Oefeningen Sorteeralgoritmen:

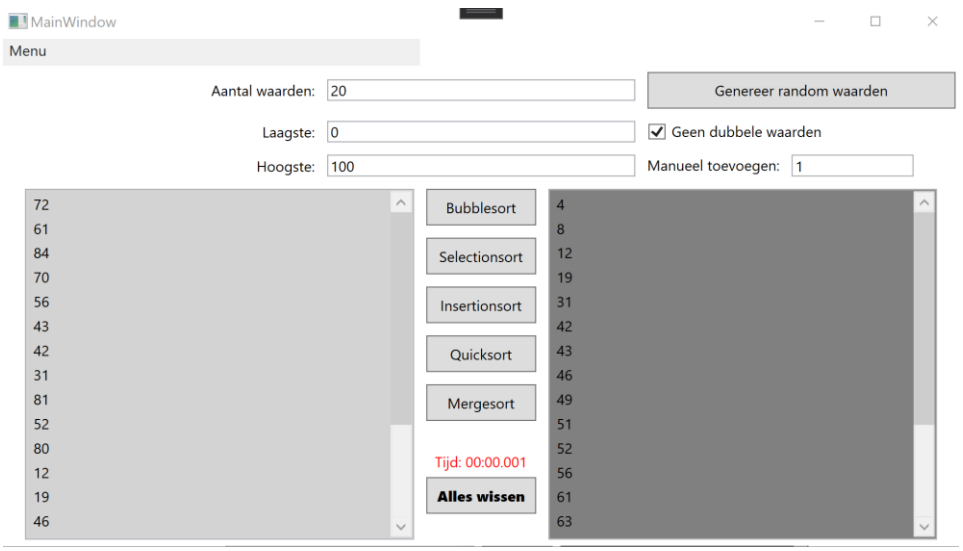
De voorbereiding hieronder kan je indien je wenst reeds uit de oplossing in github halen. Daarin zit reeds een randomgenerator + een WPF applicatie om de sorteer algoritmen te testen. Op die manier kan je sneller starten met de essentie: namelijk de sorteerlalgoritmen zelf en kan je dus overgaan naar oef 2.

1. Voorbereiding:
    a. Om onze sorteeralgoritmen te kunnen testen hebben we lijsten nodig met getallen die willekeurig in de lijst voorkomen. Het zou handig zijn dat je dus eerst een **RandomGenerator** maakt (of misschien heb je hiervoor nog wat c# code liggen).
    b. Maak een klasse Randomgenerator (int **Amount**, int **Min**, in **Max**, bool **Unique**)
    c. Deze klasse heeft 1 methode : int[] **Generate**() dewelke een array zal aanmaken met 'Amount' willekeurige getallen tussen 'Min' en 'Max'. Of een zelfde getal al dan niet meermaals mag voorkomen in die lijst zal afhangen van de 'Unique' property. *(als je dit laatste te complex vind, mag je die property ook gerust weglaten).*

Om onze sorteeralgoritmes visueel te kunnen testen heb ik een WPF Applicatie voorzien, waarmee de gebruiker een reeks willekeurige waarden kan laten genereren, deze worden vervolgens getoond in de linkse lijst. Vervolgens kiest hij/zij een sorteer algoritme om deze lijst te sorteren. Het resultaat hiervan zal getoond worden in de rechtse lijst.

De tijd die het algoritme nodig had om de sortering te doen zal worden opgemeten met behulp van een Stopwatch en worden getoond op het scherm.

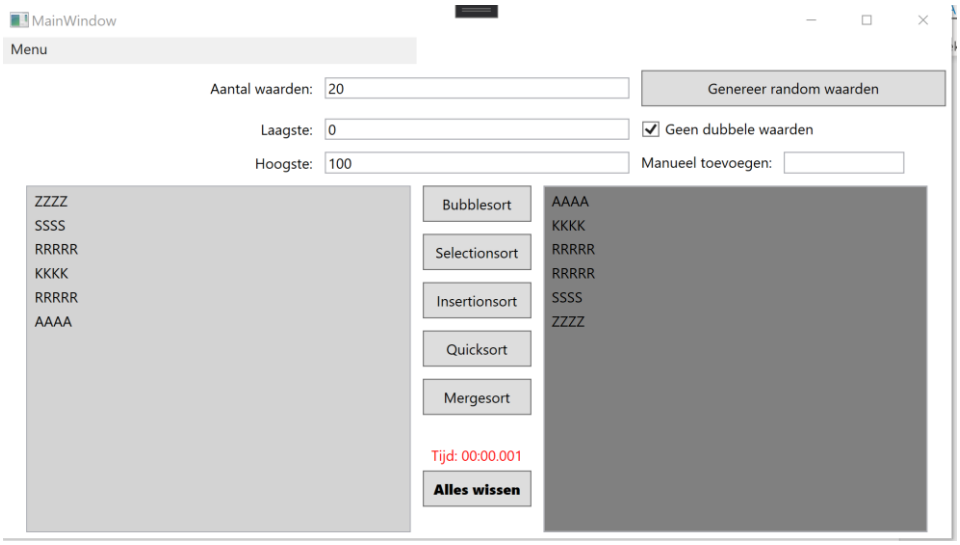
Daarnaast is er nog een controle methode voorzien die de gesorteerde lijst overloopt en nakijkt of deze wel degelijk correct gesorteerd werd (zo niet zal er een foutboodschap op het scherm verschijnen)



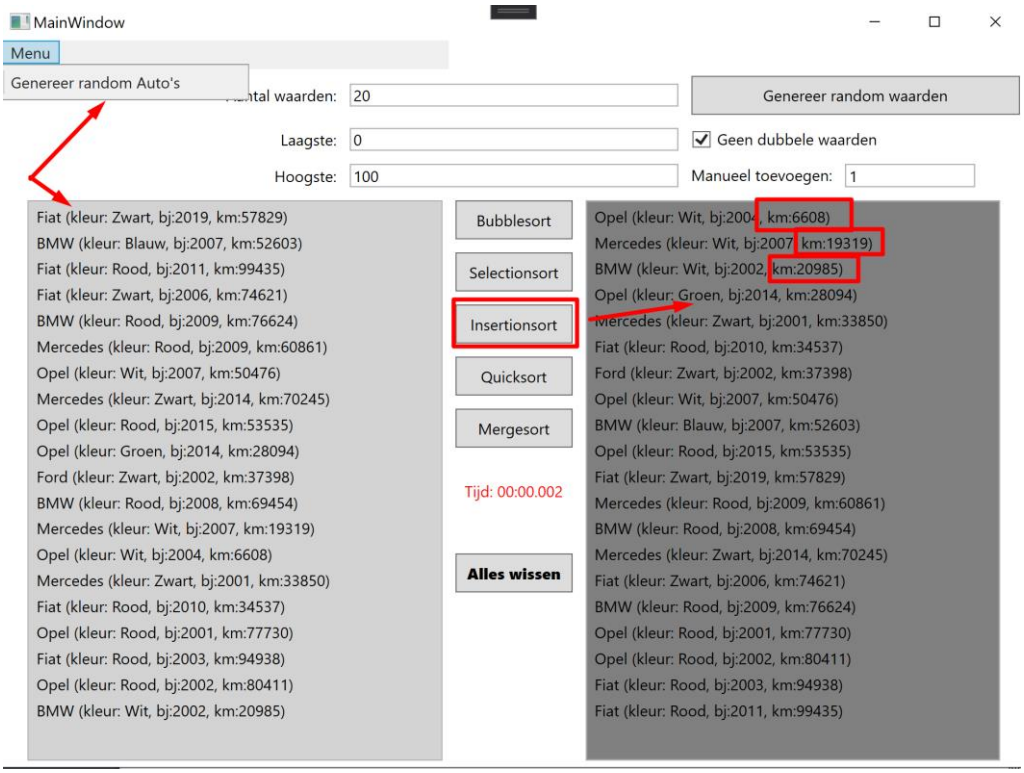
---

2. Bubble sort:
    a. Neem het Bubble sort algoritme over vanuit de cursus en test of het correct werkt (gebruik je randomgenerator) en of je de werking volledig begrijpt (gebruik hierbij de info van de cursus)
    b. Maak het algoritme intelligenter zodat het stopt **na een iteratie**, indien er tijdens die iteratie **geen enkele swap** werd uitgevoerd. Op dat moment is de lijst immers volledig gesorteerd en heeft het geen enkele meerwaarde om de overige iteraties nog te doen.
        i. Voeg een **Debug.WriteLine** statement toe dat aangeeft na hoeveel iteraties het sorteren gestopt is.
        ii. Met welke **soort van input** kan je dit beste testen of het algoritme effectief stopt na 1, 2, 3.. iteraties ?
    c. Is het mogelijk om van de Bubble sort ook een (intelligente) **recursieve** versie te maken ?
    d. Meet de **performantie** van de Bubble sort voor verschillende inputs. Hiervoor had ik reeds een stopwatch voorzien in de code van de WPF applicatie. Meet voor verschillende inputs (10,100,1000,10000,20000,30000,… waarden) de tijden op en zet ze uit in een grafiek.

3. Selection sort
    a. **Bouw zelf** het algoritme van de Selection sort gebaseerd op de info uit de cursus en de kennis die je reeds hebt van het Bubble sort algoritme. (Uiteraard als methode in een eigen klasse.)
        i. Dwz. je weet reeds dat er **2 for lussen** nodig zullen zijn.
        ii. De **binnenste for lus** zal de array overlopen op zoek naar de minimum waarde (**compare** operaties). Hierna al zal er dan niet een **swap** operatie moeten gebeuren.
        iii. Nadien komt er een volgende iteratie, enz.. tot de lijst gesorteerd is.
    b. Test het met verschillende input.
    c. Meet tevens de performantie voor verschillende input waarden.
    d. Maak eens een 2e versie van de **selection sort** die ook **strings** kan sorteren. Zorg dat de gebruiker manueel ook strings kan invoeren zodat je dit kan testen.



4. Insertion sort
    a. Bouw vervolgens ook het insertion sort algoritme. Baseer je ook hier weer op de info uit de cursus.
    b. Test en meet de performantie (1000, 10000, 20000, 30000, 50000, 100000,… elementen)
    c. Maak een nieuwe versie van het insertion algoritme dat in staat is om Auto's te sorteren op basis van hun km stand. (dus voorzie zelf een klasse Auto met deze property en nog enkele andere naar keuze: bouwjaar, model, kleur,..)



5. Quick Sort
    a. Ga op zoek naar Quick sort algoritmen op het internet en maak hiervan een c# versie (je mag uiteraard zelf ook een versie ontwikkelen op basis van de info uit de presentatie)
        i. Je zal verschillende mogelijkheden vinden wat betreft “pivot selectie”
        ii. Je zal verschillende mogelijkheden vinden wat betreft “partionering”
    b. Doe ook hiervan een tijdsperformantie meting met verschillende input.

    c. Maak ook hier een versie om de Auto’s te sorteren, echter nu een verbetering zodat de logica die bepaalt **wat er moet worden vergeleken** (km stand, bouwjaar,Model…) **niet meer in het sorteeralgoritme** zit **maar wel in de Auto klasse** zelf. (tip je zal moeten gebruik maken van operator overloading)

6. Merge Sort
    a. Bouw het merge sort algoritme aan de hand van recursie.