Damian Szopiński
Maciej Pestka

Wstęp

Uwaga. W projekcie użyłem biblioteki
(github.com) native-toolkit/libtinyfiledialogs (github.com)
(github.com) pulzed/mINI: INI file reader and writer (github.com)
Pliki .h i .c są już w projekcie w folderze tinylibdiag i simpleini.
Użyłem je, gdyż są to multiplatformowe odpowiedniki i musieliśmy zrezygnować z wbudowanych windowsowych funkcji.

Nasz projekt zawiera dwa programy. Ten który Pan czyta jest z interfejsem graficznym. Drugi jest „czystymi" funkcjami sortującymi i tylko z podstawowymi testami. Implementacje algorytmów sortujących mogą się delikatnie różnić. Interfejs graficzny może trochę spowolnić działanie sortowania ze względu, na dodane dodatkowe instrukcje do funkcji sortujących. Dodatkowo punkt dotyczący funkcji która wybiera szybszy algorytm w zależności od wielkości tablicy nie znajduje się w tym programie, a drugim.

Nasze ustawienia dla kompilatora
Dla tego programu:
Mingw32 ver 2013072200

Link Libraries:
libboost_thread-mgw6-mt-d-x32-1_78.a
libcomdlg32.a

Other linker options:
-lgdi32
-lopengl32
-lcomdlg32
-lole32

Other Compiler Options:
-std=c++17

Informacje

Niektóre implementacje nie zgadzają się z podpunktami z zadania. Oto lista z nich oraz dlaczego postanowiliśmy je zmienić:

- Program zamiast plików txt zapisuje pliki ini, gdyż w ten sposób zapisujemy więcej potrzebnych informacji w bardziej poukładany sposób (kolejność ustawień wzgl. danych). Plik ini zapisuje się w takim formacie:

```
[settings]
method=METODA SORTOWANIA
description=OPIS DO CZEGO SLUZY TABLICA
[0]
key=...
ID=...
[1]...
```

- Według podpunktu 4, program powinien wczytywać dane z wiersza poleceń. U nas większość operacji można zrobić graficznie.
- Dane o liczbie porównań, przypisań itp. Jako jedyne wyświetlają się w konsoli cmd, gdyż interfejs od nadmiaru informacji mógłby stać się nieczytelny.
- Dla każdego wykresu w lewym dolnym rogu pokażą się wartości na którym „stoi" kursor, **dla zobrazowania skali**. NALEŻY JEDNAK POCZEKAĆ NA ZAKOŃCZENIE SORTOWANIA TABLICY/BENCHMARKU. Tablica może wydawać się posortowana wizualnie, ale algorytm sortujący może jeszcze coś liczyć. Należy uzbroić się w cierpliwość.
- Program pozwala na łatwe tworzenie testów. Nawet jeżeli w sprawozdaniu nie będzie wystarczająco informacji o przebytych testach, to być może można taki test uruchomić bezpośrednio w programie.
- **Record.key** program nie testuje wartości ujemnych (nie wiedziałem jak zobrazować wykres słupkowy dla wartości ujemnych), sam sort.h dopuszcza już taką możliwość.
- Najlepiej nie klikać „Back to menu" jeżeli trwa „Waiting". Nie powinno stać się nic złego, ale algorytm może jeszcze chwilę liczyć poprzednie sortowanie zanim będzie można uruchomić kolejne sortowanie.

Dla lepszego zrozumienia

- Dla czytelności kodu, każdy prostokąt (nawet te w wykresach) są jednocześnie przyciskiem.
- Wizualizacja sortowania nie pokazuje wszystkich kroków, tylko najistotniejsze.
- Wizualizacja działa jedynie do 200 elementów, gdy liczba jest przekroczona program sortuje tablice bez niej. Zauważyłem że powyżej tej wartości taka animacja zajmuje za dużo czasu i zdecydowałem się ją od tej wartości ominąć.

Jak korzystać z programu

Jak uruchomić wizualizacje sortowania:

1. Z menu wybierz typ tablicy (pkt 1 dod.), metodę sortowania (pkt 2 dod.) oraz wielkość tablicy (pkt 4 dod. Szczegóły będą później).
   (przycisk 2 000 000 nie oznacza, że jest to maksymalny limit programu. W konsoli jesteśmy w stanie uzyskać większą liczbę)
   Opcja „Console" wczytuje wartość z konsoli CMD, która będzie wielkością tablicy.
2. Naciśnij start
3. Po skończeniu sortowania, po naciśnięciu na dowolnego punktu na wykresie pojawi się jego dokładna informacja (w punkcie czerwonej kropki). Będzie to zastępować skalę.
4. Jeżeli chcemy zobaczyć jak wyglądała pierwotna tabela przed sortowaniem, to klikamy „Save Unsorted". W tym pliku będzie można odczytać zawartość tablicy przed sortowaniem. Analogicznie działa „Save Sorted".

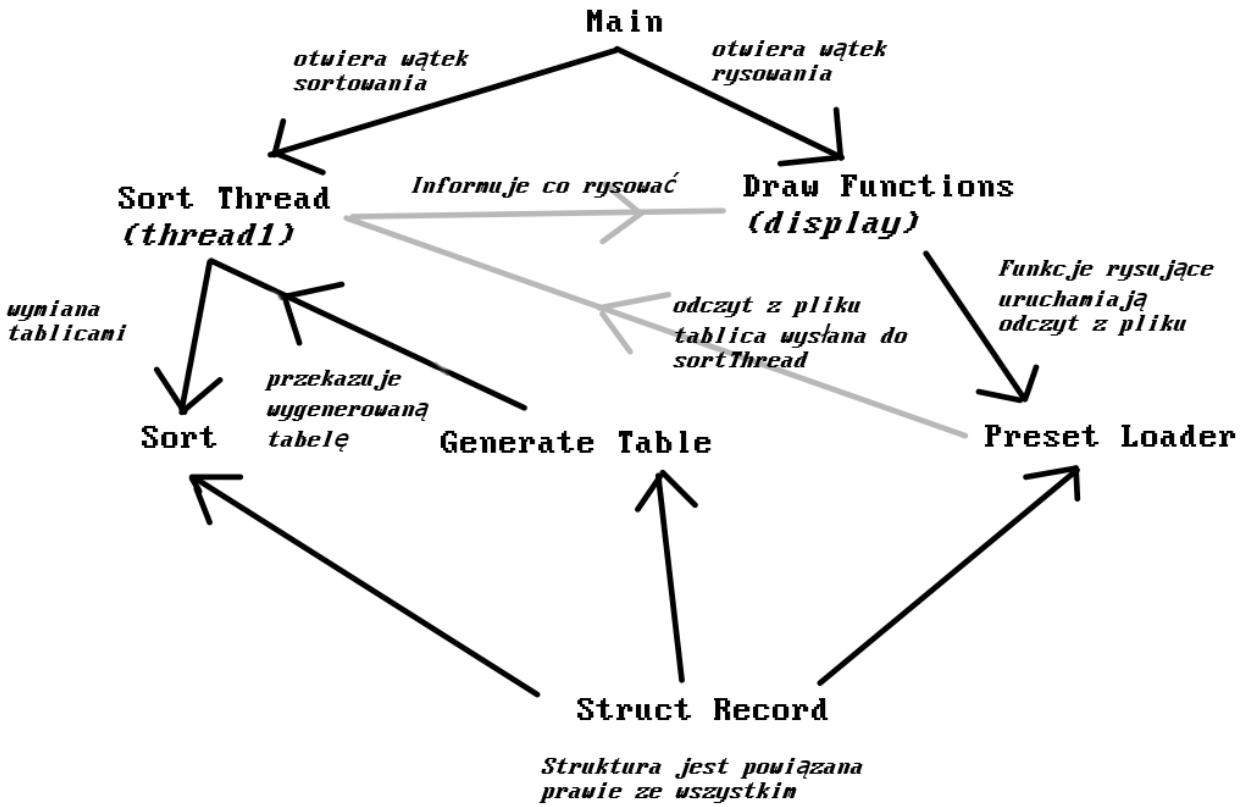
Jak wygenerować wykres czasu sortowania co do wielkości i metody sortowania:

1. W menu należy wybrać typ tablicy jaka będzie sortowana. GDY NIE WYBIERZE SIĘ ŻADNEJ, WYKRES SIĘ NIE WYGENERUJE (BĘDZIE PUSTY)
2. Należy wybrać wielkość kroku z jakim będzie rosła liczba elementów względem szerokości wykresu. („Select size of table") (minimalna wielkość to 10)
3. Naciśnij przycisk „Benchmark" (pkt 8 dod.) i poczekaj aż wykres się wygeneruje
4. Po zakończeniu, gdy naciśnie się na dowolny punkt na wykresie można zobaczyć szczegółowe pozycje (w czerwonej kropce)

Jak uruchomić sortowanie tablicy z pliku:

1. W menu wybierz opcje „Open" (pkt 6 dod. Pliki są w folderze)
2. Wybierz ścieżkę pliku.
3. Na ekranie pojawi się opis (co ma testować ta tablica) oraz wizualizacja tego jak wygląda wykres.
4. Po naciśnięciu na jakiś element można zobaczyć jego key, ID oraz pozycje. (Przyda się to do sprawdzenia czy sortowanie jest stabilne)
5. Naciśnij przycisk „Sort".
6. Na ekranie pokaże się czas sortowania oraz wykres posortowanej tablicy.

Organizacja programu



Po krótce lista zadań każdego z pliku:

- **Main.cpp** tworzy okno, otwiera dwa wątki (pętle główną, dla interfejsu graficznego GLUT (w BasicDrawFunctions.hpp funkcja display()) oraz z biblioteki BOOST „thread" do sortowania (w SortThread.hpp funkcja watek()).

- **SortThread.hpp** zawiera funkcje do obsługi wątku i komunikacji między nimi.
Zawiera również kod testu benchmark.

- Sort.h to wszystkie funkcje sortujące wymagane przez zadanie.

- StructRecord.hpp – Zawiera tylko strukturę Record.

- GenerateTable.hpp jest generatorem tablic (losowej, posortowanej i posortowanej odwrotnie) oraz unikatowych identyfikatorów.

- BasicDrawFunction.hpp zawiera wszystkie istotne funkcje do rysowania. Obsługę przycisków (które nie koniecznie muszą spełniać funkcję przycisku). Oraz obsługę menu głównego. Rysowanie wszystkich wykresów (liniowego i słupkowego).

- PresetLoader.hpp służy do obsługi odczytu i zapisu presetów. Oraz obługi ekplorera plików (przy zapisie i odczycie).

Struktuty:

Utworzone struktury z których będę korzystać w funkcjach

vector2*(DrawFunction.hpp)* – float dwuelementowy (x,y) będzie mi służył do zapisywania pozycji na ekranie

vector3*(DrawFunction.hpp*) – float trzyelementowy (x,y,z) który będzie mi służył między innymi do zapisywania kolorów (x=red,y=green,z=blue)

settingsPreset*(SortThread.hpp)* – zawiera informacje pobrane od użytkownika jak
- SortingMethod*(SortThread.hpp)* (enum który przyjmuje wartości Shell, Quick, Merge, Insertion, notSelectedMethod),
- TableTypes*(GenerateTable.hpp)* (enum który przyjmuje wartości Random, Sorted, ReversSorted, notSelectedType)
- rozmiar tablicy

PresetStruct(PresetLoader.hpp) –to odpowiednik tego co zawiera się generowanym pliku .ini. Czyli znajduje się string będącą metodą tablicy, description czyli opis który pojawi się na ekranie po wczytaniu przez program pliku oraz wartości tabeli czyli vector<Record> (gdzie każdy rekord to id oraz key)

TextBlock *(DrawFunctions.hpp)*- To informacje o prostokącie(przycisku), z tekstem. Zawiera pozycje Poz (vector2), szerokość/wysokość size(Vector2), txt tekstu jaki ma się na nim wyświetlić, oraz makro funkcji doOnClick który wykonuje się w momencie kliknięcia. Hover (bool) to informacja o tym, czy przycisk jest zaznaczony czy nie.
Posiada jeszcze Id przycisku (który nie musi być unikatowy!), oraz SelectionId który działa na zasadzie przycisków „radio” (można zaznaczyć tylko jeden przycisk z grupy prostokątów o takim samym SelectionId. Jeżeli klinknie się na inny przycisk o tym samym selectionId, to pozostałe się odznaczają)

Dopiszę tutaj również jedną klasę, gdyż jest ona ściśle powiązana z poprzednią strukturą.
Klasa blockCollection *(DrawFunctions.hpp)*- przechowuje wektor takich TextBlocków. Posiada instrukcje do narysowania wszystkich prostokątów oraz sprawdzania czy przycisk został naciśnięty.

Get/Set

Funkcje typu get i set (przede wszystkim zapewniają komunikację między wątkami) (threadsort.hpp)

settingPreset preset to ogólne ustawienie wybrane przez użytkownika. To ono decyduje który algorytm sortujący zostanie uruchomiony. Jest ono zmieniane przez następujące funkcje.

- **setSetting(SortingMethod a)** – zmiana ustawienia metody sortowania
- **setSetting(TableTypes a)** – zmiana ustawienia typu tablicy
- **setSetting(long int a)** – zmiana wielości tablicy

BackupTime jest to kopia czasu wykonywania sortowania, która jest przeznaczona do pokazywania na ekranie czasu sortowania tablicy w trybie „Open".

- **int getTime()** – zwraca ten czas

bool processing – posiada informacje czy sortowanie jest w toku czy już jest skończone. Jest bardzo często stosowane podczas wyświetlania (na przykład by określić czy już można pokazać na ekranie czas trwania sortowania, czy jeszcze ono trwa). Dotyczy on też trybu Benchmark.

- **bool getProcessing()** – zwraca boola

vector<Record> Tab - jest to ta gówna tablica która zostanie posortowana

- **vector<Record> getTab()** – zwraca tą tablice
- **vector<Record> TabBeforeSorting** - to jest kopia tablicy przed sortowaniem. Jest ona przechwytywana w momencie zapisywania nieposortowanej tablicy w pliku.

vector<Record> PrintingTable – to jest tablica "do druku" to ona ma się wyświetlać na ekranie. Program musi mieć pewność, że jest ona bezpeiczna (o restriction za chwilę…)

- **void setPrintingTable(vector< Record> tab)** – podmiana tablicy drukowanej
- **vector< Record> getPrintTable()** – zwraca tablicę drukowaną

bool strictDraw – Empirycznie zauważyłem, że gdy jeden proces podmienia lub dodaje dane do tablicy, a drugi w tym czasie próbuje je odczytać, to kończy się to wysypaniem całego programu. Dlatego utworzyłem bool który można nazwać takim to zielonym światłem do odczytywania i rysowania tablicy na ekranie. Informacji która mówi programowi czy może już bezpiecznie narysować nowe dane na ekranie (false - pozwalaj) czy nie (true - zablokuj).
(gdy jest "false" to na ekranie pojawia się napis „Wait…")

- **bool restrictDraw()** – zwraca boola
- **void SetRestrictDraw(bool OnOff)** – funkcje sortujące i benchmark korzystają z tych funkcji w momencie poważniejszych działań na tablicy.

void Start(bool skipSleep) – Informacja o tym, że w najbliższym możliwym momencie zacząć proces sortowania. SkipSleep to informacja czy należy spowalniać sortowanie boost thread_sleep_for. Sortowanie małych tabel trwają co najwyżej kilka mikrosekund, w ten sposób wizualizacja sortowania pokazuje dokładniej swoje kroki.
W tej funkcji zmieniają się zmienne:

- **bool skipSleepState** – pomijaj celowe spowalnianie
- **bool normalStart** – informacja o rozpoczęciu sortowania

void startBenchmark() – informacja o tym, że w najbliższym możliwym momencie zacząć proces sortowania. Zmienia się następująca zmienna.

- **bool startBenchmarkThr**

string description – W każdym pliku .ini znajduje się description. Posiada on informacje co dana tabela ma pokazywać.

- **string getDescription()** – zwraca description. Jest wykorzystana dla funkcji rysującej w trybie „Open”.

stepSizeBenchmark() – zwraca wielkość jendego kroku benchmarku.

Operacje na plikach

(PresetLoader.hpp)

UWAGA! Polskie znaki diakrytyczne nazw plików nie są respektowane!

string openFile() – To jest funkcja która otwiera eksplorer plików (do otwierania pliku). Zostaje tu użyta biblioteka libtinyfiledialogs. Zwraca ścieżkę otwartego pliku. Ta funkcja wypisuje w oknie CMD ścieżkę do pliku w celu upewnienia się, że przechwycono odpowiednie dane.

string saveFile() - To jest funkcja która otwiera eksplorer plików (do zapisywania pliku). Zostaje tu użyta biblioteka libtinyfiledialogs. Zwraca ścieżkę otwartego pliku. Ta funkcja wypisuje w oknie CMD ścieżkę do pliku w celu upewnienia się, że przechwycono odpowiednie dane.

void savePreset(string src,PresetStruct presetStruct) – zapisuje preset z argumentu w ścieżce o podanej w pierwszym argumencie.

PresetStruct openPreset(std::string src) – otwiera plik o podanej ścieżne i zwraca preset.

(SortThread.hpp)

PresetStruct GeneratePresetStruct(string description,bool unsorted) – eksportuje aktualne dane (tablice I jej ustawienia) na presetstruct gotowy do zapisu.
Argument description to opis który ma zostać wpisany do presetu.
Bool unsorted to informacja czy ma być wczytana posortowana czy nieposortowana tablica.

void ApplyPresetStruct(PresetStruct presetStruct) – Preset z argumentu zostaje wdrożony do programu. (Wczytywanie danych)

Komunikacja z użytkownikem

(DrawFunction.hpp)

vector2 MouseBackup – kopia zapasowa dla skali. Nawet, gdy nie kilkamy chcemy by skala się pokazywała.

void getMouse (int button, int state,int x, int y) – informacje openGL o naciśnięciu przycisku myszy. Wywołuje BoxCollection.checkIfClick oraz kopiuje pozycje myszy do MouseBackup. Argumenty : buton – typ przycisku / state – down/up / x,y pozcja myszy.

Klasa BoxCollection (lista przycisków)

- **checkIfClick(vector2 mouse)** – sprawdza czy któryś z podanych przycisków został naciśnięty (O ile selectionID jest różny od „-1”. Jest to wyjątek bo nie wszystkie prostokąty chcemy klikać). Jeżeli tak, to zaznacza ten przycisk (zmienia hover). Są to przyciski radio czyli tylko jeden przycisk z danego selectionId można zaznaczyć.

Odczytywanie z konsoli (wielkość tablicy. Przycisk „Console”)

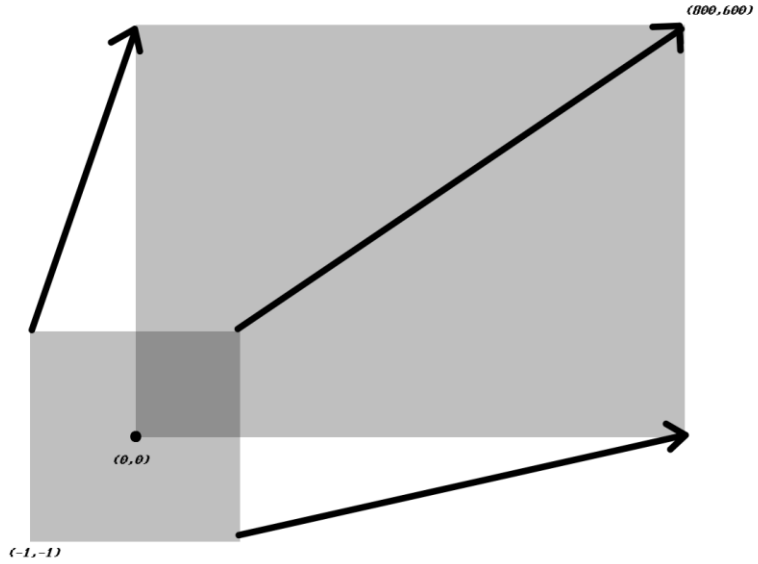
Void getNumber() – rozpoczyna nowy wątek cinInput, by okno aplikacji nie stanęło w miejscu i nie czekało na odpowiedź użytkownika.

size_t inputGetNumber=0 – to jest liczba odczytana od użytkownika. Jeżeli „-1”, to program wypisuje na ekranie, że czeka na odpowiedź od użytkownika („Input value in console”)

void cinInput() – (zamknięty w cpp) proces cin który nadpisuje inputGetNumber.

Rysowanie

vector2 convertSize(float x,float y) – skaluje prostokąt [0,0]x[800,600] na kwadrat [-1,-1]x[1,1]. Zwraca odpowiedni punkt z podanego x,y łatwy do odczytania przez openGL.
Jest to dość częsta praktyka. Operowanie na małych liczbach jest ciężkie. Wygląd programu został narysowany przez nas w formacie 800x600, przez co bardzo mi to ułatwiło przeniesienie tego na kod.
Graficzna interpretacja: (punkt w kwadracie [-1,-1]x[1,1] odpowiada punktowi na prostokącie [0,0]x[800,600])



float autoScale(const vector<Record> &Tab) – znajduje element z największym kluczem i zwraca optymalną wartość skali według której wszystkie wartości powinny znaleźć się na ekranie.

void drawTable(const std::vector<Record> &Tab,bool onlyLines,bool autoScaleMe,int columns,vector3 color) – instrukcje rysujące tablice. Tutaj duże znaczenie mają argumenty:
Tab – tablica która ma zostać narysowana
onlyLines – Jeżeli false to narysuje się wykres słupkowy, jeżeli true to narysuje wykres liniowy.
AutoScaleMe – Czy wykres ma się skalować automatycznie, czy należy to robić manualnie (Zmienna Scale jest za to odpowiedzialna)
Colums – Gdy -1 to pokaż tyle kolumn ile jest w tabeli elementów, jeżeli nie to pokaż tyle kolumn ile podano
Color – Kolor wykresu liniowego (słupkowy nie potrzebował takiej funkcjonalności)

void drawBenchmarkResult() – rysuje kilka wykresów liniowych na raz. Benchmark pokazuje 4 wykresy odpowiadające dla każdego typu sortowania.
ID dla wykresów liniowych nie są unikatowe, w benchmarku ID recordu to informacja dla tej funkcji o kolorze na jaki ma się pomalować wykres.

int sortingTime – kopia czasu sortowania. Jest łatwo dostępny dla funkcji rysującej. Ten czas jest rysowany na ekranie.

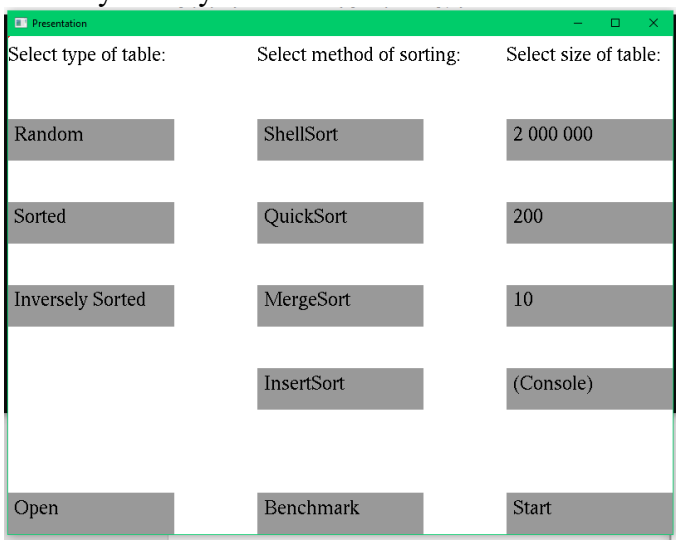
void drawString(const char* txt, vector2 pos,vector3 color,vector2 offset) – rysuje na ekranie tekst. O treści txt, pozycji (pos-offset) oraz kolorze color.

void Circle(float r,float pos_x,float pos_y) – rysuje czerwony okrąg o promieniu r, pozycji x,y. Czerwona kropa będzie interpretować skalę. Po naciśnięciu punktu na wykresie pojawi się na ekranie jakie wartości są przyjmowane w tym miejscu. Do tego posłużą także zmienne:
float scaleX,scaleY;
Które mówią jak należy interpretować jednostkę przypadająca na piksel x i y.

Klasa BoxCollection

- **Void drawAll()** – rusuje na ekranie wszystkie przyciski z tej klasy. (prostokąt + tekst). Jeżeli text==”” to kolor prostokąta będzie zielony. Jeżeli nie, to jeżeli przycisk jest naciśnięty to będzie ciemno siwy, jeżeli nie będzie naciśnięty to jasno siwy.
- **void addNew(textBlock nowy)** – dodaje kolejny prostokąt „nowy” do swojej listy.

void onStart() – Rysuje menu główne i zeruje niepotrzebne wartości, by nie zostawiać za sobą nieaktualnych danych.



void display() – główna funkcja rysująca OpenGL. Tak na dobrą sprawę odpowiada za większość rzeczy rysujące się na ekranie.
Tutaj istotne są zmienne:

alreadySorted – czy można na ekranie wyświetlić wynik sortowania?
openMenuActive – czy rysować tryb “Open”? (odczyt z pliku)
benchState – czy przejść do ekranu rysowania wykresu Benchmark?
menuActive – Czy menu aktualnie jest aktywne?
DefaultStart – Czy zostało uruchomione normalne sortowanie funkcji z wizualizacją lub bez?

Generator Tablic

(generateTable.hpp)

string getUniqueId(const size_t &num) – generuje unikalne id na bazie numeru num. (Jak pewien podpunkt z zadania tego wymagał).
Jest to na swój sposób zapisywane jako system liczbowy 36-owy.

void generateTable(vector<Record> &tab,TableTypes type,long int size) – generuje odpowiednią tablice typu “type” I wielkości size.

Funkcje sortujące

Idea wszystkich funkcji sortujących jest prawie taka sama jak w drugim programie. Zamiast pisać drugi raz to samo, napiszę tylko, to czym się różnią:

Wzbogaciłem algorytm *sort.cpp* o o dwie funkcje:

void FunctionFinish(vector <Record> &records) – nadpisuje tablice PrintingTable na “records” I nakłada blokadę restrict.

void FuntionForPrinting(bool threadSleep,vector <Record> &records) – Jeżeli program jest w trybie spowolnienia threadSleep, to wykonaj poprzednią funkcję I zatrzymaj czas na chwilkę, dla pokazania lepszej wizualizacji kroków.

Zmienione zostały funkcje Diag. To one zostały poświęcone do pokazywania wizualizacji. Zostały wzbogacone o nowy argument threadSleep (które celowo spowalniają program w celu lepszego pokazywania kroków). Wykonywane są te dwie funkcje wymienione wcześniej.

Funkcje sterujące zadaniami

(sortThread.cpp)

Void thread1() – to wieczna pętla. Wykonuje te polecenia które zadały im funkcje Get/Set. W przypadku złych danych, funkcja kończy działanie oraz pokazuje komunikat o błędzie.
Generuje tablice korzystając z wcześniejszych funkcji. Sortuje na żądanie. Zapisuje czasy.
Wypisuje w oknie CMD odczytaną liczbę operacji.
Od wielkości 200 wizualizacja nie zostaje odtworzona.
Teoretycznie dla tablic >= 200 elementów, sortowanie następuje dwa razy. Diag, które pokazuje na ekranie i wypisuje dane, oraz zwykła funkcja sortująca bez udziwnień. Wizualizacja może trwać kilka sekund, ale na ekranie pokaże się, że sortowanie trwało mikrosekundy ponieważ pokazywany jest czas tylko tego drugiego sortowania. (Przed przejściem na kolejne sortowanie zamieniane jest SortedTable na Unsorted Table, by dane były w obu przypadkach takie same)

void Benchmark() – Bywa wywoływane przez thread1. W przypadku złych danych, funkcja kończy działanie oraz pokazuje komunikat o błędzie.
Dla każdej z 4 funkcji sortujących po 100 razy (liczba próbek).
vector< vector< Record>> I ta konstrukcja jest Tablicą Tablic Rekordów. Rekordy są tu interpretowane jako record.key=czas w mikrosekundach oraz record.ID=kolor wykresu.
Wyświetla w oknie CMD informacje o rozpoczęciu i zakończeniu.

Lista komunikatów o błędach:

"Type of table was not selected, try again" – Podczas próby uruchomienia benchmarku, nie został zaznaczony typ tablicy jaki ma sortować tablica. (Czy ma być losowa, posortowana, czy odwrotnie posortowana)

"Invalid step size" – Dla benchmarku nie wybrano wielkości kroku (albo wielkość jest <10), który wybierany jest za pomocą „Select size of table”.

"Type of table/size is not selected, or is empty" – wygenerowana tablica jest pusta. Zapewne wielkość tablicy nie została zaznaczona, albo otwierany plik ini jest niepoprawny.

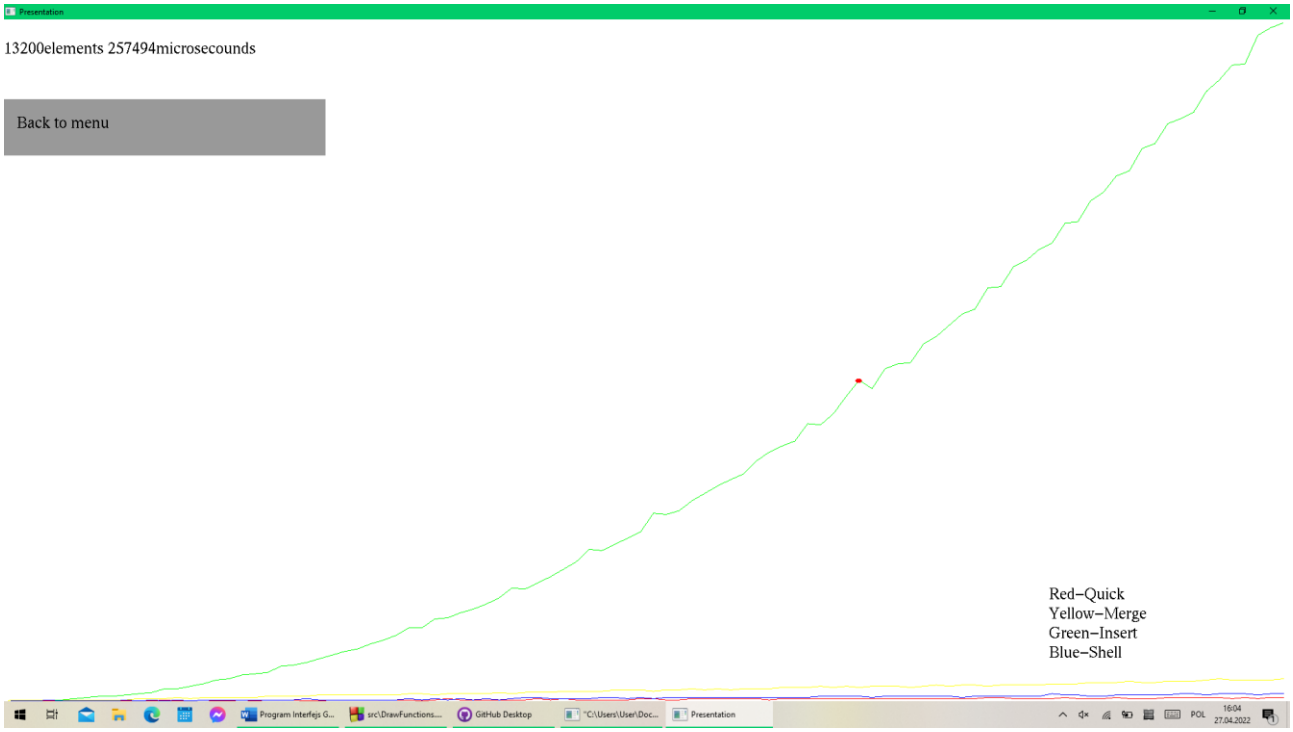
"Method is not selected" – Metoda sortowania nie jest oczywista dla programu. Albo nie została wybrana. Albo plik .ini zawiera nieistniejący sposób sortowania (Może wielkość liter jest niepoprawna, poprawne są jedynie formy „Shell”,”Quick”,”Merge”,” Insertion”).

"Can not open the file" – Albo wyłączyłeś eksplorer plików zanim plik został wybrany, albo program nie posiada dostępu do tego pliku (plik i jego podfoldery nie mogą mieć polskich znaków).

Nasze wyniki

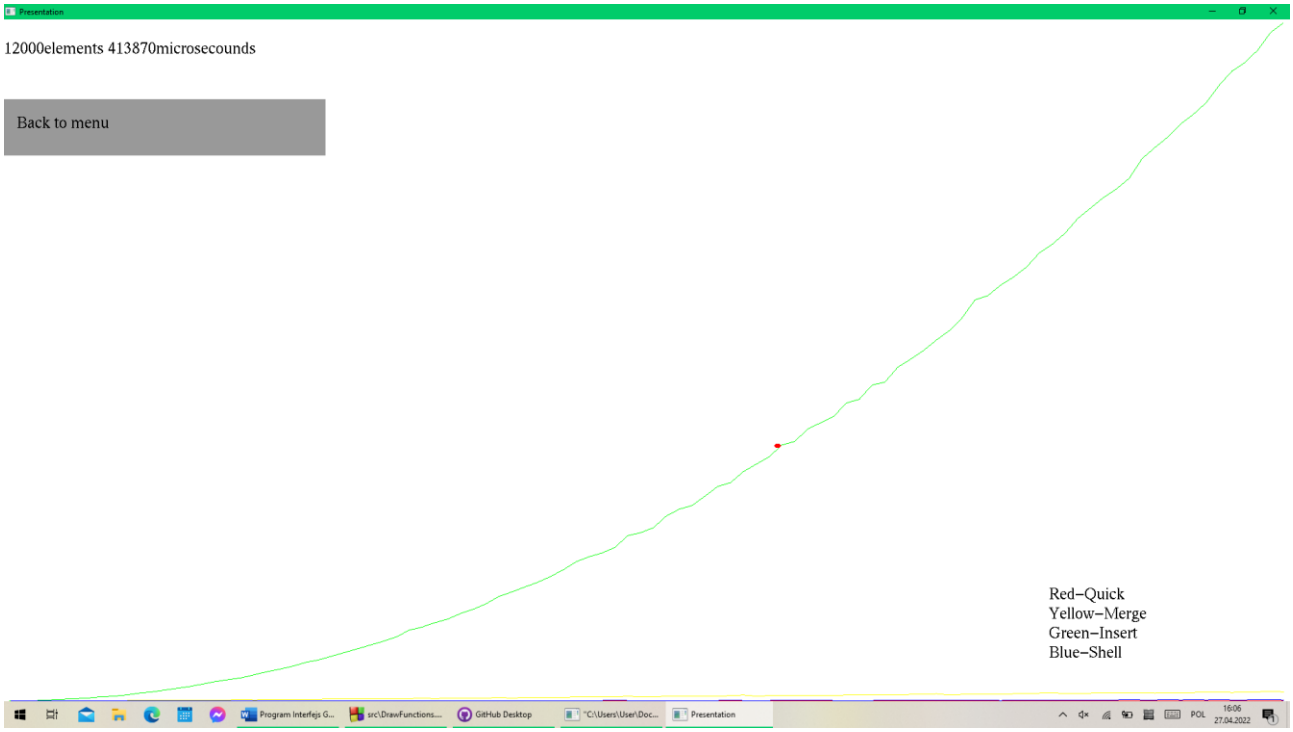
Testy będą przeprowadzone na procesorze Ryzen 5800h
Dla pokazania skali jednak najlepiej jest wykonać te testy na swoim komputerze, ponieważ teraz jedynie mogę pokazać zrzuty ekranu. Postaram się jednak by wnioski były bardziej uniwersalne.
Każdy wykres jest zbudowany z 100 próbek.

Tablica losowa, od 100 do 20 000 elementów. Od 0 do 547309 mikrosekund.



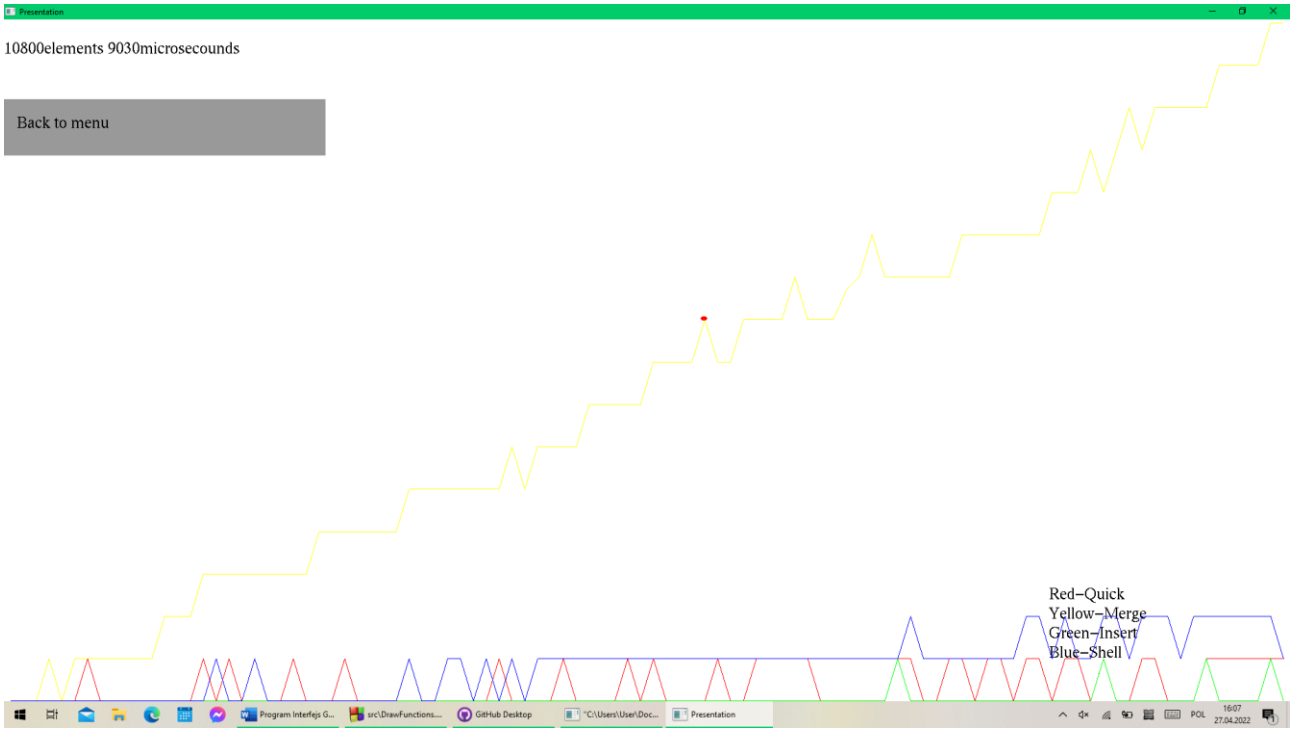
Widzimy tutaj jak duże znaczenie ma dobór odpowiedniej metody sortowania. Dla większej ilości elementów sortowanie Insertion jest nieopłacalne.

Tutaj Tablica odwrotnie posortowana, od 100 do 20 000 elementów. Od 0 do 1104742 mikrosekund.



Jak można było się spodziewać Insertion sort ponownie trwał najdłużej. Różnica czasu między pozostałymi sortowaniami powiększyła się jeszcze bardziej.

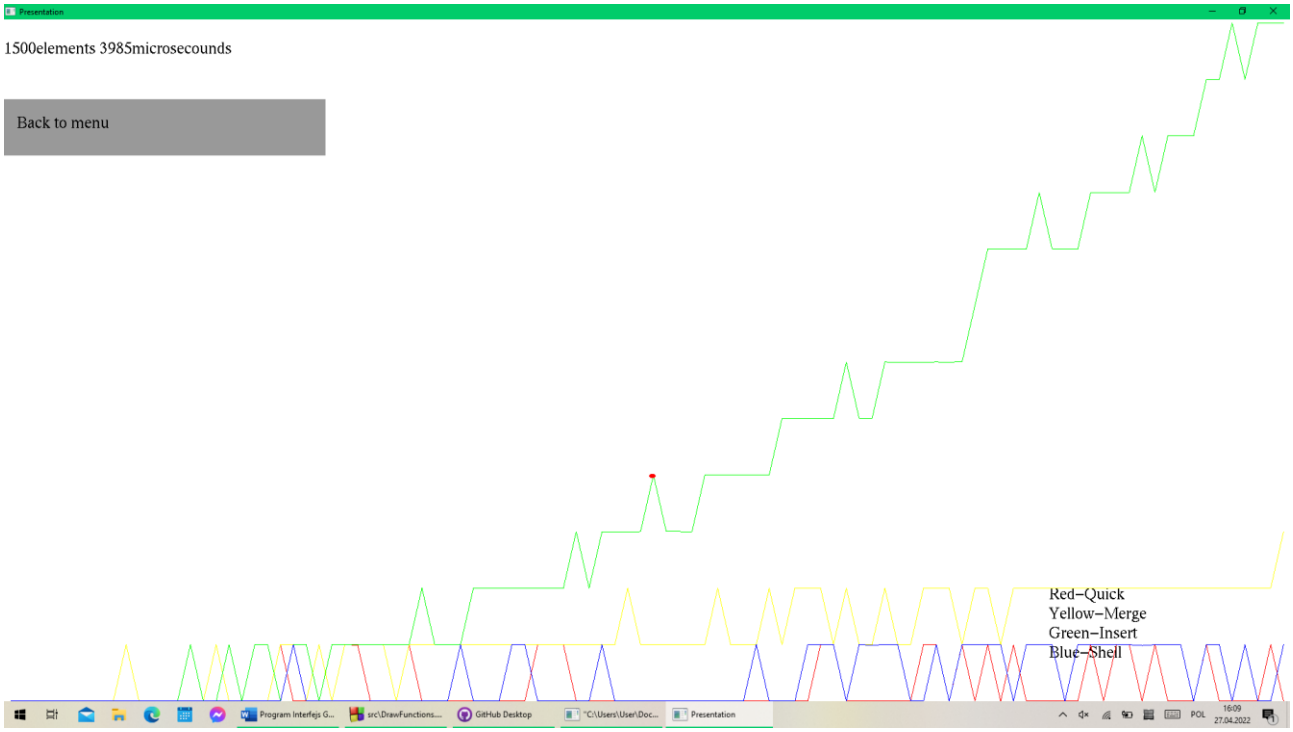
Tutaj Tablica posortowana, od 100 do 20 000 elementów. Od 0 do 16084 mikrosekund.



Tutaj trwało to dużo krócej. Zauważamy, że Merge sort tutaj trwa najdłużej. Insert w przeciwieństwie do poprzedniego wykresu wyprzedził wszystkich.

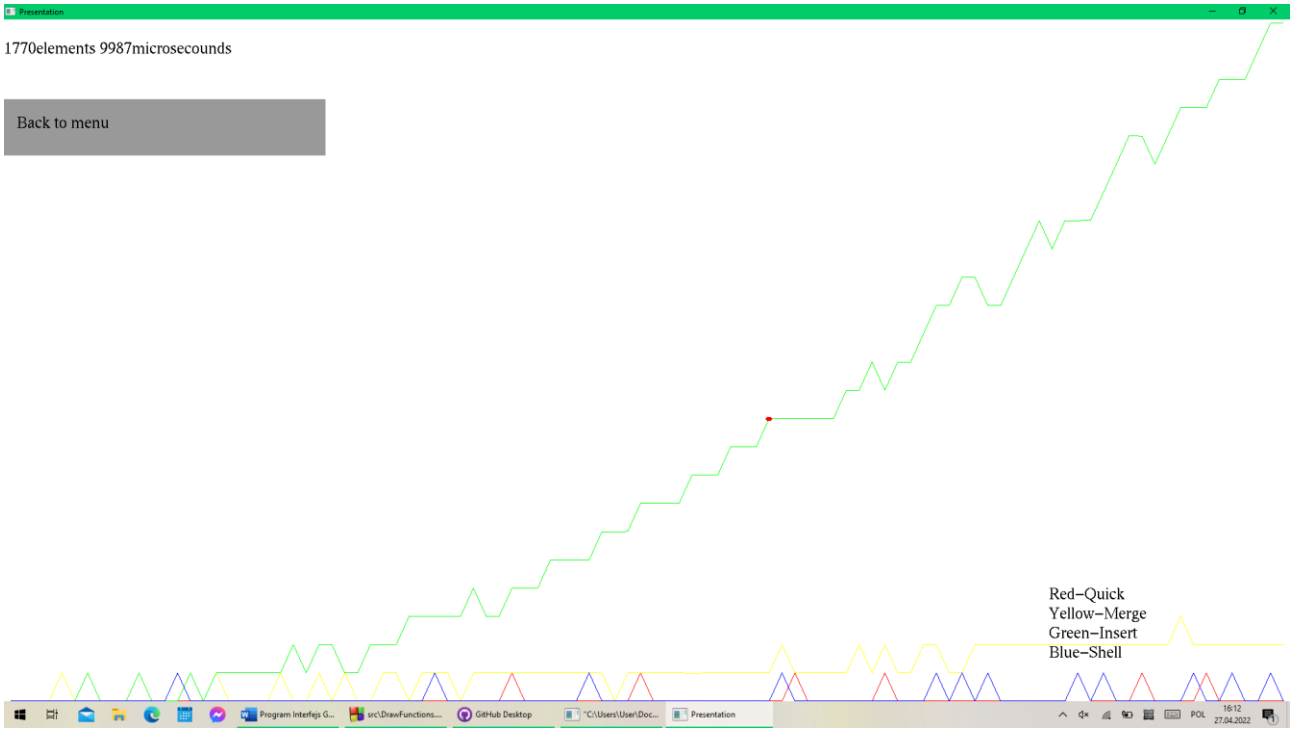
Sprawdźmy teraz jak to wygląda dla mniejszych zbiorów.

Tutaj Tablica losowa, od 30 do 3 000 elementów. Od 0 do 12051 mikrosekund.



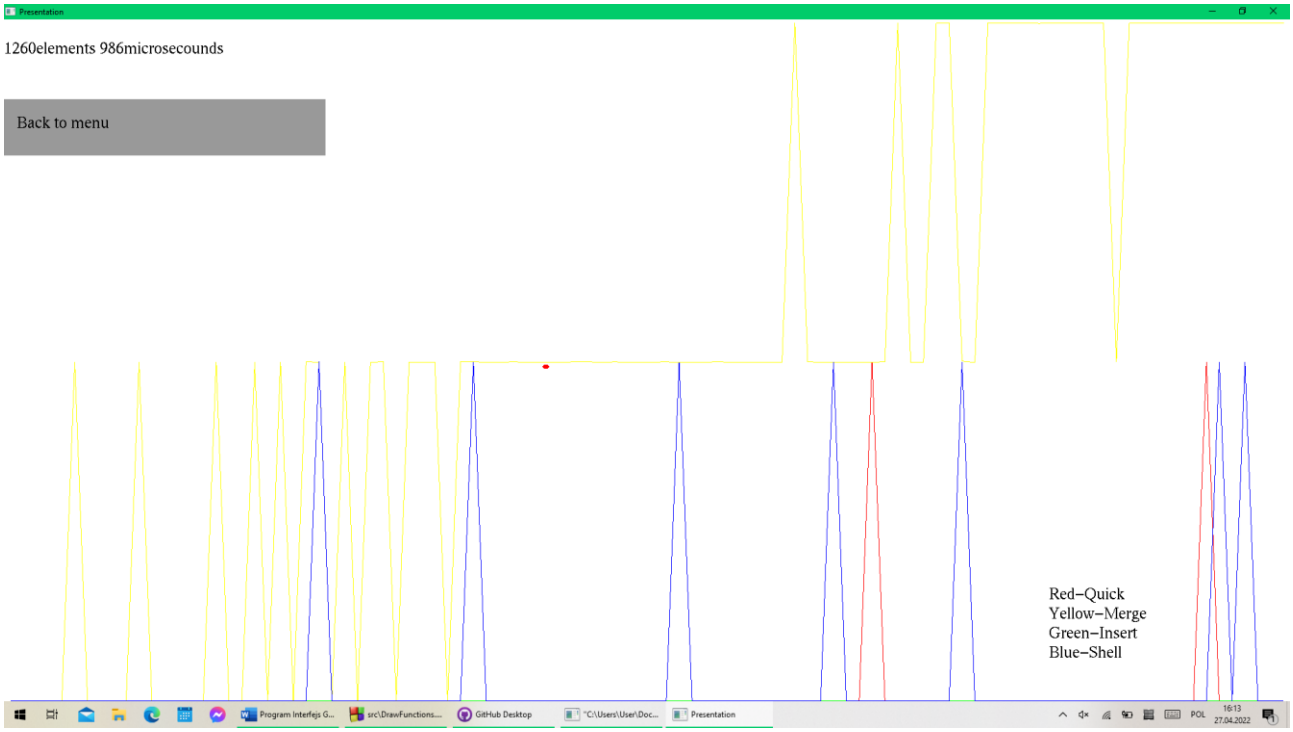
Przy mniejszych tablicach możemy zauważyć, że Inserion sort potrafi być szybsze (przy wielkości rzędu <100 el) niż np. Merge.

Tablica posortowana odwrotnie, od 10 do 1 000 elementów. Od 0 do 24102 mikrosekund.



Wyniki można było przewidzieć.

Tutaj Tablica posortowana, od 10 do 1 000 elementów. Od 0 do 2011 mikrosekund.



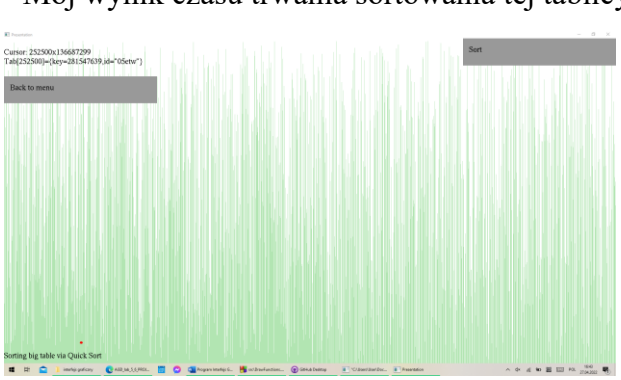
Ponownie możemy zauważyć, że Merge sort nie jest najlepszy do sortowania posortowanych już tablic. Najlepiej tutaj spisał się Insert.

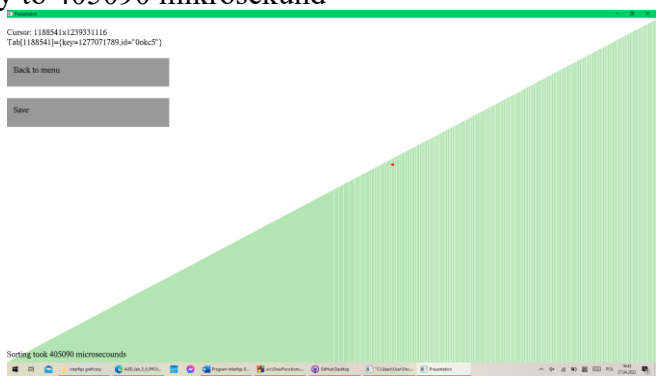
Przygotowane przykładowe tablice z pliku

Pierwszy plik **Zestaw 01** posłuży nam do tego, by zauważyć, że sortowanie Quick Sort nie jest stabilne.
Jest to przykład takiej tabeli gdzie stabilność nie działa.
Istnieją tam 3 wartości przyjmujące key=2; są one odpowiednio w kolejności (o takim samym ID) 2,3,5. Po posortowaniu tablicy otrzymujemy je w kolejności 5,3,2.

Zestaw 2 i 3 pokazuje nam tablicę prawie posortowaną (punkt 6) (z wyjątkiem elementu na początku i na końcu).
Wiemy już z testów, że Inserion sort poradzi sobie z nimi najszybciej, a Merge Sort najgorzej.
Dla takiej samej tablicy (10 000 elementów)
- Insertion trwa 1000 mikrosekund
- Merge trwa 8002 mikrosekund

Wiemy, że u nas statystycznie Quick Sort jest najszybszy przy największych tablicach. **Zestaw 4** posiada 2 000 000 elementów losowych. (Uwaga, wczytywanie tej tablicy trochę trwa)
- Mój wynik czasu trwania sortowania tej tablicy to 405090 mikrosekund





Zestaw 5 i 6 to pokazanie różnicy czasu między sortowaniem Insert i Quick dla tablicy odwrotnie posortowanej. Testujemy 100 000 elementów.
- Quick Sort 4994 mikrosekund
- Insertion Sort 27213310 mikrosekund (Trzeba poczekać chwilkę)