Sprawozdanie

Politechnika Rzeszowska im. Ignacego Łukasiewicza
Wydział Matematyki i Fizyki Stosowanej
Inżynieria i Analiza Danych

Maciej Żak
Algorytmy i struktury danych
Semestr 1, grupa numer 8

Zadanie: Zaimplementuj sortowanie bąbelkowe oraz sortowanie grzebieniowe.

Do napisania programu użyłem wektorów ponieważ pozwalają one działać na dowolnie długich ciągach liczb oraz funkcji switch która pozwala wybrać rodzaj sortowania które chcemy użyć.

```
int main()
{
    vector <int> s,g;
    vector <int> liczby= wczytaj("liczby.txt");
 char x;
 for(;;)
 {
 cout<<" Ktora metode sortowania chcesz uzyc"<<endl;
 cout<<" 1. Sortowanie grzebieniowe"<<endl;
 cout<<" 2. Sortowanie babelkowe"<<endl;
 cout<<" 3. Koniec Programu"<<endl;
 cout<<" Wybierz: "<<endl; x=getch();

 switch(x)
 {
     case '1':{
        cout<<" grzebieniowe"<<endl<<endl;
       s=grzebieniowe(liczby);
       zapisz_do_pliku(s,liczby);
         break;
     //bublesort
     }
     case '2':{
         cout<<" babelkowe"<<endl<<endl;
         g=babelkowe(liczby);
         zapisz_do_pliku(g,liczby);
         break;

     //sortowanie grzebieniowe
     }
     case '3': exit(0);
     default: cout<<" Wybierz liczbe 1,2 lub 3 !!"<<endl;
 }
 getchar();getchar();
 system("cls");
 }

return 0;
```

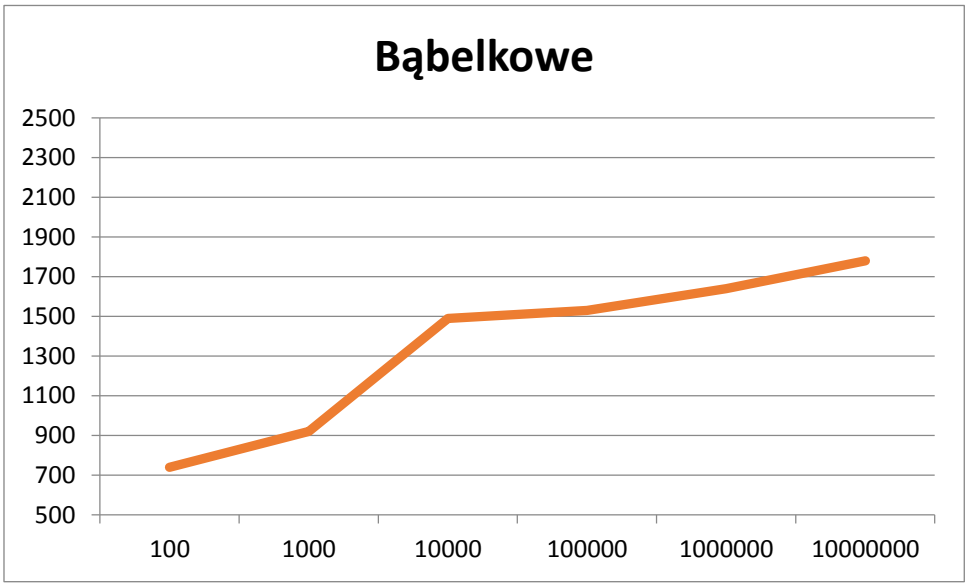
W funkcji głównej mamy wyświetlone menu w którym wybieramy jaki rodzaj sortowania zastosujemy. W przypadku gdy wciśniemy na klawiaturze „1” zostanie użyte sortowanie grzebieniowe, gdy wciśniemy „2” zostanie użyte sortowanie bąbelkowe, natomiast gdy wciśniemy „3” program zakończy działanie.

Sortowanie Bąbelkowe

```
vector<int> babelkowe(vector<int> licz) {
        bool swapped;
        int n;
        n=licz.size();
        do {
            swapped = false;
            for (int i = 0; i < n - 1; i++) {
                if (licz[i] > licz[i + 1]) {
                    swap(licz[i], licz[i + 1]);
                    swapped = true;
                }
            }
            n=n-1;
        } while (swapped);
        return licz;
    }
```

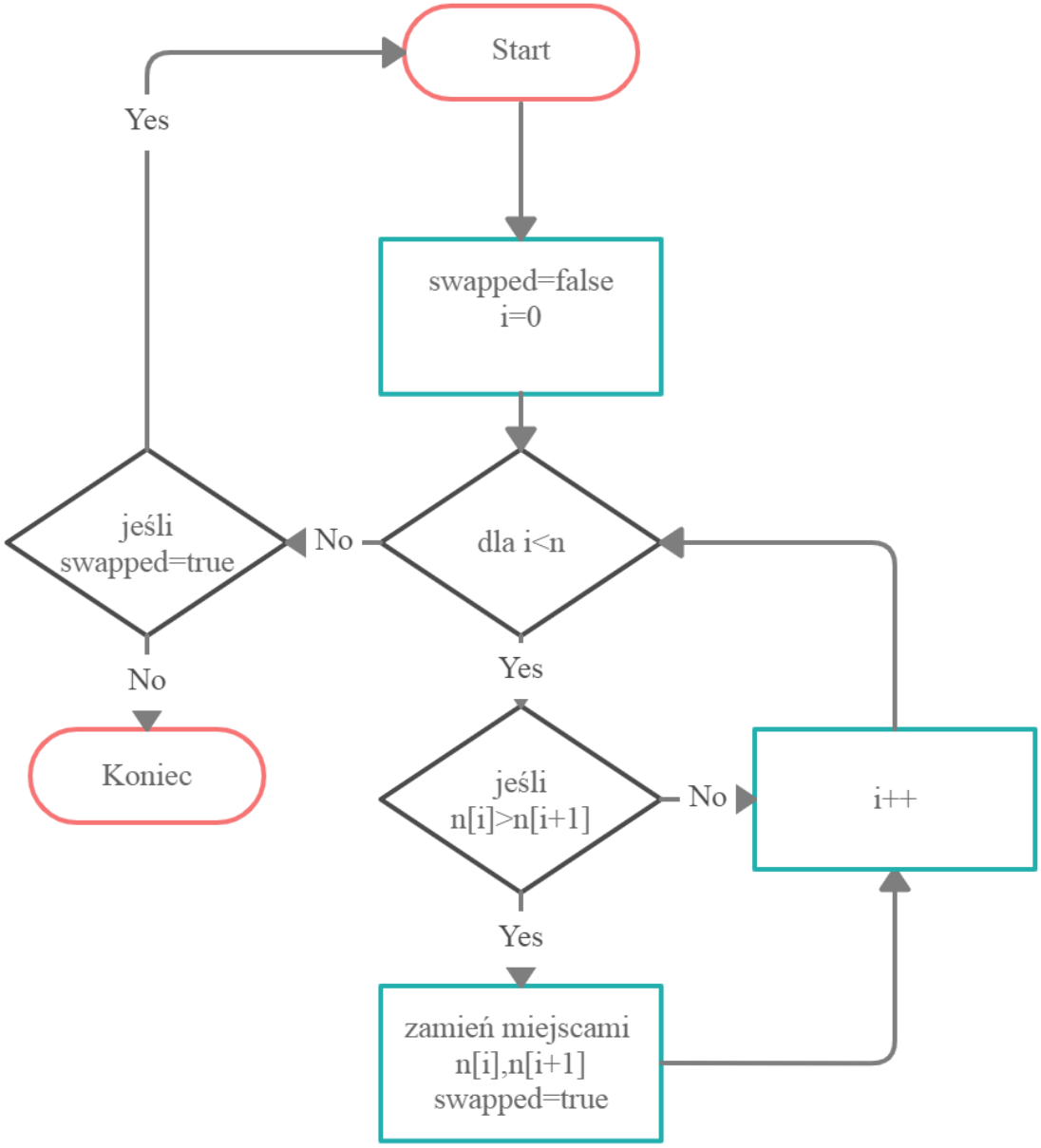
Sortowanie bąbelkowe polega na porównywaniu ze sobą dwóch sąsiadujących wyrazów. Jeżeli element n-1 jest większy od elementu n następuje zamiana ich kolejności. Sortowanie zakańcza się gdy podczas pełnego obiegu pętli żaden element nie zostanie zamieniony. Pesymistyczna złożoność algorytmu to $O(n^2)$, optymistyczna to O(n) natomiast średnia złożoność to O(n^2). Algorytm można zoptymalizować poprzez zmniejszanie n o jeden po każdym przejściu pętli.

Średnia złożoność:



Po lewej stronie wykresu mamy czas podany w nanosekundach a pod spodem dla jakiej ilości liczb został przeprowadzony.

Schemat blokowy dla sortowania bąbelkowego



Pseudokod:

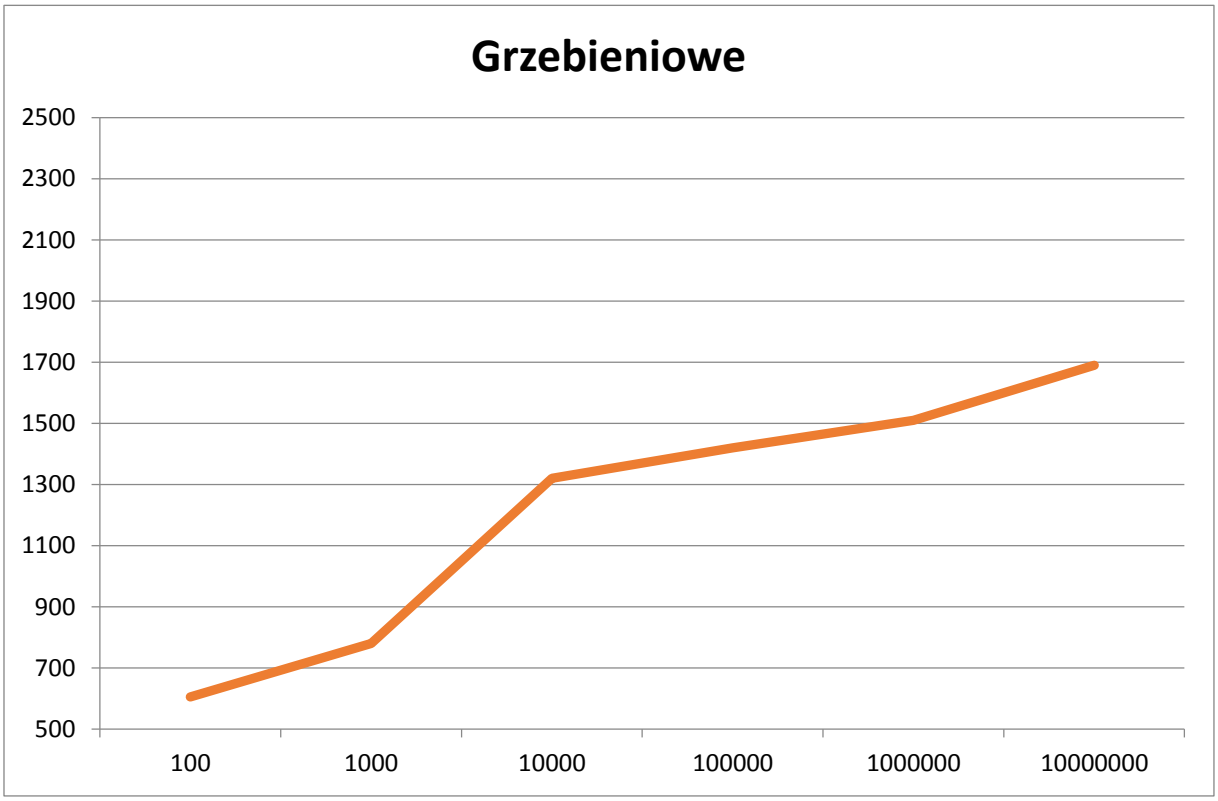
```
Start
n = liczba_elementów(A)
    rób
     dla (i = 0; i < n-1; i++) rób :
        jeśli A[i] > A[i+1] then
         swap(A[i], A[i+1])
       koniec jeśli
     koniec dla
     n = n-1
   kiedy n > 1
 Koniec
```

Sortowanie Grzebieniowe

```
vector<int> grzebieniowe(vector<int> licz)
{

    int  tmp,gap;
     gap= licz.size();
    bool swapped = true;
    while (gap > 1 || swapped){
       gap = gap/1.3;
       if(gap==0)
             gap=1;
       swapped = false;
       for ( int i = 0; i + gap < licz.size(); ++i ) {
          if ( licz[i + gap] < licz[i] ) {
             tmp = licz[i];
             licz[i] = licz[i + gap];
             licz[i + gap] = tmp;
             swapped = true;
           }
       }
    }
     return licz;
}
```

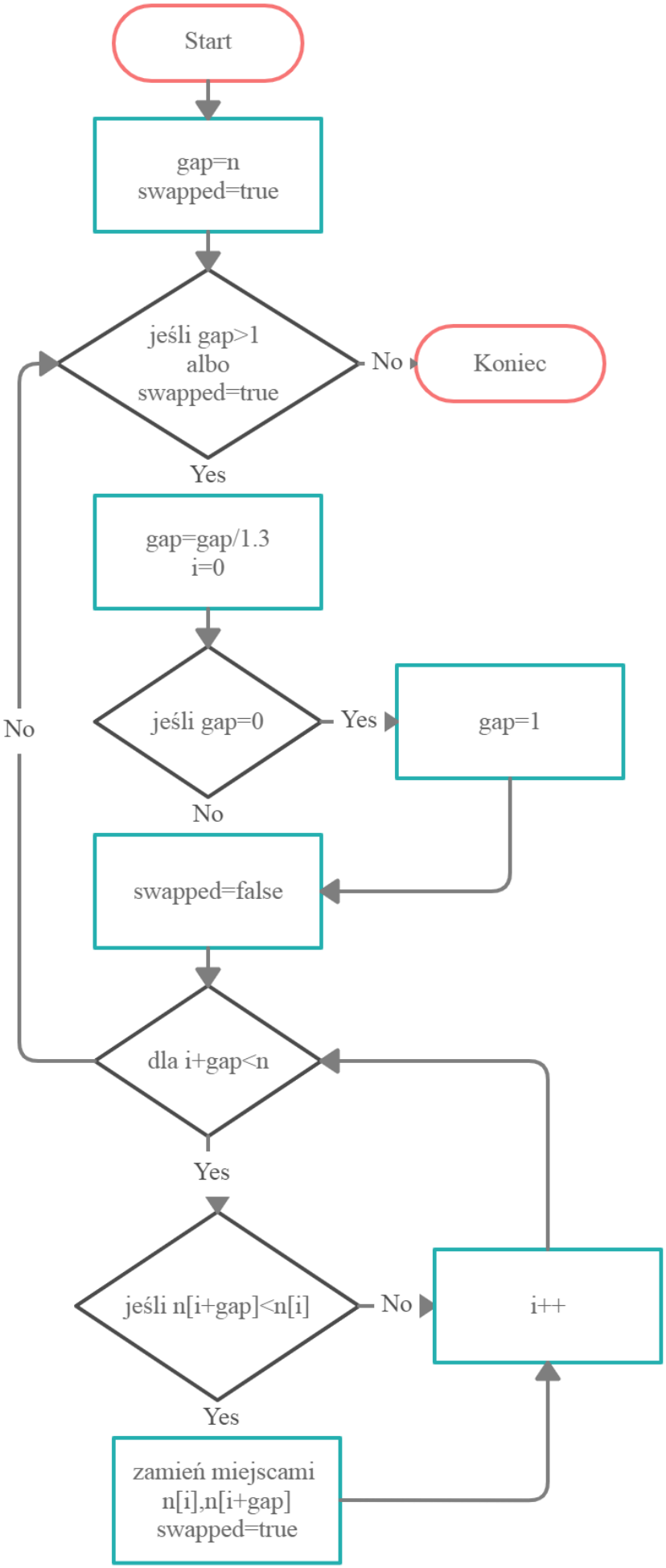
Sortowanie grzebieniowe jest bardzo podobne do sortowania bąbelkowego lecz zamiast porównywać wyrazy sąsiadujące wyznaczamy piwot który jest równy wielkości tablicy podzielonej przez 1.3. Podczas każdego obiegu pętli porównujemy wyraz n i wyraz n + piwot jeżeli ten drugi jest mniejszy następuję zamiana miejscami. Z każdym przejściem pętli rozpiętość jest dzielona przez 1.3 tym samym zmniejszając piwot. Funkcja zakańcza się gdy piwot jest równy jeden i przez cały obieg pętli nie doszło do ani jednej zamiany miejsc. Pesymistyczna złożoność algorytmu to $O(n^2)$, optymistyczna to O(n log n) natomiast średnia złożoność to O($n^2/2^p$) gdzie p jest liczbą obejść pętli.

Średnia złożoność



Po lewej stronie wykresu mamy czas podany w nanosekundach a pod spodem dla jakiej ilości liczb został przeprowadzony.

Schemat blokowy dla sortowania grzebieniowego.



Pseudokod:

```
Start
n = liczba_elementów(A)
gap=n
swapped = true
kiedy (gap>1 albo swapped=true)
    gap = gap /1.3
    jeśli gap=0
        gap=1
    swapped=false
    dla(i=0;  i+gap<n; i++)
        jeśli (n[i+gap]<n[i])
                swap (n[i],n[i+gap])
                swapped=true
         koniec jeśli
    koniec dla
koniec kiedy
Koniec
```

Testy

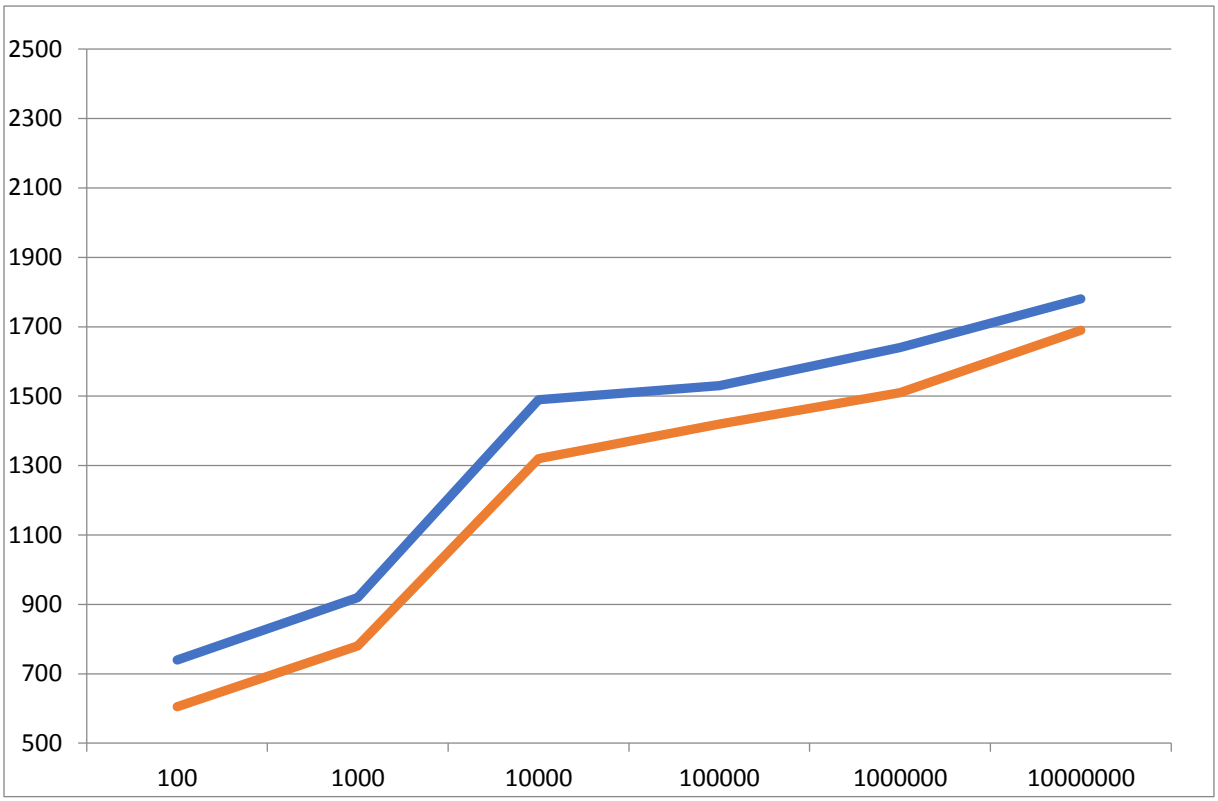
Funkcja której użyłem do testowania została zaimplementowana w oddzielnym programie. Losowała ona najpierw ciąg liczb takiej długości jaką wybraliśmy, następnie zapisywała go do pliku po czym z powrotem go wczytywała, zrobiłem tak by funkcja testująca była jak najbardziej zbliżona w działaniu do funkcji głównej. Po wczytaniu danych następowało rozpoczęcie liczenie czasu następnie sortowanie danych a po posortowaniu zatrzymanie czasu. Wielkość ciągów dla których były robione testy zaczynała się od 100 a kończyła na 10 mln. Testowanie dla mniejszych ciągów odbywało się 100 razy, dla większych 10 razy, następnie program dodawał czas ze wszystkich prób i liczył średni czas działania. Rozpiętość liczb losowanych wynosiła od 1 do 100 tys.

```
void testy(int a){
   vector<int> s,c;
double sredni_czas=0;
int i,liczba_testow = 10,wynik;
for(i=1;i<=liczba_testow;i++){

   s=generujlosowe(a);
   zapisz_do_pliku(s);
   c=wczytaj("liczby.txt");

   auto start =chrono::steady_clock::now();
   grzebieniowe(c);
   //babelkowe(c);
   auto stop = chrono::steady_clock::now();

   wynik=chrono::duration_cast<chrono::nanoseconds>(stop-start).count();

   sredni_czas=sredni_czas+wynik;
   }
   sredni_czas=sredni_czas/liczba_testow;
   cout<<endl<<" sredni czas dzialania funkcji przy 10 probach dla ciagu "
   <<a<<" liczb "<<endl<<" to: "<<sredni_czas<<" nanosekund"<<endl;
}
```

Z testów wynikło, że sortowanie grzebieniowe średnio ma lepsze wyniki niż buble sort , ale w skrajnych przypadkach różnice praktycznie nie istnieją.

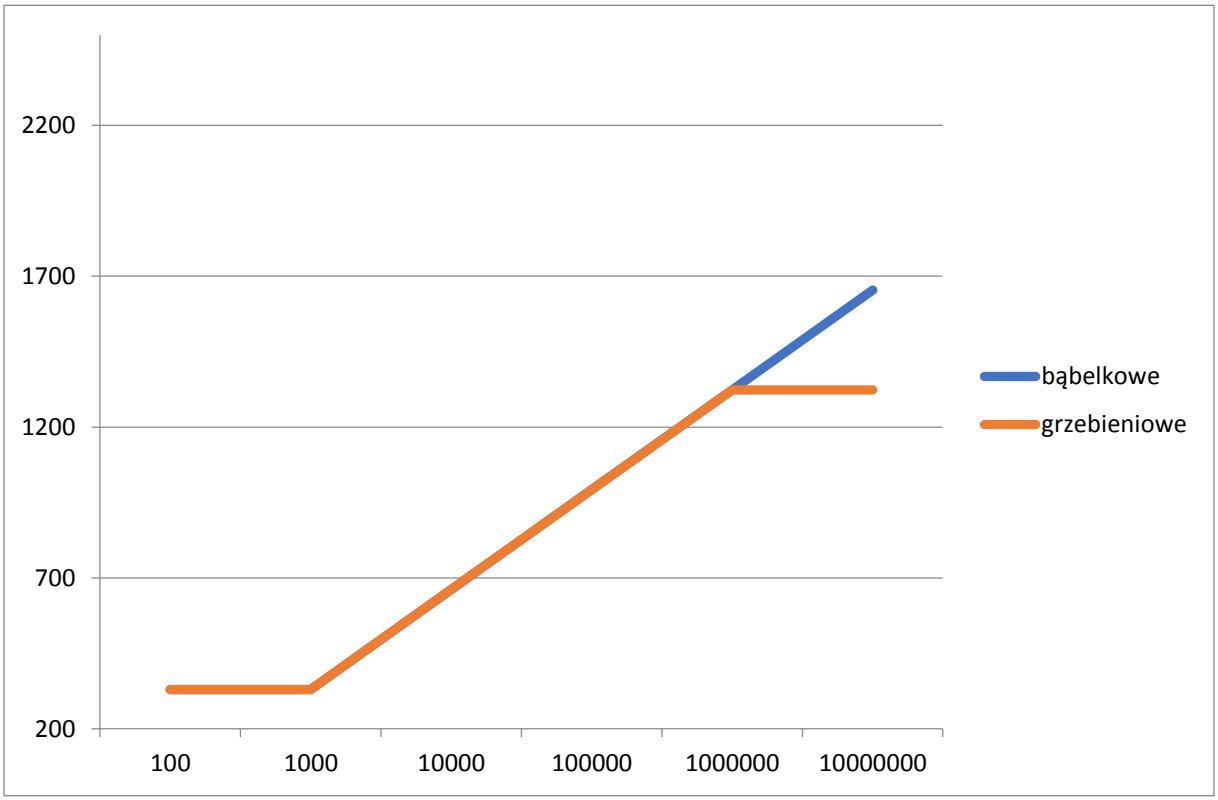
Porównanie dwóch wykresów. Na niebiesko sortowanie bąbelkowe, na pomarańczowo sortowanie grzebieniowe(czas podawany w nanosekundach).

Złożoność średnia:



Ilość elementów ciągu	Czas dla bąbelkowego	Czas dla grzebieniowego
100	740	605
1000	920	780
10000	1490	1320
100000	1530	1420
1000000	1640	1510
10000000	1780	1690

Złożoność optymistyczna:



Złożoność pesymistyczna: