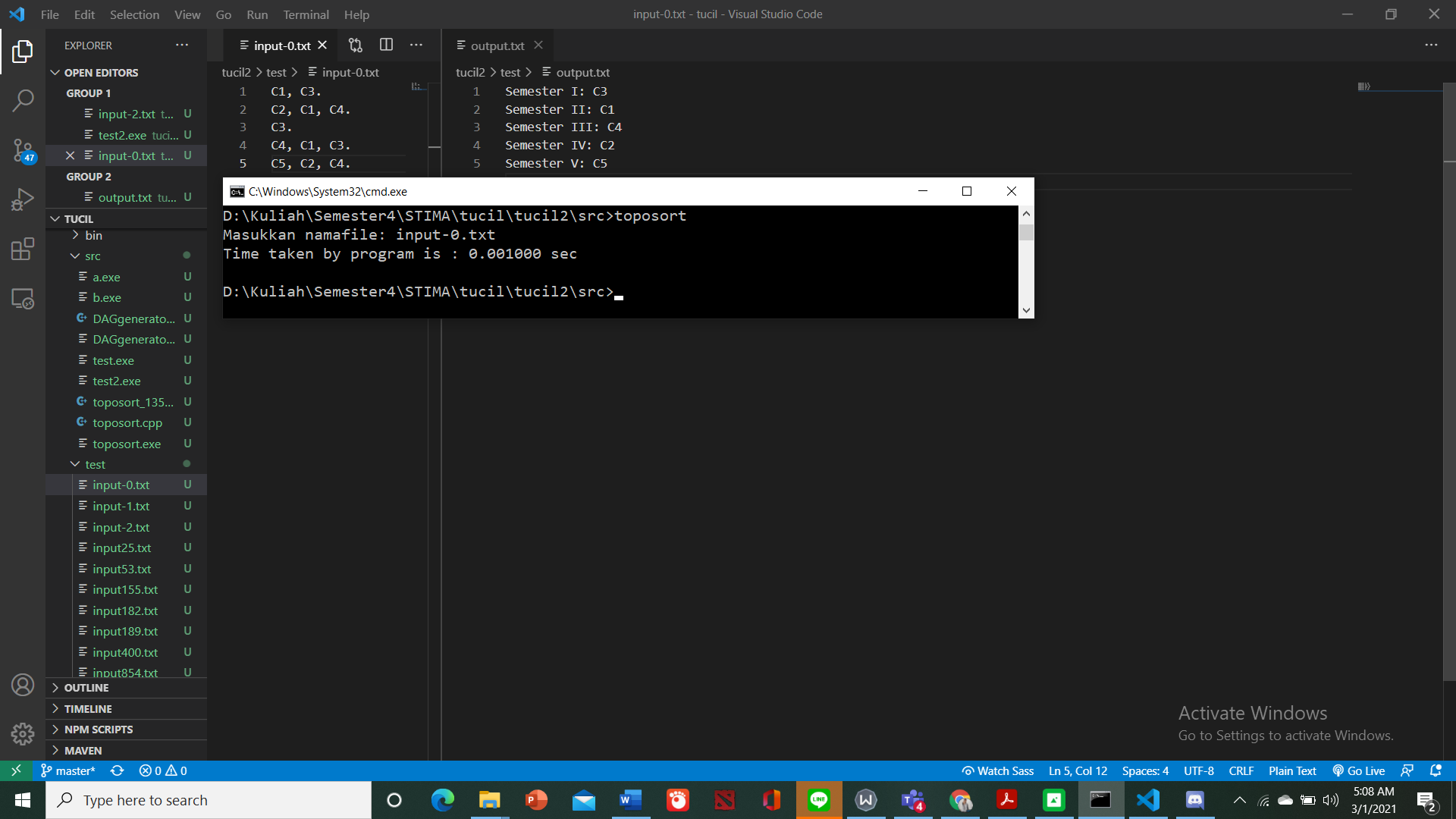
1. DAGgenerator.cpp

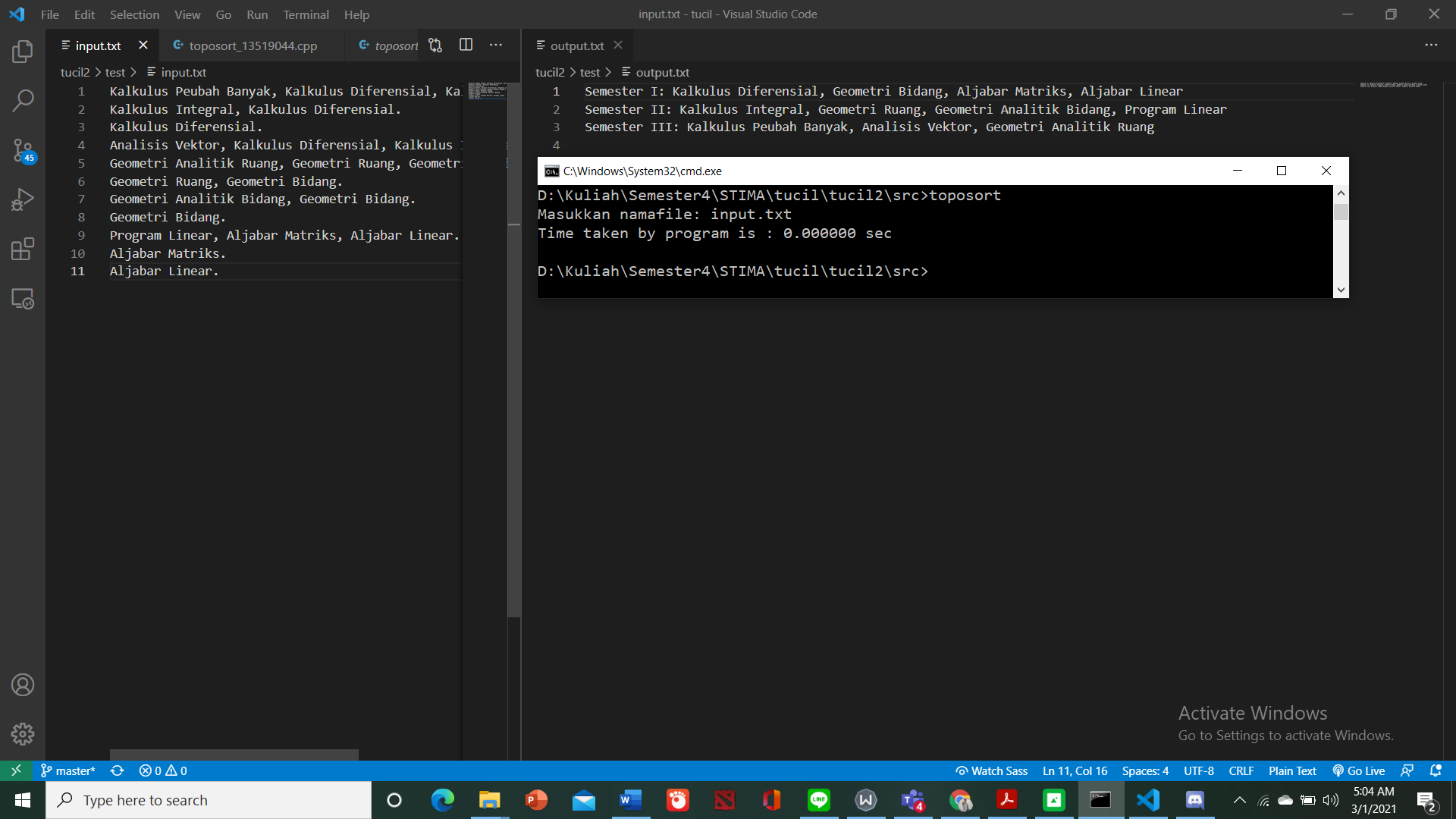


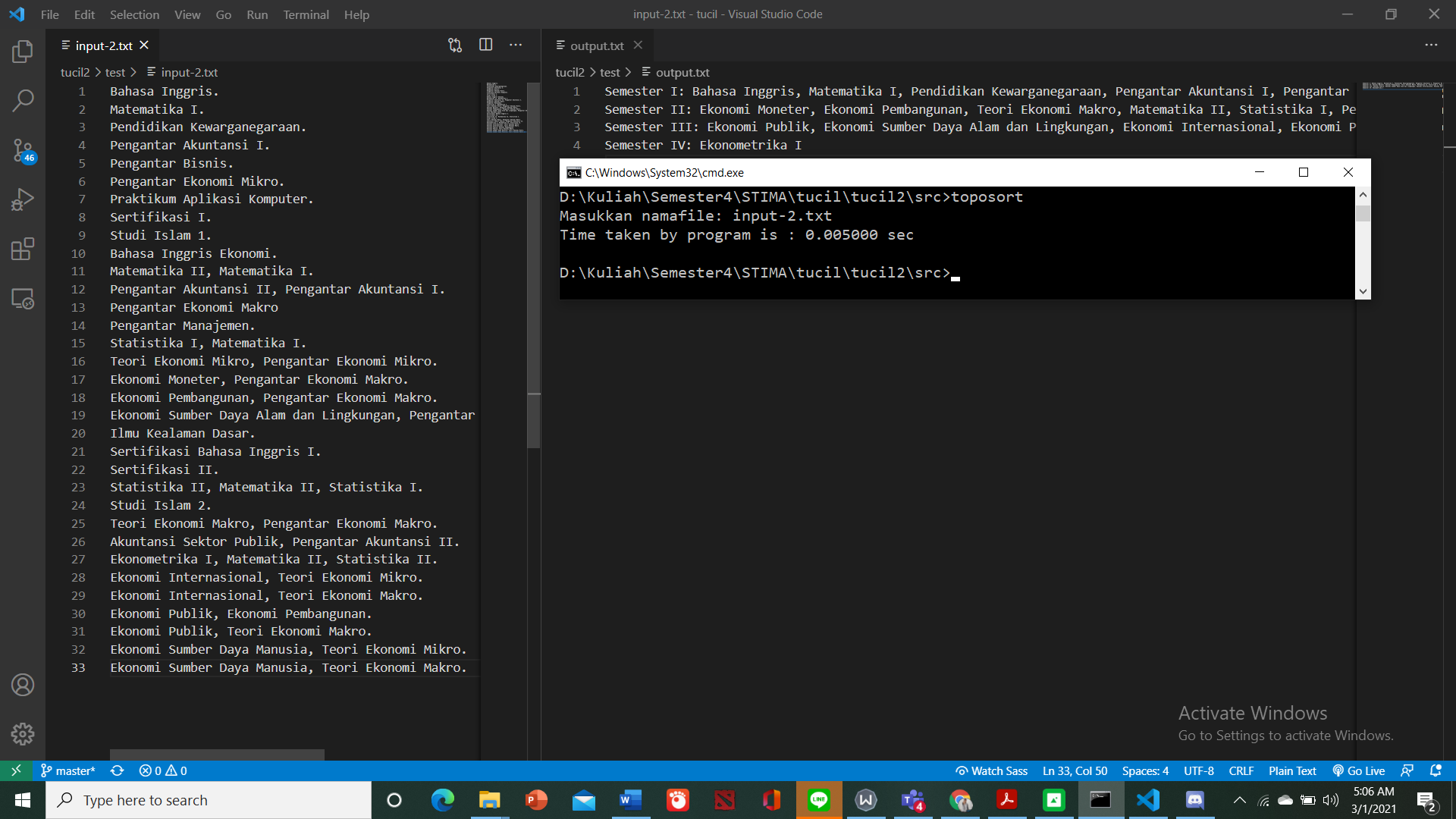


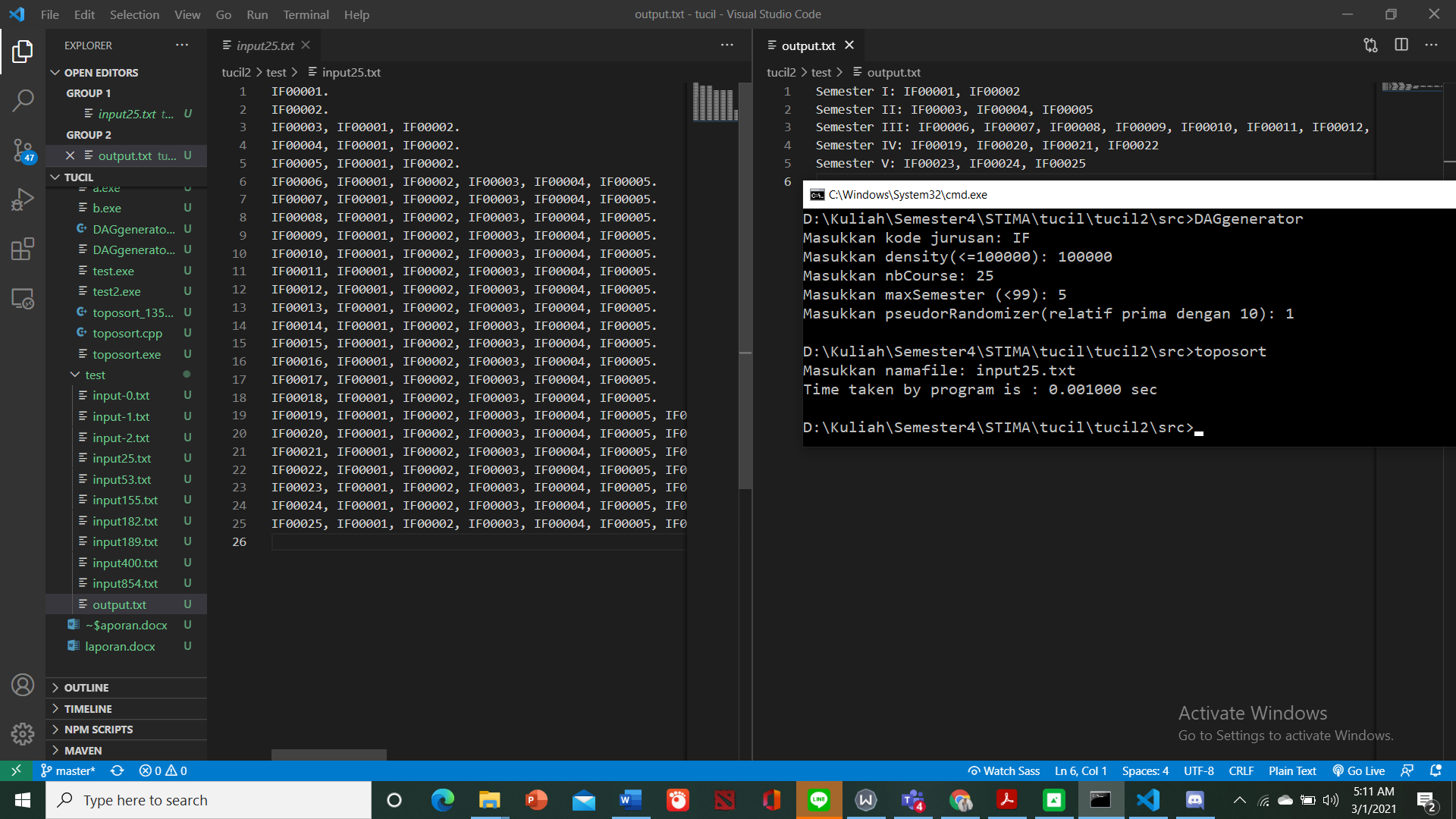


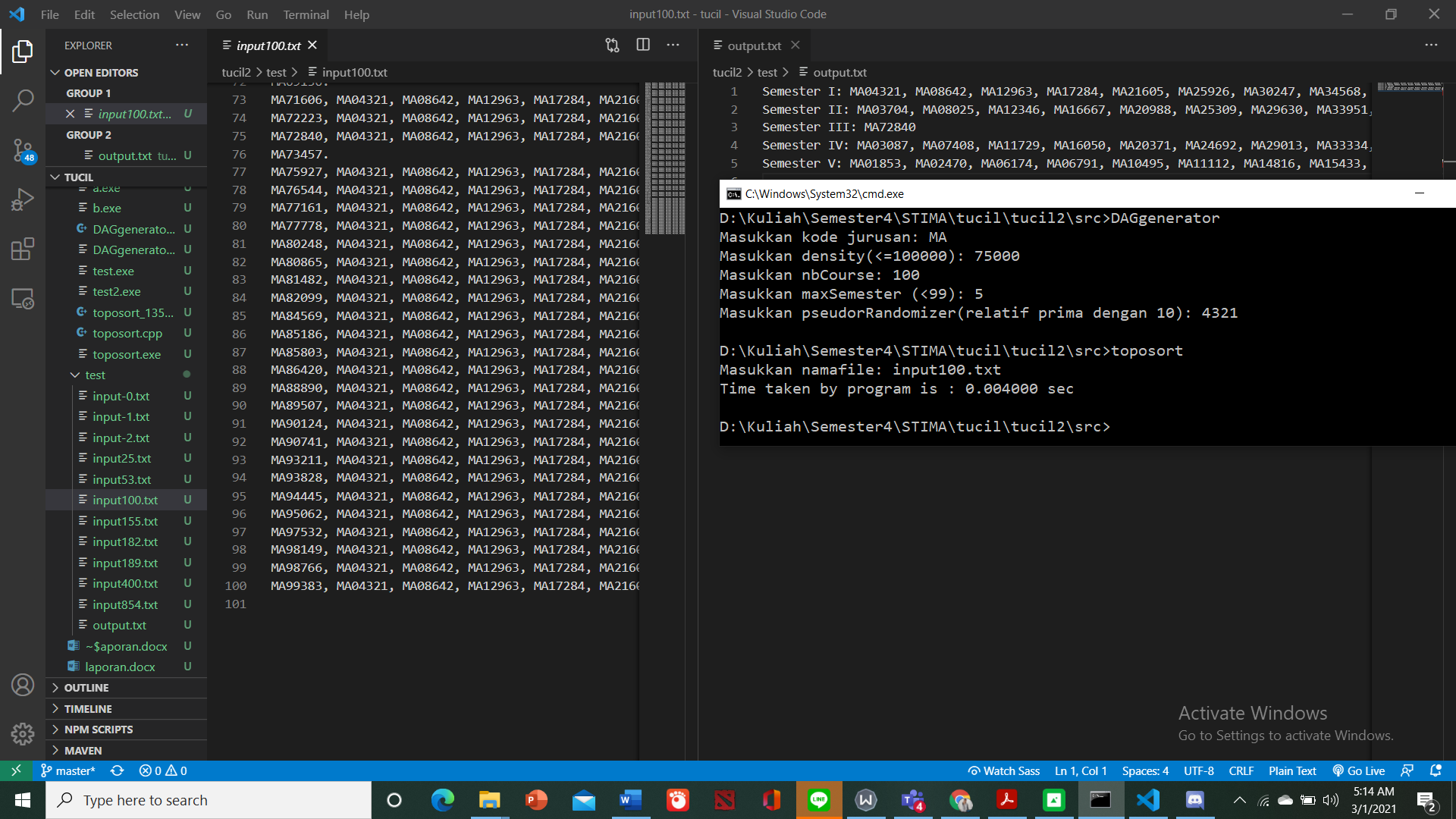
1. **Input dan Output**

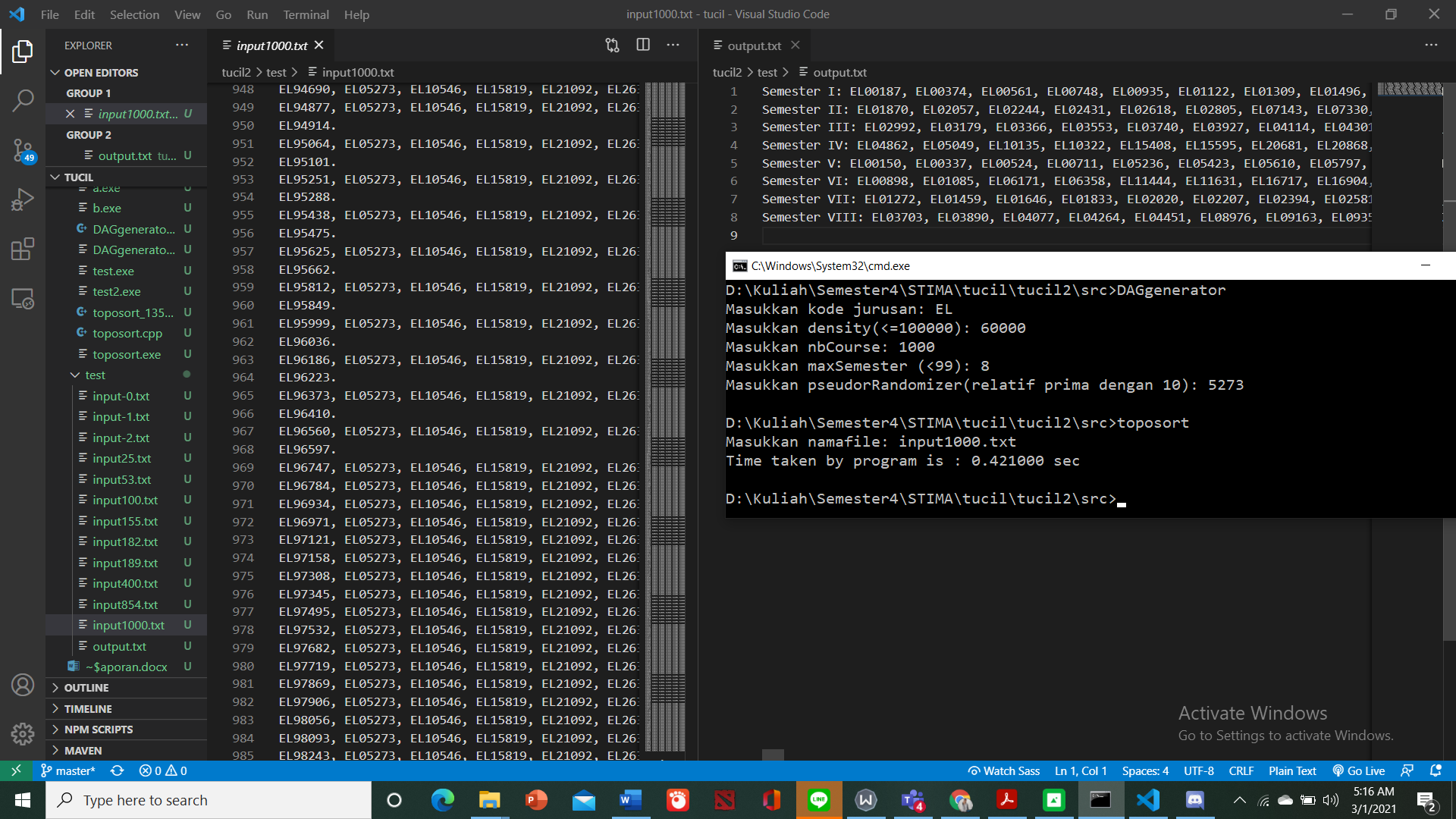


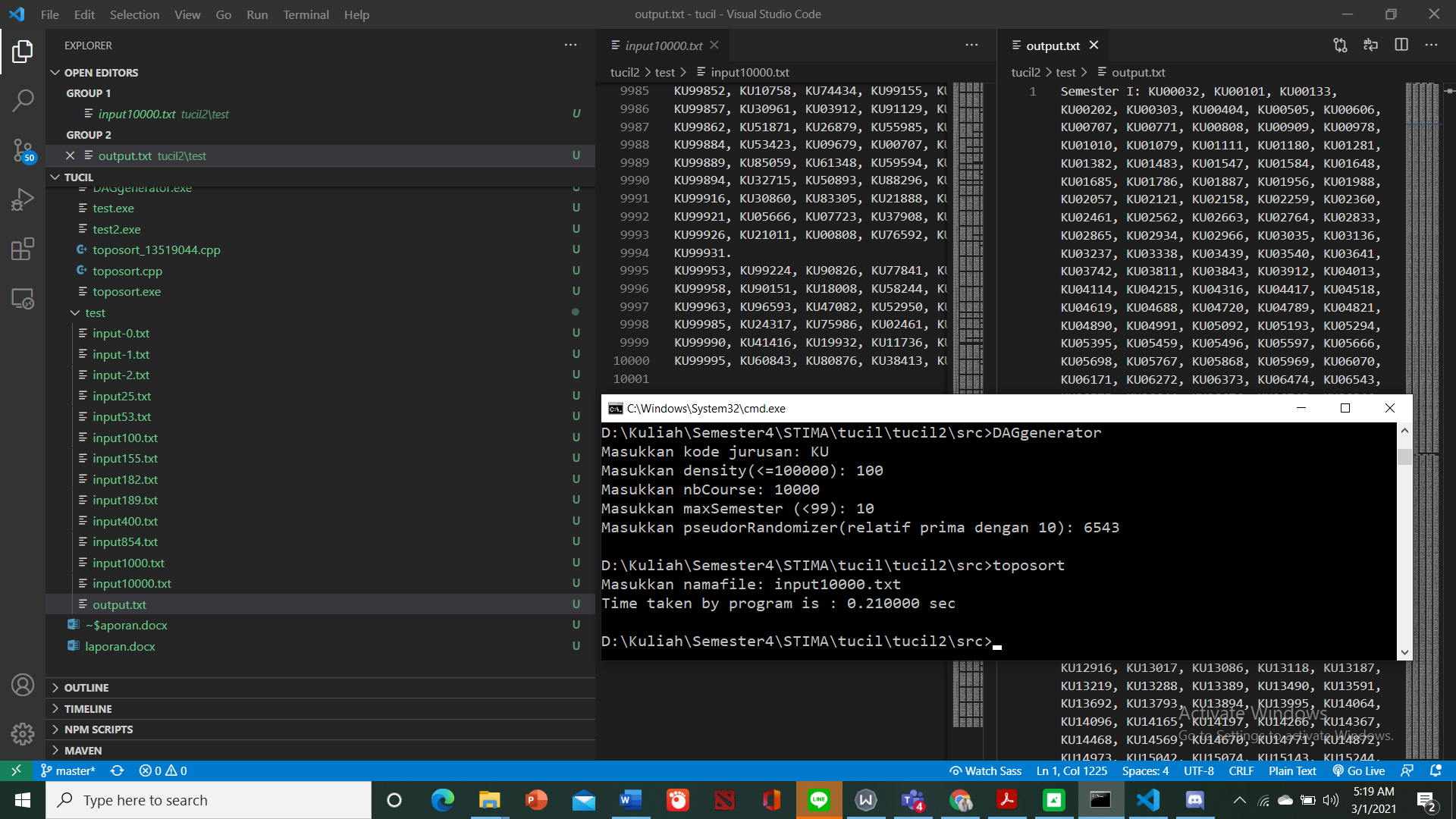


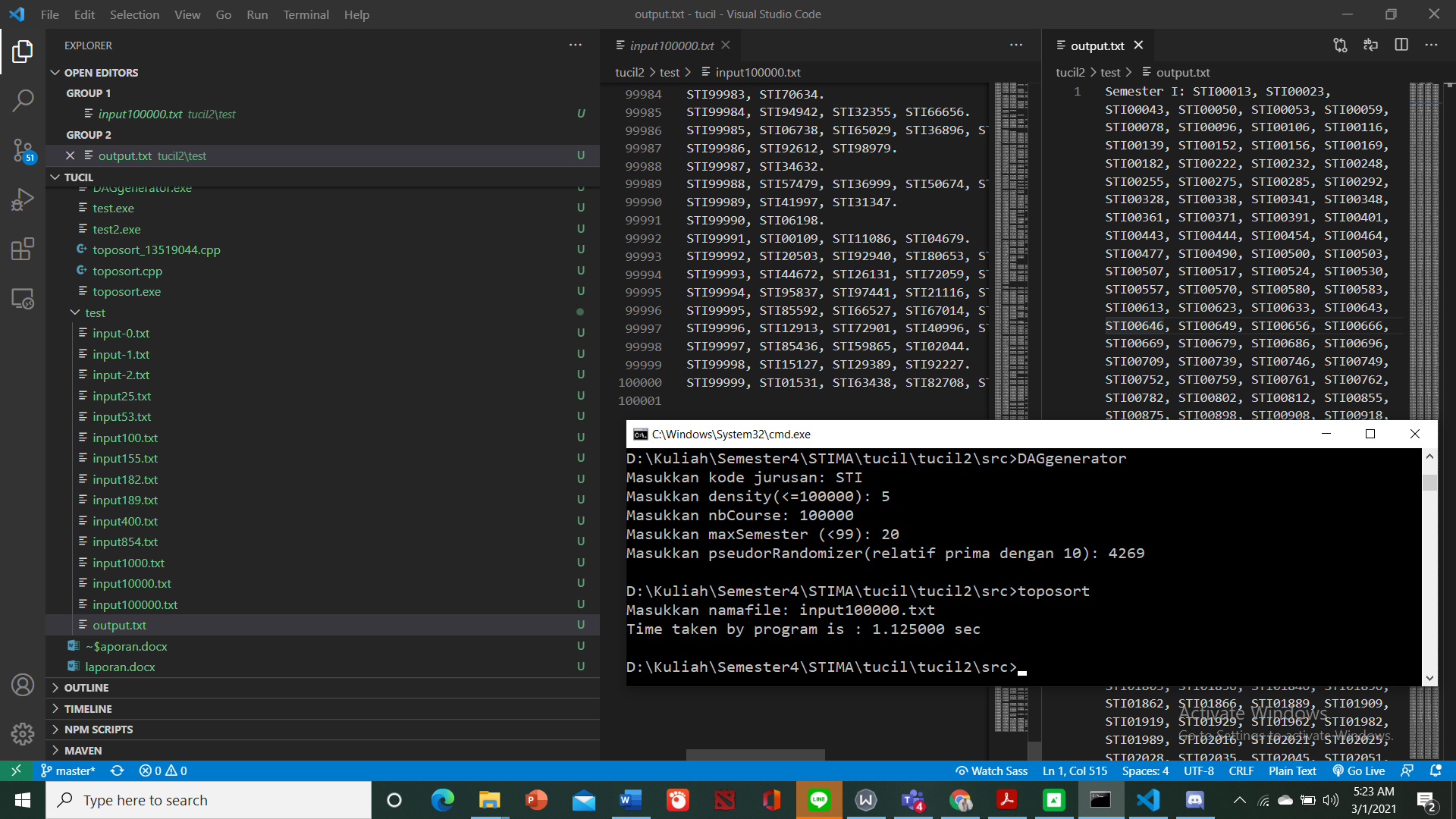












1. **Alamat Drive**

<https://drive.google.com/drive/folders/11ujwLwBe8aRZPMXDNL8mpoIqBsVb5NvO?usp=sharing>

1. **Tabel Ceklist**

|  |  |  |
| --- | --- | --- |
| **Poin** | **Ya** | **Tidak** |
| Program berhasil dikompilasi tanpa kesalahan | √ |  |
| Program berhasil running | √ |  |
| Program dapat membaca berkas input dan menuliskan output | √ |  |
| Luaran sudah benar untuk semua kasus input | √ |  |