1. ***Wizualizacja działania algorytmów sortowania bąbelkowego i quicksort.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na pracę krokową lub ciągłą. Wynik działania programu powinien być wyświetlany na ekranie z możliwością zapisu/odczytu do/z pliku tekstowego. Dane wykorzystywane podczas sortowania (działania programu) powinny być wczytywane z **klawiatury**, **generowane losowo** lub z **pliku** (program powinien posiadać opcję wyboru *„menu”*). Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

2. ***Wizualizacja działania algorytmów sortowania przez wybór i heapsort.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na pracę krokową lub ciągłą. Wynik działania programu powinien być wyświetlany na ekranie z możliwością zapisu/odczytu do/z pliku tekstowego. Dane wykorzystywane podczas sortowania (działania programu) powinny być wczytywane z **klawiatury**, **generowane losowo** lub z **pliku** (program powinien posiadać opcję wyboru *„menu”*). Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

3. ***Wizualizacja działania algorytmów sortowania przez wstawianie i mieszane.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na pracę krokową lub ciągłą. Wynik działania programu powinien być wyświetlany na ekranie z możliwością zapisu/odczytu do/z pliku tekstowego. Dane wykorzystywane podczas sortowania (działania programu) powinny być wczytywane z **klawiatury**, **generowane losowo** lub z **pliku** (program powinien posiadać opcję wyboru *„menu”*). Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

4. ***Wizualizacja działania algorytmów sortowania przez scalanie i Shella.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na pracę krokową lub ciągłą. Wynik działania programu powinien być wyświetlany na ekranie z możliwością zapisu/odczytu do/z pliku tekstowego. Dane wykorzystywane podczas sortowania (działania programu) powinny być wczytywane z **klawiatury**, **generowane losowo** lub z **pliku** (program powinien posiadać opcję wyboru *„menu”*). Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

5. ***Wizualizacja działania algorytmów sortowania gnoma i bibliotecznego.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na pracę krokową lub ciągłą. Wynik działania programu powinien być wyświetlany na ekranie z możliwością zapisu/odczytu do/z pliku tekstowego. Dane wykorzystywane podczas sortowania (działania programu) powinny być wczytywane z **klawiatury**, **generowane losowo** lub z **pliku** (program powinien posiadać opcję wyboru *„menu”*). Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

6. ***Wizualizacja działania algorytmu Naiwnego (Bruteforce) wyszukiwania wzorca w tekście.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wczytywania poszukiwanego wzorca z **klawiatury**, **losowo** oraz grupy wzorców z **pliku**. Tekst przeszukiwany powinien być również wczytywany z pliku tekstowego i wyświetlany na ekranie. Wyszukiwanie powinno odbywać się **krokowo** lub **ciągle** (opcja wyboru w programie), każde wystąpienie wzorca powinno być podświetlone. Program powinien wyświetlać statystykę wystąpień danego wzorca (wzorców) w danym tekście. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

7. ***Wizualizacja działania algorytmu Knutha-Morrisa-Pratta wyszukiwania wzorca w tekście.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wczytywania poszukiwanego wzorca z **klawiatury**, **losowo** oraz grupy wzorców z **pliku**. Tekst przeszukiwany powinien być również wczytywany z pliku tekstowego i wyświetlany na ekranie. Wyszukiwanie powinno odbywać się **krokowo** lub **ciągle** (opcja wyboru w programie), każde wystąpienie wzorca powinno być podświetlone. Program powinien wyświetlać statystykę wystąpień danego wzorca (wzorców) w danym tekście. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

8. ***Wizualizacja działania algorytmu Boyera-Moora wyszukiwania wzorca w tekście.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wczytywania poszukiwanego wzorca z **klawiatury**, **losowo** oraz grupy wzorców z **pliku**. Tekst przeszukiwany powinien być również wczytywany z pliku tekstowego i wyświetlany na ekranie. Wyszukiwanie powinno odbywać się **krokowo** lub **ciągle** (opcja wyboru w programie), każde wystąpienie wzorca powinno być podświetlone. Program powinien wyświetlać statystykę wystąpień danego wzorca (wzorców) w danym tekście. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

9. ***Wizualizacja działania algorytmu Aho-Corasicka wyszukiwania wzorca w tekście.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wczytywania poszukiwanego wzorca z **klawiatury**, **losowo** oraz grupy wzorców z **pliku**. Tekst przeszukiwany powinien być również wczytywany z pliku tekstowego i wyświetlany na ekranie. Wyszukiwanie powinno odbywać się **krokowo** lub **ciągle** (opcja wyboru w programie), każde wystąpienie wzorca powinno być podświetlone. Program powinien wyświetlać statystykę wystąpień danego wzorca (wzorców) w danym tekście. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

10. ***Wizualizacja działania algorytmu Rabina-Karpa wyszukiwania wzorca w tekście.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wczytywania poszukiwanego wzorca z **klawiatury**, **losowo** oraz grupy wzorców z **pliku**. Tekst przeszukiwany powinien być również wczytywany z pliku tekstowego i wyświetlany na ekranie. Wyszukiwanie powinno odbywać się **krokowo** lub **ciągle** (opcja wyboru w programie), każde wystąpienie wzorca powinno być podświetlone. Program powinien wyświetlać statystykę wystąpień danego wzorca (wzorców) w danym tekście. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

11. ***Wizualizacja algorytmów fraktali zbioru Mandelbrot'a oraz zbioru Julii.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wizualizację wybranego algorytmu fraktali. Dodatkowo program powinien posiadać możliwość definiowania oraz zmiany parametrów algorytmów, z możliwością porównania obu wizualizacji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

12. ***Wizualizacja algorytmów fraktali zbioru Mandelbrot'a wyższych rzędów oraz zbioru Julii wyższych rzędów.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wizualizację wybranego algorytmu fraktali. Dodatkowo program powinien posiadać możliwość definiowania oraz zmiany parametrów algorytmów, z możliwością porównania obu wizualizacji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

13. ***Wizualizacja algorytmów fraktali zbioru Phoenix Mandelbrot'a oraz zbioru Phoenix Julia.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wizualizację wybranego algorytmu fraktali. Dodatkowo program powinien posiadać możliwość definiowania oraz zmiany parametrów algorytmów, z możliwością porównania obu wizualizacji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

14. ***Wizualizacja algorytmów fraktali Buddhabrot oraz Newtona.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na wizualizację wybranego algorytmu fraktali. Dodatkowo program powinien posiadać możliwość definiowania oraz zmiany parametrów algorytmów, z możliwością porównania obu wizualizacji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

15. ***Implementacja algorytmów kompresji danych RLE (Run Length Encoding) oraz ByteRun.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na kompresję pliku lub grupy plików. Dodatkowo program powinien posiadać możliwość dekompresji skompresowanych plików oraz możliwość zmiany parametrów kompresji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

16. ***Implementacja algorytmu kompresji danych z wykorzystaniem kodowania Huffmana.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na kompresję pliku lub grupy plików. Dodatkowo program powinien posiadać możliwość dekompresji skompresowanych plików oraz możliwość zmiany parametrów kompresji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

17. ***Implementacja algorytmu kompresji danych LZMA.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na kompresję pliku lub grupy plików. Dodatkowo program powinien posiadać możliwość dekompresji skompresowanych plików oraz możliwość zmiany parametrów kompresji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

18. ***Implementacja algorytmu kompresji danych GZIP.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na kompresję pliku lub grupy plików. Dodatkowo program powinien posiadać możliwość dekompresji skompresowanych plików oraz możliwość zmiany parametrów kompresji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

19. ***Implementacja algorytmu kompresji danych Bzip2.***

Program powinien posiadać graficzny interfejs użytkownika (ang. GUI) pozwalający na kompresję pliku lub grupy plików. Dodatkowo program powinien posiadać możliwość dekompresji skompresowanych plików oraz możliwość zmiany parametrów kompresji. Całość należy wykonać w języku C/C++. W celu stworzenia interfejsu użytkownika należy wykorzystać jedną z bibliotek do tworzenie interfejsu graficznego **wxWidgets**, **Qt**, **Ultimate++** lub **GTK+**.

20. ***Gra „Tetris”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

21. ***Gra „Miny” (saper).***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

22. ***Gra „Wąż” (snake).***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

23. ***Gra „Space Invaders”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

24. ***Gra „Memory”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

25. ***Gra „Arkanoid”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

26. ***Gra „Frogger”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablica wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

27. ***Gra „Pacman”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

28. ***Gra „Mini-putt” (gra w mini-golfa).***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

29. ***Gra w „Szachy”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Możliwość gry pojedynczo, w 2 osoby lub z komputerem. Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

30. ***Gra w „Warcaby”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Możliwość gry pojedynczo, w 2 osoby lub z komputerem Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.

31. ***Gra w „Statki”.***

Gra powinna posiadać przyjazny interfejs graficzny wraz z **opisem gry (help)** oraz **obsługą klawiatury**, **licznik punktów**, **tablicą wyników**. Możliwość zapisu i odczytu statystyk najlepszych graczy do pliku oraz możliwość zdobywania bonusów lub punktów dodatkowych. Możliwość gry pojedynczo, w 2 osoby lub z komputerem Całość należy wykonać w języku C/C++. W celu stworzenia gry można wykorzystać bibliotekę **Allegro** lub **SFML**.