Porównanie algorytmów sortujących

Zadanie projektowe numer 2

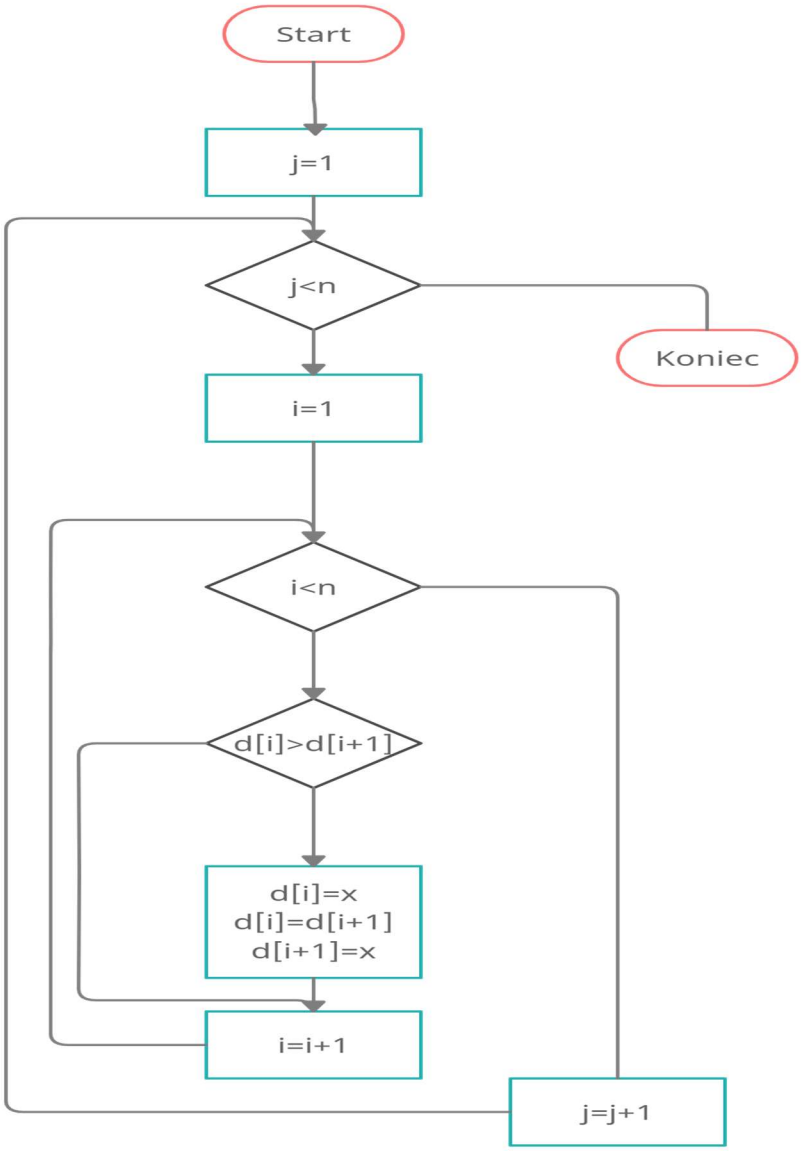
Kacper Chmura, 169767
Inżynieria i analiza danych, grupa P1

1 PRZEDSTAWIENIE OMAWAINYCH ALGORYTMÓW SORTOWANIA

1.1 SORTOWANIE BĄBELKOWE OPIS

Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących. Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

1.1.1 Schemat blokowy sortowania bąbelkowego



1.1.2 Pseudokod sortowania bąbelkowego

Dla j=1 wykonuj dopóki j<n:

    Dla i=1 wykonuj dopóki i<n:

        Jeżeli d[i]<d[i+1]

        To zamień miejscami d[i] oraz d[i+1]

        i=i+1

    j=j+1

1.1.3 Pesymistyczne i Optymistyczne ustawienie dla sortowania bąbelkowego

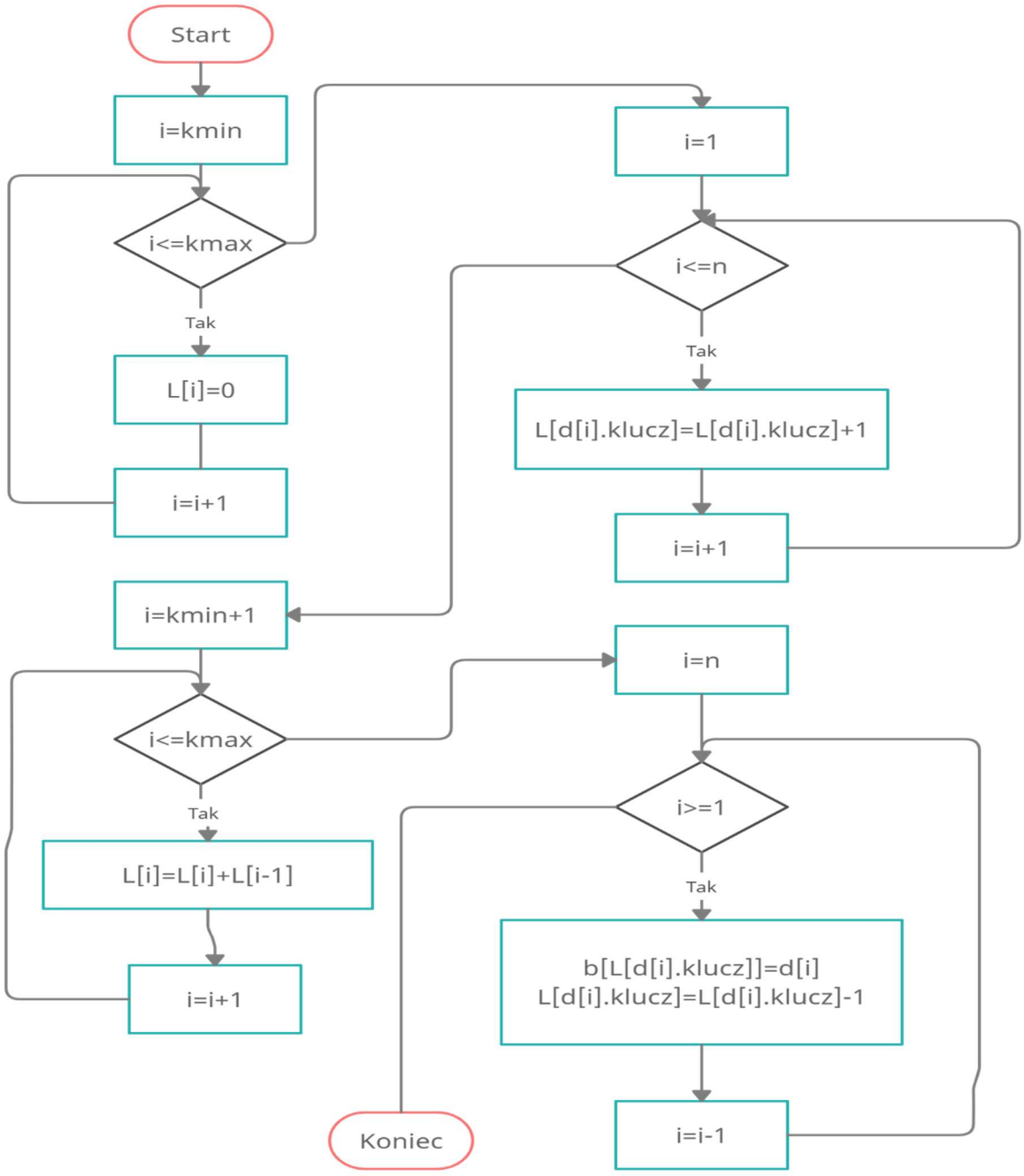
Przypadek najbardziej niekorzystyny to taki, w którym przykładowo podczas sortowania elementów od najmniejszego do najwiekszego, element najmniejszy znajduje sie na samym końcu. Ponieważ wtedy algorytm musi wykonać pełen obieg pętli. Różnorodność elementów nie ma tutaj znazenia.

Przypadek najkorzystniejszy to odwrotność powyższego czyli element najmniejszy jest juz na właściwym miejscu.

1.2 SORTOWANIE PRZEZ ZLICZANIE OPIS

Sortowanie przez zliczanie jest to metoda sortowania danych, która polega na sprawdzeniu ile wystąpień kluczy mniejszych od danego występuje w sortowanej tablicy. Algorytm zakłada, że klucze elementów należą do skończonego zbioru (np. są to liczby całkowite z przedziału 0-100), co ogranicza możliwości jego zastosowania. Co ciekawe algorytm ten nie porównuje ze sobą żadnego elementu zbioru, a zapisuje „jego dane” do odpowienich tablic. Następnie dane z tych tablic są wykorzystywane posortowac dane w odpowiedniej kolejnosci.

1.2.1 Schemat blokowy sortowania przez zliczanie



1.2.2 Pseudokod sortowania przez zliczanie

Dla i=kmin wykonuj dopoki i<=max:

L[i]=0

i=i+1

```
dla i=1 wykonuj dopóki i<=n:
        L[d[i].klucz]=L[d[i].klucz+1]
        i=i+1
dla i=kmin+1 wykonuj dopóki i<=kmax:
        L[i]=L[i]+L[i+1]
        i=i+1
dla i=n wykonuj dopók i>=1:
        b[L[d[i].klucz]]=d[i]
        L[d[i].klucz]=L[d[i].klucz]-1
        i=i+1
```

1.2.3 Pesymistyczne ustawienie dla sortowania przez zliczanie

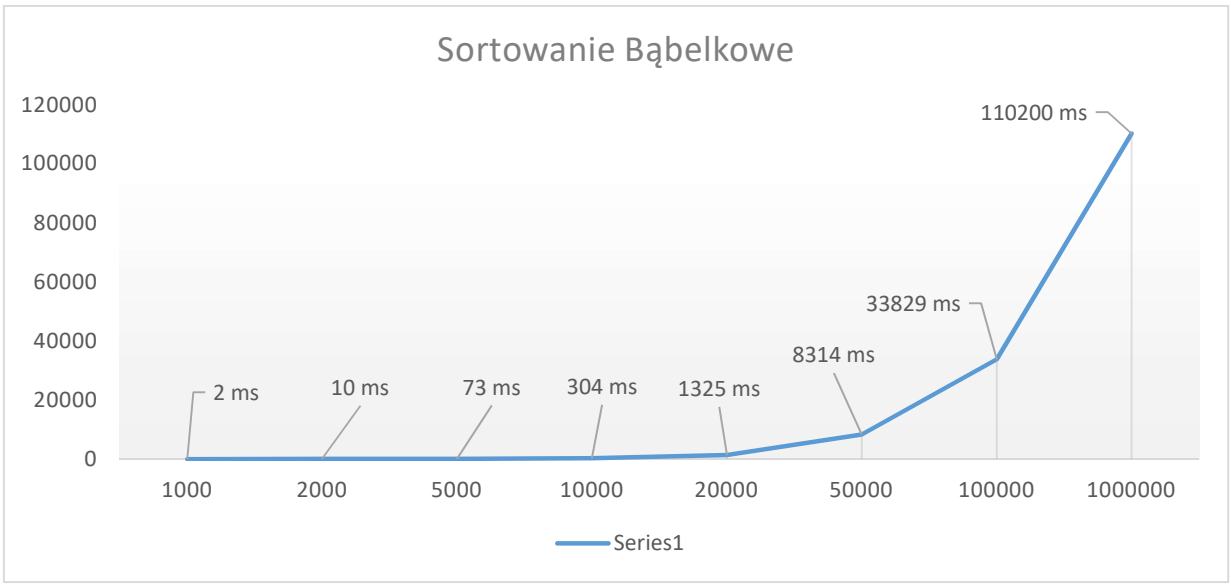
Złożoność obliczeniowa algorytmu to O(n+k) gdzie n to ilość elementów a k to zakres sortowanych wartości. Więc najbardziej niekorzystna dla sortowania przez zliczanie jest sytuacja gdy nietylko elementów jest wiele ale również gdy są różnorodne.

2 ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMÓW

2.1 ZŁOŻONOŚĆ OBLICZENIOWA SORTOWANIA BĄBELKOWEGO

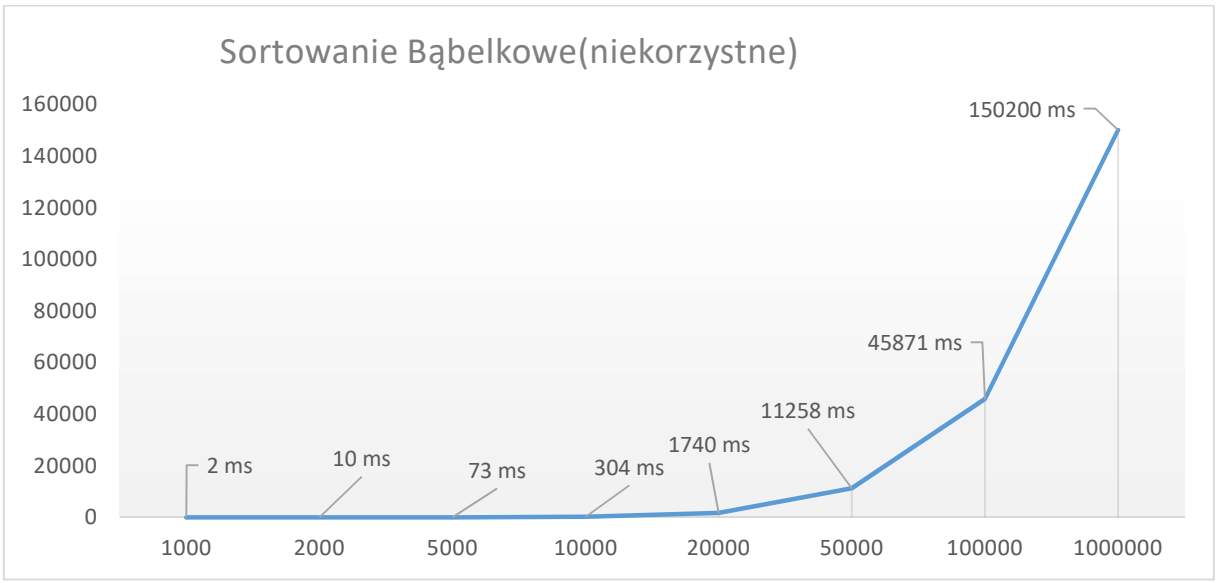
Algorytm sortowania bąbelkowego jest bardzo prymitywnym sposobem sortowania danych(chodź lepszym od sortowania „głupiego”), jest on jedynie użyteczny dla posortowania małych zbiorów ponieważ jego złożoność wynosi $O(n^2)$.

2.1.1 Wykreres czasu od ilości elementów(nie spreparowane)

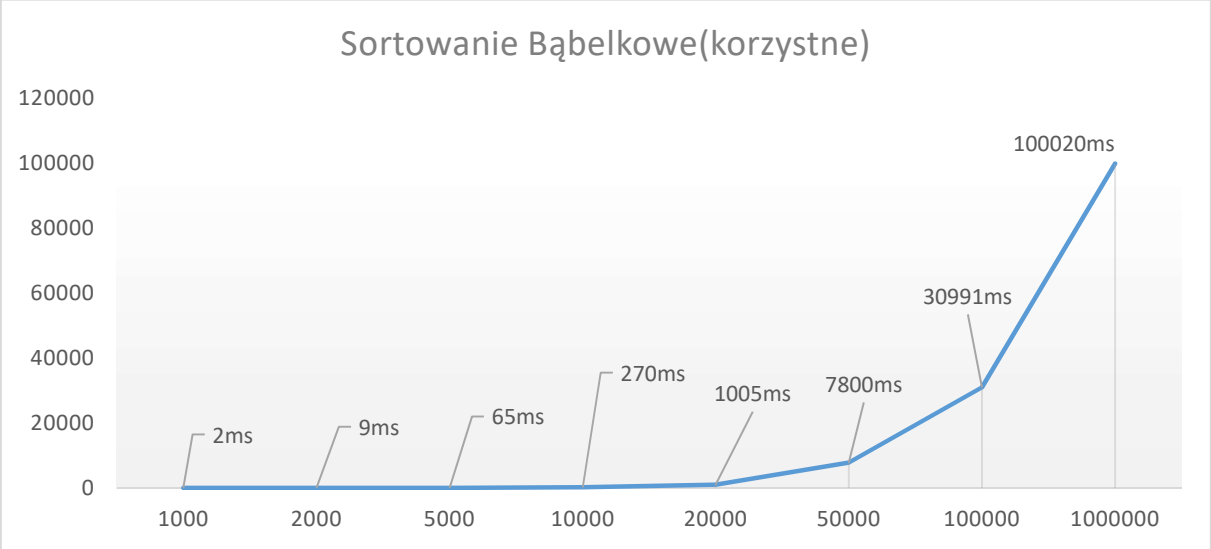


Jak widzimy na powyższym wykresie algorytm ten słabo radzi sobie ze sporą ilością danych już do posortowania miliona liczb, program z zaimplementowanym algorytmem potrzebuje prawie 2 minuty.

2.1.2 Wykreres czasu od ilości elementów(niekorzystne)



2.1.3 Wykres czasu od ilosci elementów(korzystne)



Spreparowanie danych polegało na tym, że dane w tablicy przed posortowaniem zostałty ułożone tak że wartości małe, które maja po posortowaniu znaleźć sie na początku tablicy zostały wrzucone na koniec, a duże na początek.

Sposób spreparowania danych:

Niekorzystne:

```
for(i = 0; i < N/3; i++) d[i] = (rand() % 3)+7; //losowanie liczb 7-9
for(i = N/3; i < N*2/3; i++) d[i] = (rand() % 3)+4;//losowanie liczb 4-6
for(i = N*2/3; i < N; i++) d[i] = (rand() % 3)+1; // losowanie liczb 1-3
//for(i = 0; i < N; i++) cout << setw(4) << d[i];
```

Korzystne:

```
for(i = 0; i < N/3; i++) d[i] = (rand() % 3)+1; //losowanie liczb od 1 do 3
for(i = N/3; i < N*2/3; i++) d[i] = (rand() % 3)+4; //losowanie liczb od 4 do 6
for(i = N*2/3; i < N; i++) d[i] = (rand() % 3)+7; //losowanie liczb od 7 do 9
//for(i = 0; i < N; i++) cout << setw(4) << d[i];
```

2.2 Złożoność obliczeniowa Sortowania przez Zliczanie

Sortowanie przez zliczanie to znacznie bardziej skąplikowany algorytm od poprzednika, lecz jego złożoność czasowa wynosząca O(n+k) jest znacznie korzystniejsza zwłaszcza dla wiekszych ilości danych.

2.2.1 Wykres czasu od ilości elementów dla zakresu od 1 do 100(korzystne)

2.2.2 Wykres czasu od ilosci elementów dla zakresu od 1 do 100000(niekorzystne)

W tym przypadku spreparowanie danych polegało jedynie na zwiększeniu zakresu sortowanych liczb.

3 ZUŻYCIE CPU PODCZAS DZIAŁANIU ALGORYTMÓW

3.1 PODSUMOWANIE

Ilość elementów	Bąbelkowe	Przez zliczanie
40000	11.8%	4.5%
85000	13.7%	6.5%
100000	15.2%	7.1%
250000	22.1%	-
1000000	26.1%	-

Porównując dane z powyższej tabeli można zauważyć, że korzystniejsza czasowo złozoność obliczeniowa więże sie z wiekszym zużyciem procesora. Tak jak w powyższym przypadku, gdzie porównywane są sortowania „bąbelkowe” oraz „przez zliczanie” o złożoności czasowej odpowiednio O(n^2) oraz O(n+k).

4 SPIS TREŚCI