Projekt 2 – sortowanie

Wiktor Zmiendak gr. 13

1. Opis Zadania:

Zadanie projektowe polega na zaimplementowaniu 6 różnych algorytmów sortujących tablice, a następnie przetestowaniu i porównaniu ich sposobu działania. Testy będą przeprowadzane dla trzech rodzajów danych wejściowych typu integer z zakresu [-100 ; 100]: losowo wygenerowanych, posortowanych oraz posortowanych odwrotnie.

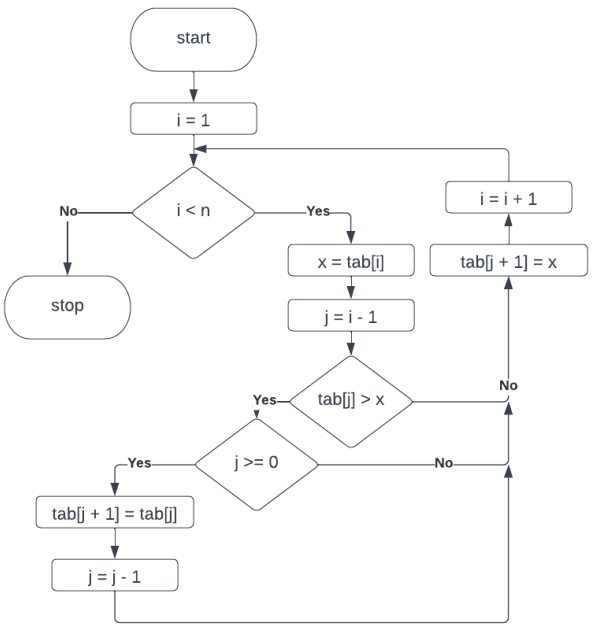
2. Zawartość projektu:

Do wykonania zadania został wykorzystany plik Algorithms.c zawierający algorytmy sortujące, Algorithms.h będący headerem oraz plik Main.c służący do wywoływania poszczególnych algorytmów przy pomocy prostego interfejsu użytkownika. Po uruchomieniu program prosi użytkownika o wybór algorytmu oraz o wielkość tablicy którą ma stworzyć. Następnie do pliku dane.txt generowanych jest „n" wartości, które w dalszym kroku są pobierane do tablicy „tab" i sortowane przy pomocy wybranego algorytmu. W celu przeprowadzenia kolejnego testu program bierze posortowaną tablicę, zapisuje ją do dane.txt oraz sortuje ją jeszcze raz. Na koniec program odwraca tablicę i sortuje ją po raz ostatni. Po każdej próbie sortowania użytkownik jest informowany o czasie w jakim dokonano dane sortowanie.

3. Algorytmy:

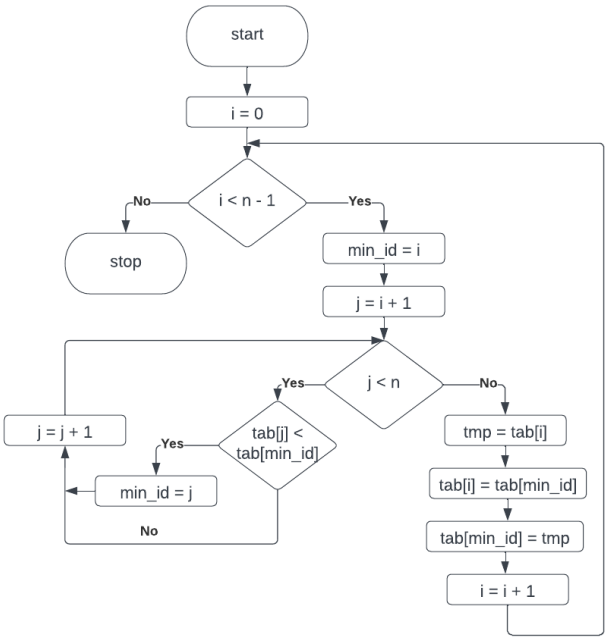
Grupa I

Insertion sort:
Polega on na wstawianiu kolejnych wartości z brzegu tablicy w odpowiednie miejsca między dwoma wartościami tym samym przesuwając wszystkie pozostałe wartości na lewo od siebie o jedną pozycję. Jego złożoność wynosi $O(n^2)$ lub $O(n)$ w przypadku optymistycznym.



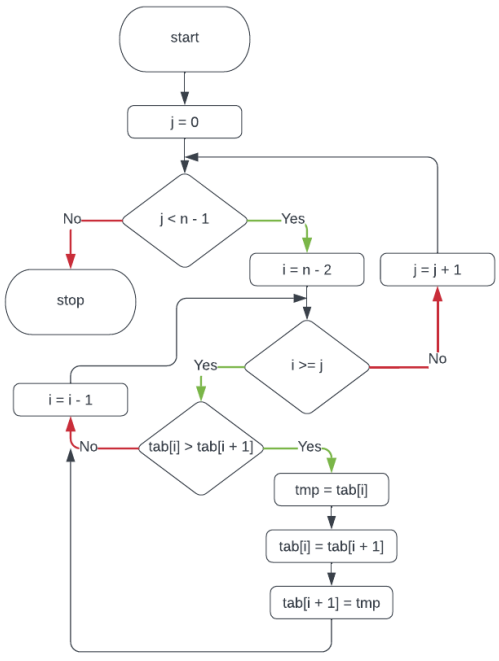
Selection sort:

Polega on na wyszukiwaniu obecnie najmniejszej wartości w tablicy i zamianie jej z wartością najbardziej na lewo. W ten sposób powoli budujemy posortowaną tablicę od lewej strony. Jego złożoność wynosi $O(n^2)$.



Bubble sort:

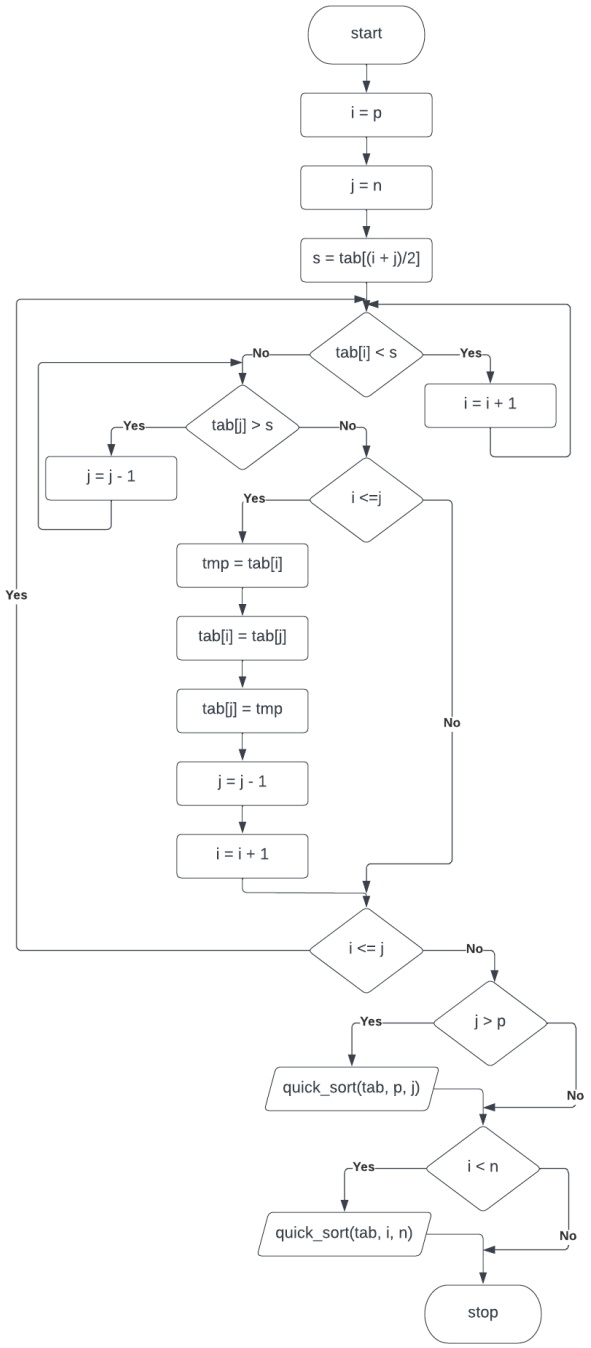
Polega na porównaniu dwóch kolejnych elementów w tablicy. Jeżeli wartość pierwsza jest większa od drugiej to następuje ich zamiana miejsc. Algorytm kończy działanie jeżeli podczas przejścia przez tablicę nie wykonano żadnej zamiany. Jego złożoność wynosi $O(n^2)$.



Grupa II

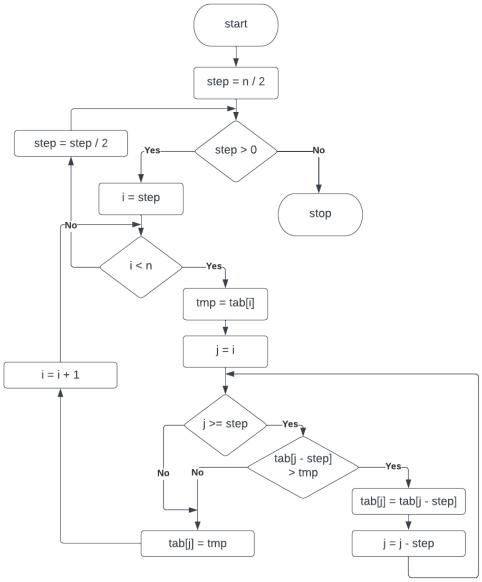
Quick sort:

Z tablicy wybierany jest element rozdzielający nazywany pivotem, po czym tablica dzielona jest na dwa fragmenty. Do pierwszego fragmentu lądują elementy mniejsze od naszego pivota, a do drugiego te większe. W następnym kroku osobno sortujemy oba fragmenty stosując tą samą technikę jak na początku. Jest to algorytm rekurencyjny którego złożoność obliczeniowa wacha się pomiędzy $O(n^2)$ w przypadku pesymistycznym, a $O(nlogn)$ w przypadku optymistycznym.



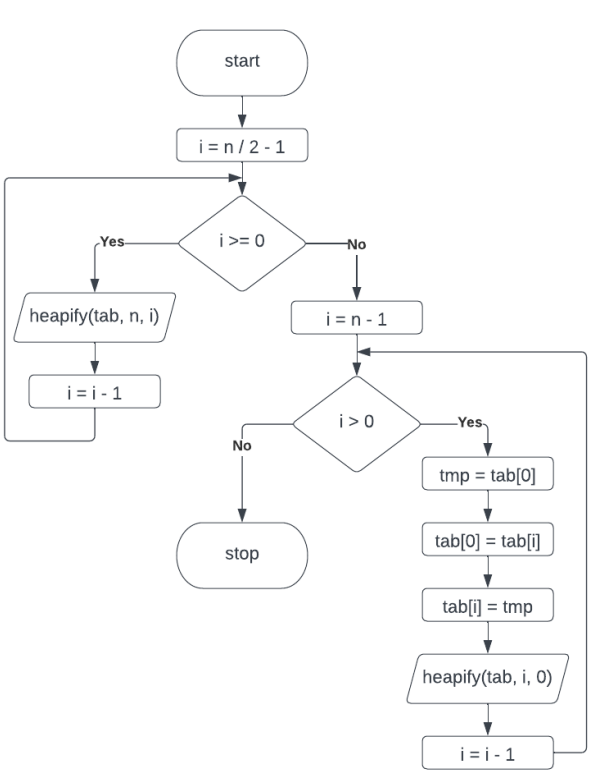
Shell sort:

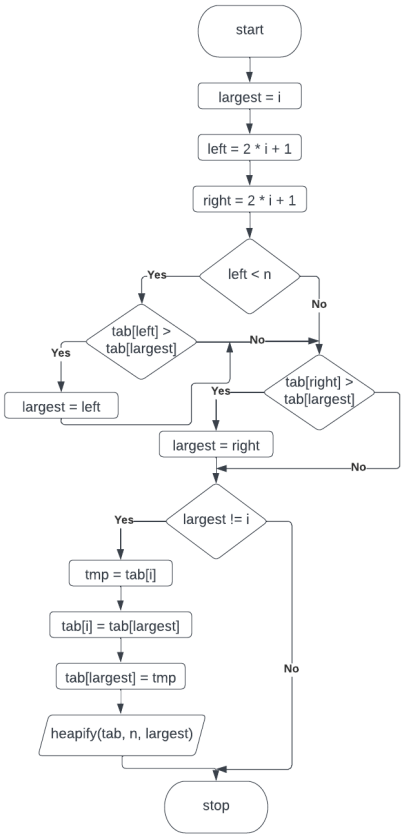
Algorytm ten działa w sposób bardzo podobny do insertion sort. Różnica polega na tym, że na początku porównujemy ze sobą wartości oddalone o daną ilość miejsc, a następnie odległość tą zmniejszamy po przejściu przez tablicę**.** Jego złożoność obliczeniowa wynosi *O(nlogn).*



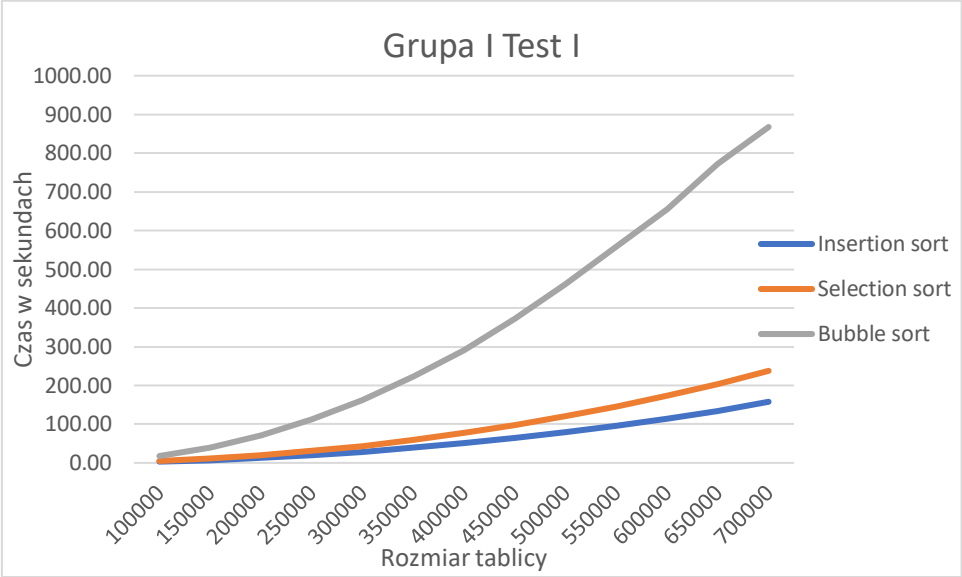
Heap sort:

Podstawą algorytmu jest użycie kolejki priorytetowej zaimplementowanej w postaci binarnego kopca zupełnego. Pierwszym etapem jest stworzenie kopca przy pomocy naszej tablicy, a następnie stopniowe zmniejszanie go poprzez wyciąganie z niego poszczególnych elementów. Jego złożoność obliczeniowa wynosi *O(nlogn).*

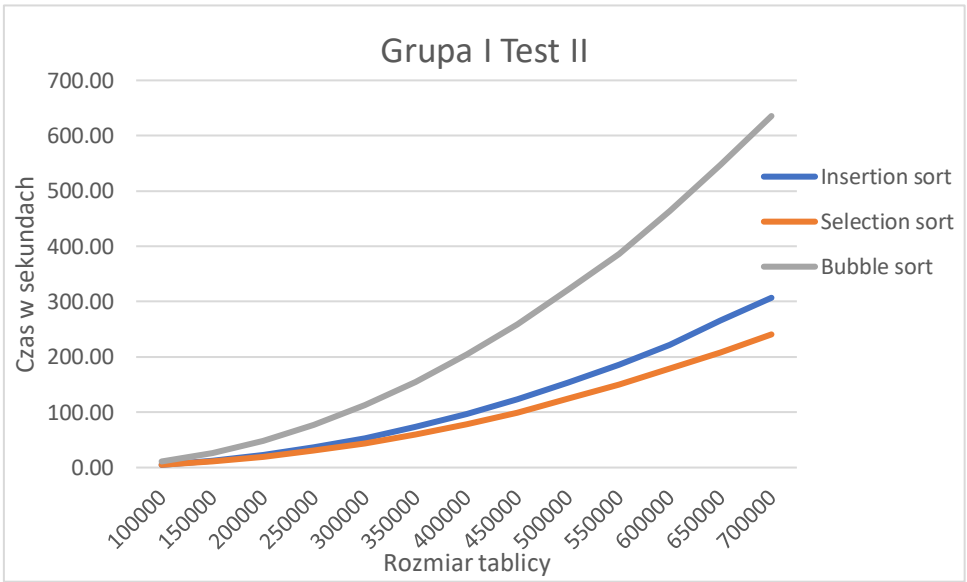




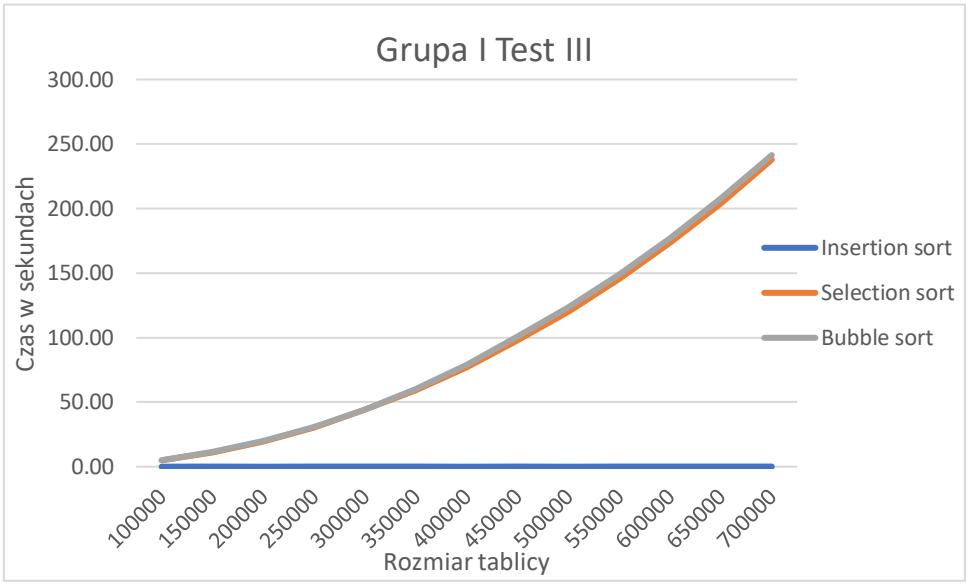
4. Testy czasowe:



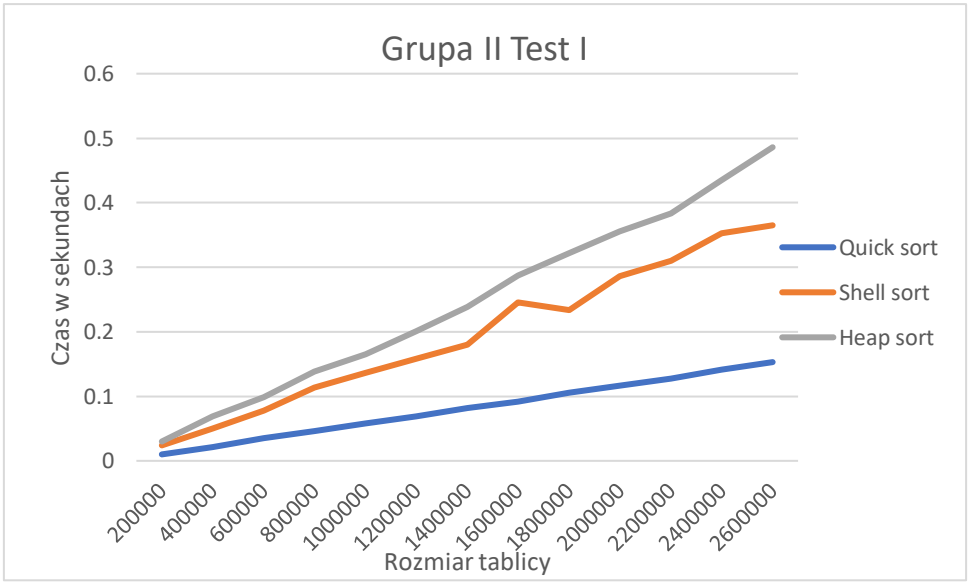
W teście nr I nasze algorytmy były poddane próbie na losowo wygenerowanych liczbach całkowitych z zakresu od -100 do 100. Na podstawie sporządzonego wykresu wyraźnie widać przewagę czasową algorytmu insertion sort, który jednak nie różni się znacząco od algorytmu selection sort. Najgorszym okazał się bubble sort którego czas był kilkukrotnie dłuższy.



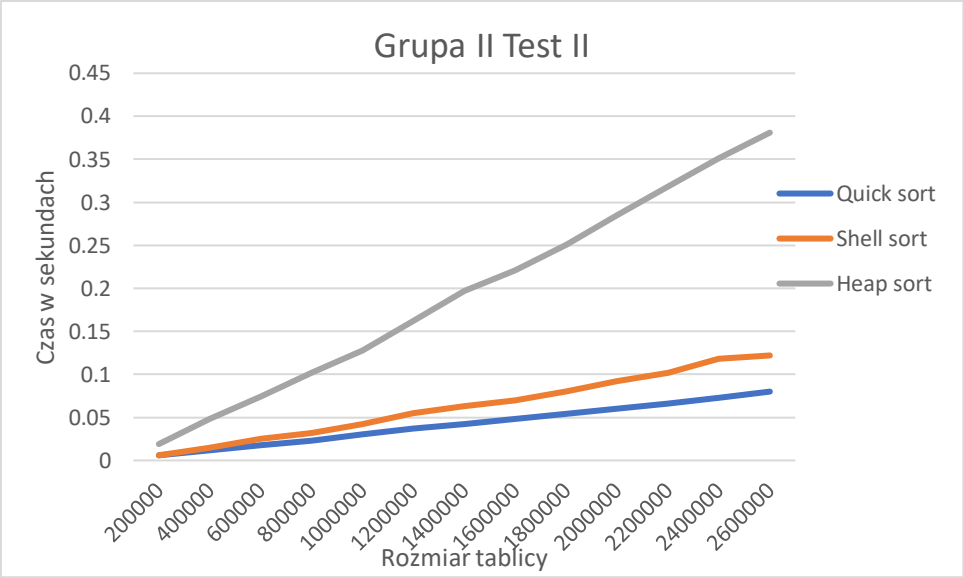
W teście nr II sortowaliśmy dane które zostały już wcześniej uporządkowane w odwrotnej kolejności. Ponownie najgorszym okazał się bubble sort, a najlepszym tym razem selection sort, który jednak po raz kolejny był bardzo zbliżony czasowo do insertion sort.



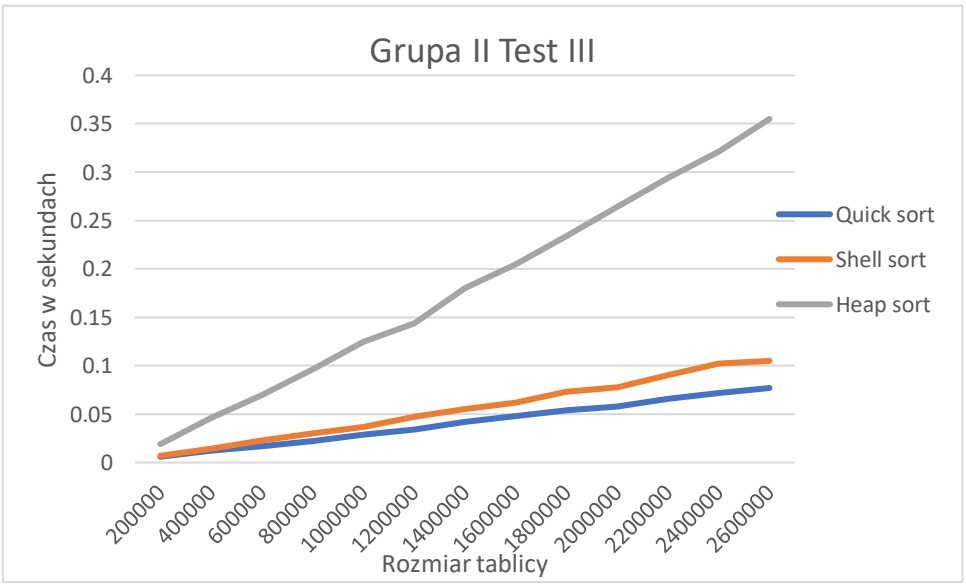
W ostatnim teście dla tej grupy algorytmów nasze dane zostały posortowane w odpowiedniej kolejności. Widzimy że czas działania zarówno bubble sort jak i selection sort jest w zasadzie identyczny natomiast w przypadku insertion sort ten czas jest właściwie niezauważalny ponieważ złożoność obliczeniowa dla tego algorytmu wynosi *O(n)*.



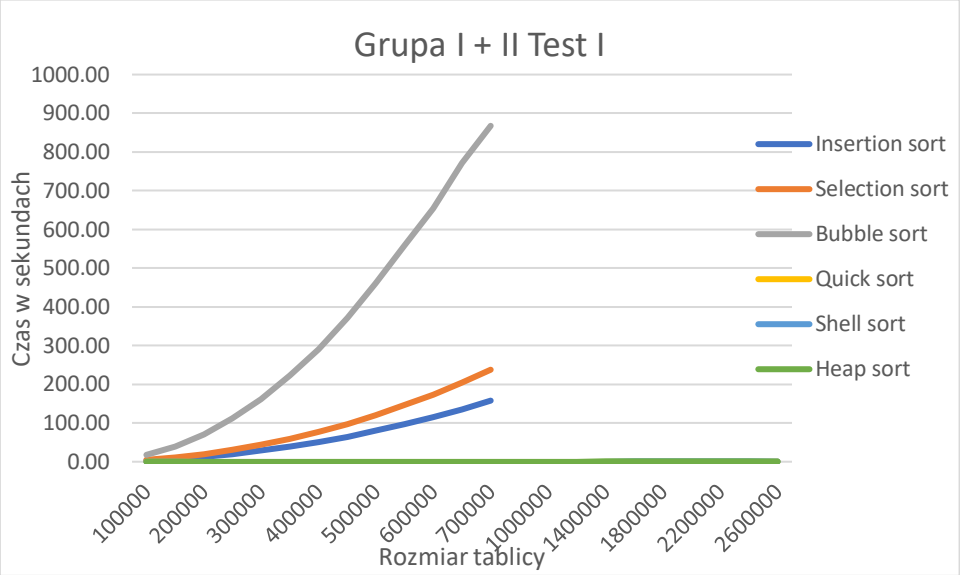
Wraz z kolejnym wykresem przechodzimy do drugiej grupy naszych algorytmów. W teście nr I dla danych losowych zdecydowanie najlepiej wypada quick sort. Nieco gorzej ale też jak najbardziej w porządku działają shell sort oraz heap sort.

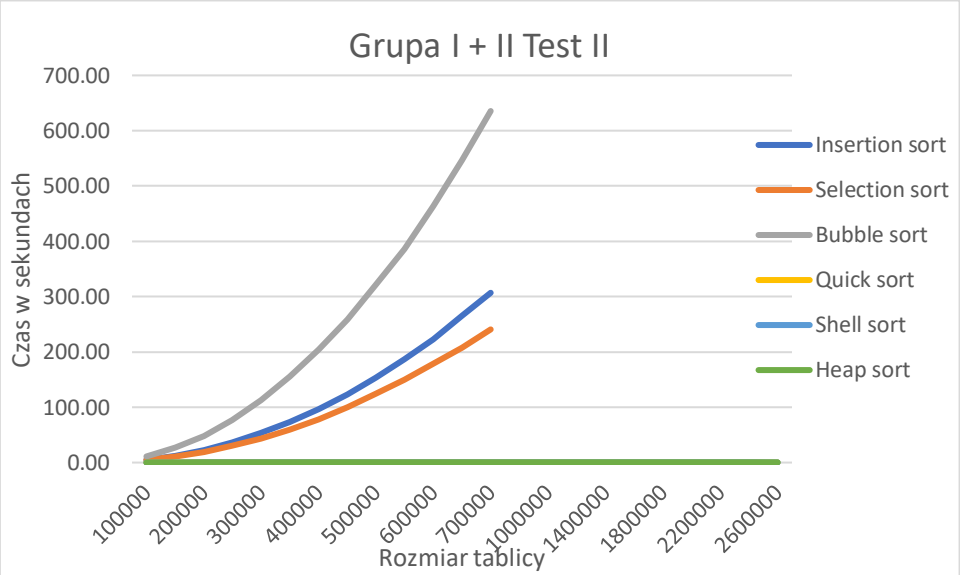


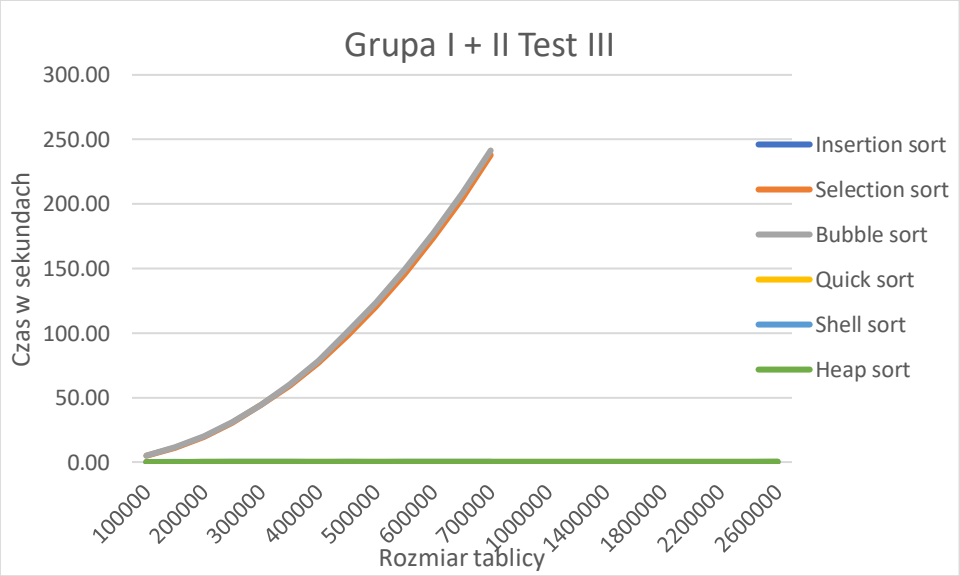
W teście nr II dla danych posortowanych w kolejności odwrotnej widzimy już zdecydowanie dłuższy czas działania dla heap sortu który wydaje się być mało efektywny. Bardzo podobne wyniki odnotowują zarówno quick sort jak i shell sort przy czym quick sort w dalszym ciągu pozostaje najszybszy.



W ostatnim teście, a więc dla danych posortowanych dla tej grupy algorytmów z grupy II widzimy w zasadzie identyczną sytuacje jak przy teście nr II. Najgorzej wypada heap sort, a zdecydowanie lepiej z bardzo podobnymi wynikami quick sort i shell sort.







W powyższych zestawieniach widzimy całą szóstkę algorytmów dla Testów I, II i III. W każdym przypadku dostrzec można zatrważającą różnicą między pierwszą a drugą grupą gdzie tej drugiej w zasadzie nie widać, a więc czas działania tych algorytmów jest bardzo niski.