

Rzeszów, 14.12.2021

Sprawozdanie
z projektu

Politechnika Rzeszowska
Algorytmy i struktury danych

Krzysztof Motas
Inżynieria i analiza danych, 1 rok, grupa nr 5

Link do repozytorium projektu na platformie GitHub:
https://github.com/kmotas/aisd_projekt2

Spis treści

1. Opis problemu	3
2. Podstawy teoretyczne	3
3. Schematy blokowe	4
4. Pseudokody	7
5. Wykresy złożoności czasowej.....	9
6. Omówienie złożoności obliczeniowej.....	12
7. Testy działania algorytmów.....	12

1. Opis problemu

Zaimplementuj sortowanie przez zliczanie oraz sortowanie przez kopcowanie.

2. Podstawy teoretyczne

a) Sortowanie przez zliczanie

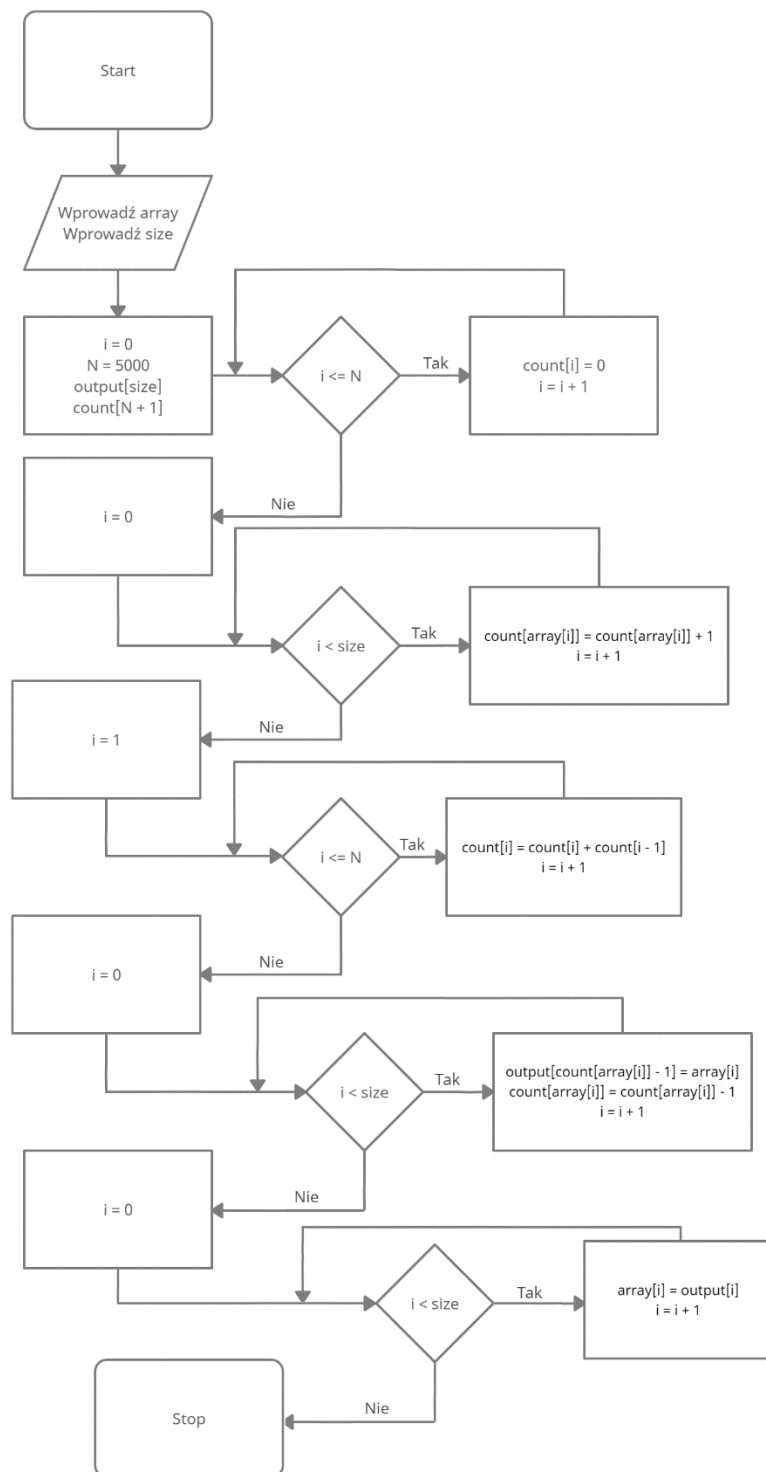
Algorytm polega na zadeklarowaniu dodatkowej tablicy do której będziemy zliczać wystąpienia poszczególnych wartości. Następny krok polega na analizie wszystkich elementów listy zliczającej. i -tą liczbę na liście wypisujemy tyle razy jaką wartość ma lista pod indeksem i . W ten sposób uzyskujemy posortowaną listę. Jeśli listę zliczającą będziemy przeglądać od lewej do prawej to uzyskamy listę posortowaną rosnącą.

b) Sortowanie przez kopcowanie

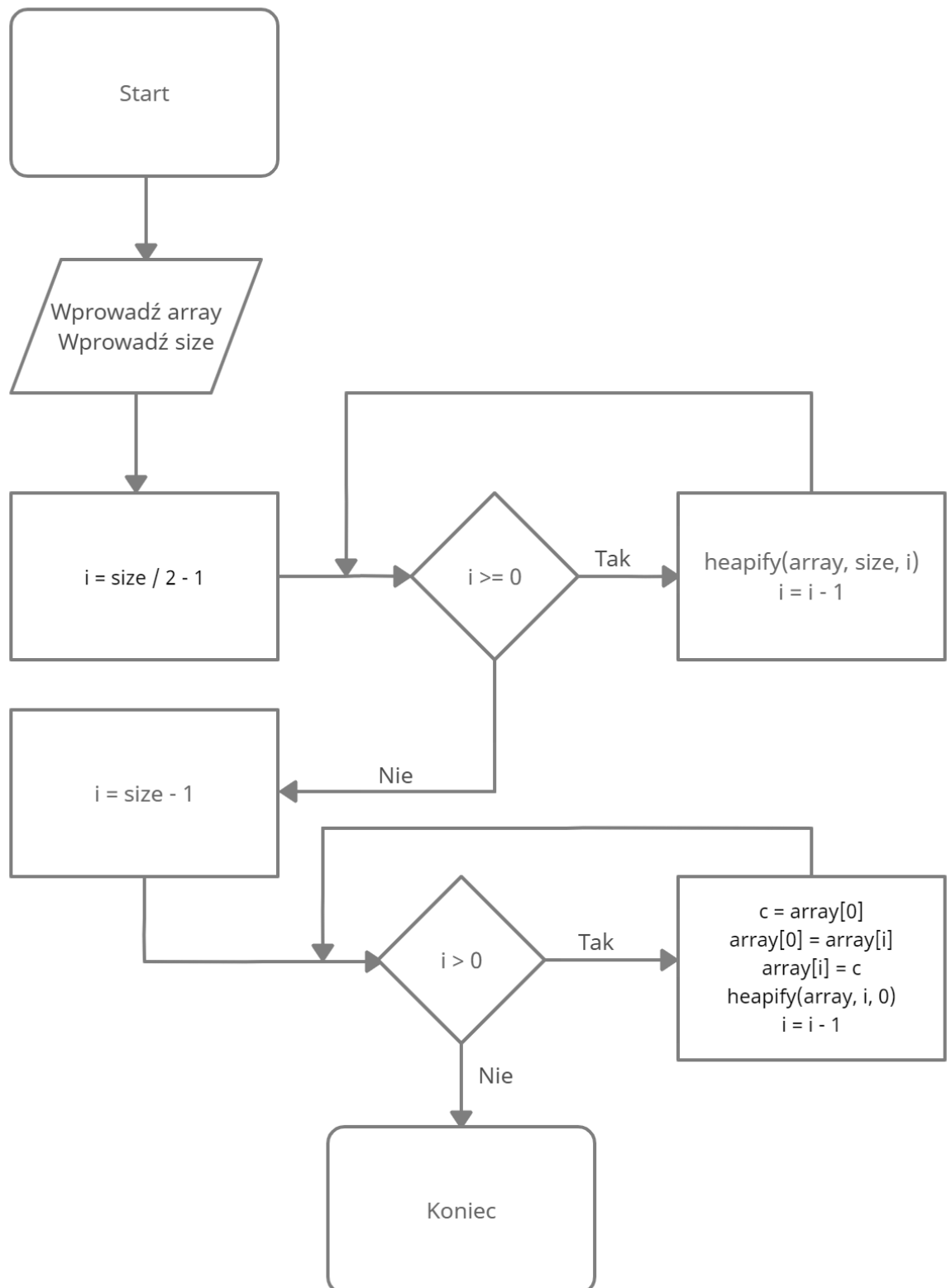
Algorytm sortowania przez kopcowanie składa się z dwóch faz. W pierwszej sortowane elementy reorganizowane są w celu utworzenia kopca. W drugiej zaś dokonywane jest właściwe sortowanie. Podstawową zaletą algorytmu jest to, że do stworzenia kopca wykorzystać można tę samą tablicę, w której początkowo znajdują się nieposortowane elementy. Dzięki temu uzyskuje się stałą złożoność pamięciową. Początkowo do kopca należy tylko pierwszy element w tablicy. Następnie kopiec rozszerzany jest o drugą, trzecią i kolejne pozycje tablicy, przy czym przy każdym rozszerzeniu, nowy element jest przemieszczany w górę kopca, tak aby spełnione były relacje pomiędzy węzłami. Po utworzeniu kopca następuje właściwe sortowanie. Polega ono na usunięciu wierzchołka kopca, zawierającego element maksymalny (minimalny), a następnie wstawieniu w jego miejsce elementu z końca kopca i odtworzenie porządku kopcowego. W zwolnione w ten sposób miejsce, zaraz za końcem zmniejszonego kopca wstawia się usunięty element maksymalny. Operacje te powtarza się aż do wyczerpania elementów w kopcu.

3. Schematy blokowe

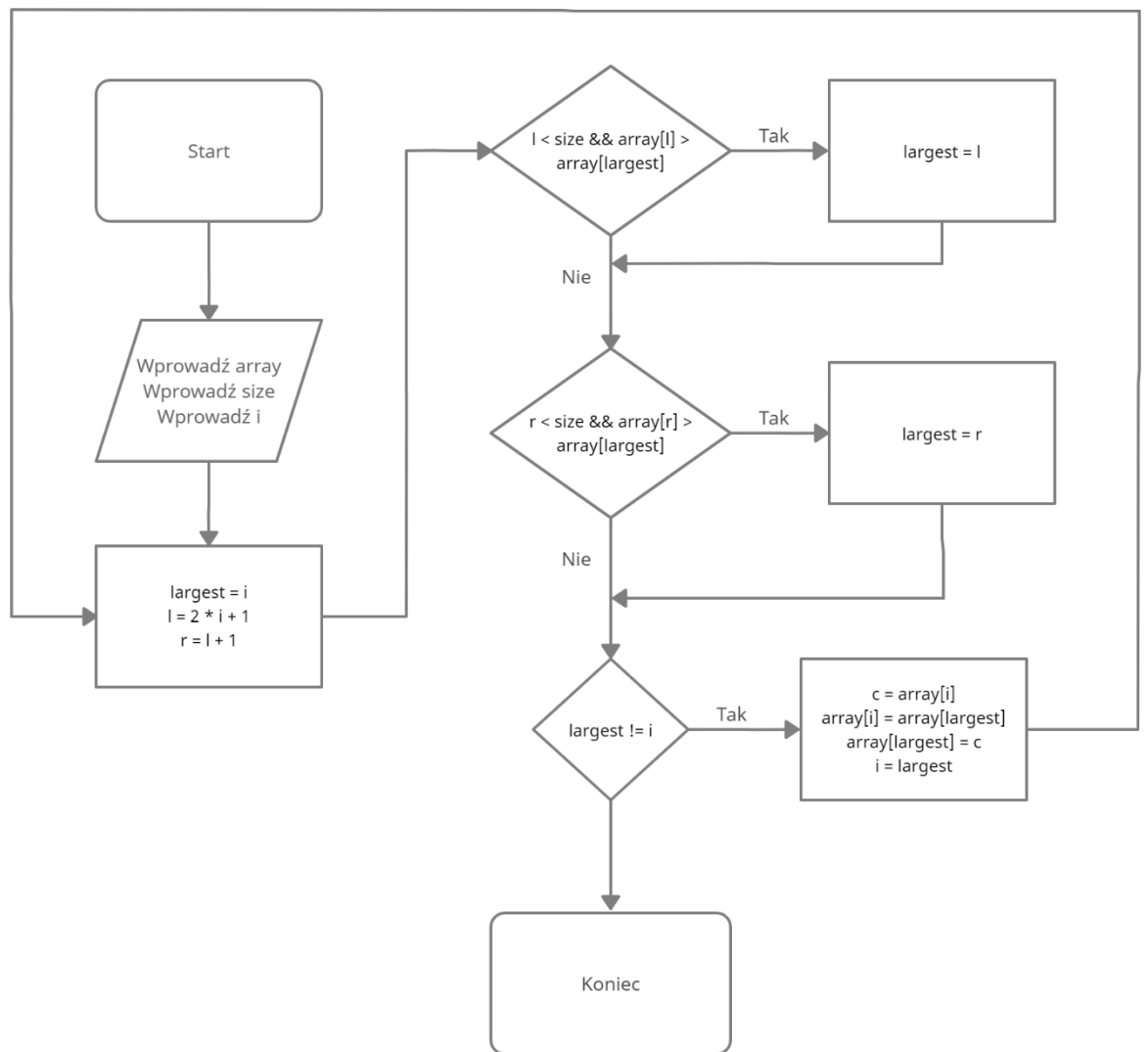
a) Sortowanie przez zliczanie



- b) Sortowanie przez kopcowanie
- Główna funkcja sortująca



- Funkcja „kopcująca” - **heapify**



4. Pseudokody

a) Sortowanie przez zliczanie

wczytaj(array)

wczytaj(size)

$i \leftarrow 0$

$N \leftarrow 5000$

output[size]

count[N + 1]

dopóki $i \leq N$ wykonuj

 count[i] \leftarrow 0

$i \leftarrow i + 1$

$i = 0$

dopóki $i < \text{size}$ wykonuj

 count[array[i]] \leftarrow count[array[i]] + 1

$i \leftarrow i + 1$

$i = 1$

dopóki $i \leq N$ wykonuj

 count[i] \leftarrow count[i] + count[i - 1]

$i \leftarrow i + 1$

$i = 0$

dopóki $i < \text{size}$ wykonuj

 output[count[array[i]] - 1] \leftarrow array[i]

 count[array[i]] \leftarrow count[array[i]] - 1

$i \leftarrow i + 1$

$i = 0$

dopóki $i < \text{size}$ wykonuj

 array[i] \leftarrow output[i]

$i \leftarrow i + 1$

b) Sortowanie przez kopcowanie

- Główna funkcja sortująca
wczytaj(array)
wczytaj(size)

```
i <- size / 2 - 1
```

```
dopóki i >= 0 wykonuj  
  heapify(array, size, i)  
  i <- i - 1
```

```
i <- size - 1
```

```
dopóki i > 0 wykonuj  
  c <- array[0]  
  array[0] <- array[i]  
  array[i] <- c  
  heapify(array, i, 0)  
  i <- i - 1
```

- Funkcja „kopcująca” - **heapify**
wczytaj(array)
wczytaj(size)
wczytaj(i)

```
largest <- i  
l <- 2 * i + 1  
r <- l + 1
```

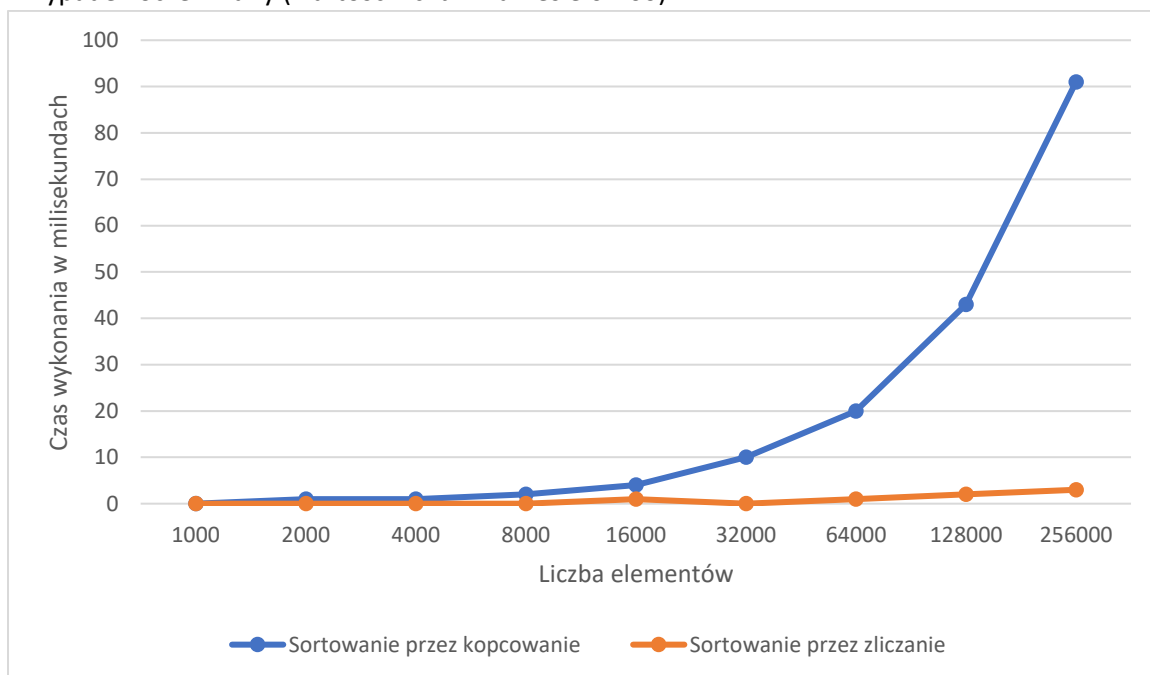
```
jeżeli l < size i array[l] > array[largest] wykonaj  
  largest <- l
```

```
jeżeli r < size i array[r] > array[largest] wykonaj  
  largest <- r
```

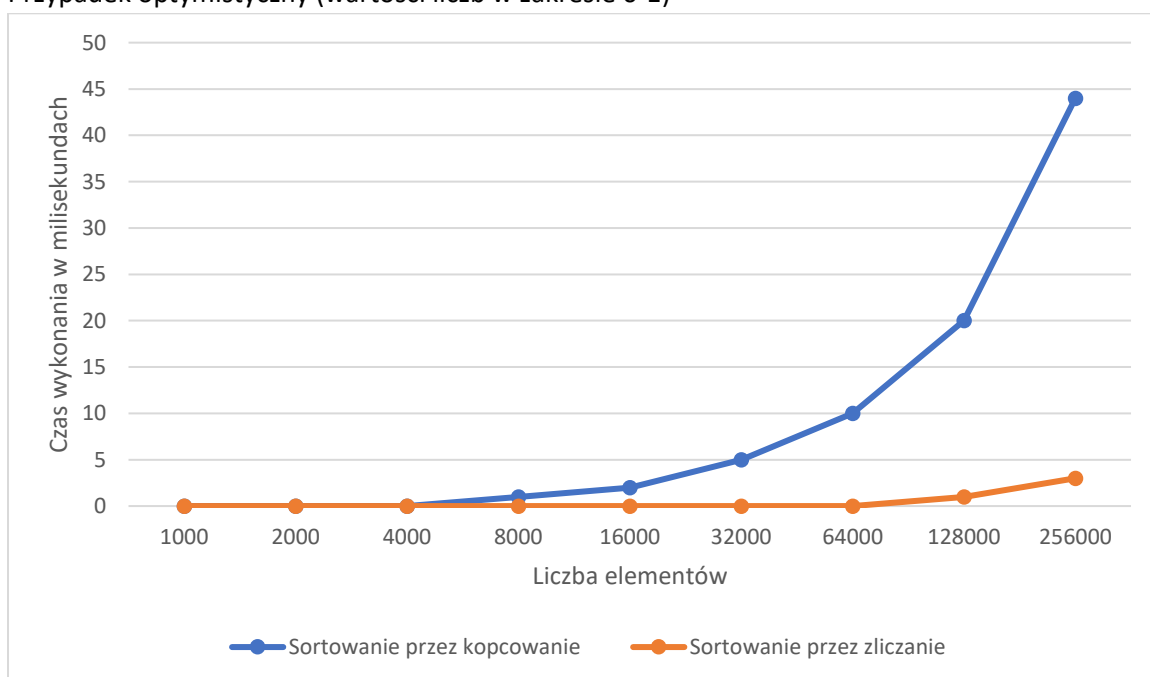
```
jeżeli largest != i wykonaj  
  c <- array[i]  
  array[i] <- array[largest]  
  array[largest] <- c  
  heapify(array, size, largest)
```


5. Wykresy złożoności czasowej

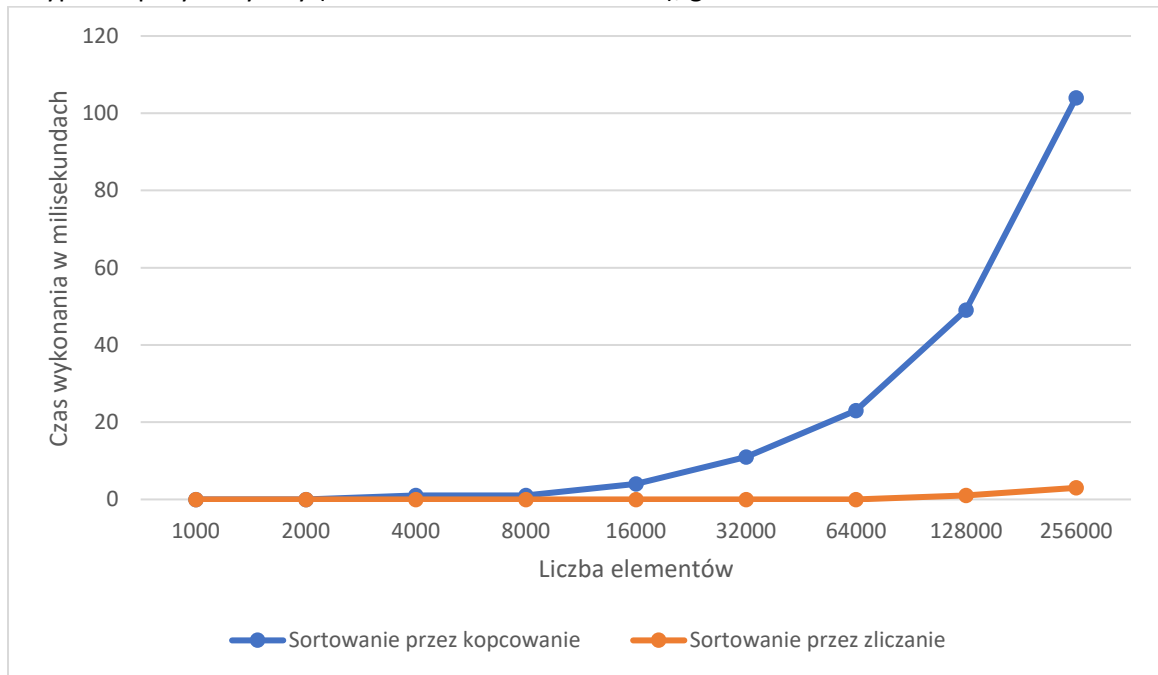
a) Przypadek oczekiwany (wartości liczb w zakresie 0-100)



b) Przypadek optymistyczny (wartości liczb w zakresie 0-1)



c) Przypadek pesymistyczny (wartości liczb w zakresie 0-N), gdzie $N = 5000$



Wykresy zostały skonstruowane na podstawie danych wygenerowanych z funkcji:

```
void timeTests()
{
    fstream file;
    file.open("results/tests.txt", ios::out);

    for (int i = 0, c, size, max; i < 3; i++)
    {
        // i: 0 - przypadek oczekiwany, 1 - przypadek optymistyczny, 2 - przypadek pesymistyczny

        file << "Przypadek " << (i == 2 ? "pesymistyczny" : i ? "optymistyczny" : "oczekiwany") << endl;
        max = (i == 2 ? N : i ? 1 : 100);

        for (c = 0, size = 1000; c < 9; c++)
        {
            int heapArray[size], countArray[size];

            generateRandomData(heapArray, size, max);
            copy(heapArray, heapArray + size, countArray);

            // Dla sortowania przez kopcowanie
            auto start = chrono::steady_clock::now();

            heapSort(heapArray, size);
            file << "heapSort: " << size << " liczb, " <<
            chrono::duration_cast<chrono::milliseconds>(chrono::steady_clock::now() - start).count() << endl;

            // Dla sortowania przez zliczanie
            start = chrono::steady_clock::now();

            countSort(countArray, size);
            file << "countSort: " << size << " liczb, " <<
            chrono::duration_cast<chrono::milliseconds>(chrono::steady_clock::now() - start).count() << endl <<
            endl;

            size *= 2;
        }
        file << endl;
    }

    file.close();
    cout << "Wykonano testy złożoności obliczeniowej algorytmów sortowania. Wyniki znajdziesz w
    results/tests.txt." << endl;
}
```

6. Omówienie złożoności obliczeniowej

Złożoność czasowa algorytmu sortowania przez kopcowanie wynosi $O(n \log n)$. Jest on w praktyce z reguły wolniejszy od sortowania przez zliczanie. Główną zaletą sortowania przez zliczanie jest liniowa złożoność obliczeniowa algorytmu ($O(n+k)$, gdzie n – liczebność zbioru, k – rozpiętość danych). Największymi ograniczeniami algorytmu są konieczność uprzedniej znajomości zakresu danych i złożoność pamięciowa (wymaga dodatkowo $O(k)$ lub $O(n+k)$ pamięci).

7. Testy działania algorytmów

a) Liczba elementów w tablicy: 10

- Dane wejściowe: 27 17 67 8 4 10 42 83 30 87
- Dane wyjściowe (sortowanie przez zliczanie): 4 8 10 17 27 30 42 67 83 87
- Dane wyjściowe (sortowanie przez kopcowanie): 4 8 10 17 27 30 42 67 83 87

b) Liczba elementów w tablicy: 15

- Dane wejściowe: 1 1 0 1 0 0 0 0 1 1 1 1 1 0 0
- Dane wyjściowe (sortowanie przez zliczanie): 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
- Dane wyjściowe (sortowanie przez kopcowanie): 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

d) Liczba elementów w tablicy: 30

- Dane wejściowe: 4 9 26 39 4 19 54 31 47 29 12 22 55 51 34 51 20 21 18 55 54 48 24 6 29 15 20 47 20 38
- Dane wyjściowe (sortowanie przez zliczanie): 4 4 6 9 12 15 18 19 20 20 20 21 22 24 26 29 29 31 34 38 39 47 47 48 51 51 54 54 55 55
- Dane wyjściowe (sortowanie przez kopcowanie): 4 4 6 9 12 15 18 19 20 20 20 21 22 24 26 29 29 31 34 38 39 47 47 48 51 51 54 54 55 55

e) Liczba elementów w tablicy: 100

- Dane wejściowe: 229 230 65 64 59 176 81 48 24 246 194 130 277 286 200 151 187 178 89 64 13 273 175 219 35 80 128 71 281 140 6 64 27 296 218 54 24 136 158 55 225 98 96 96 3 119 167 42 179 76 290 238 101 18 58 190 262 191 208 160 240 82 223 294 102 172 16 152 64 127 236 12 70 1 298 119 256 145 120 211 132 271 92 269 23 218 229 38 270 48 226 28 106 280 34 277 2 206 205 158
- Dane wyjściowe (sortowanie przez zliczanie): 1 2 3 6 12 13 16 18 23 24 24 27 28 34 35 38 42 48 48 54 55 58 59 64 64 64 64 65 70 71 76 80 81 82 89 92 96 96 98 101 102 106 119 119 120 127 128 130 132 136 140 145 151 152 158 158 160 167 172 175 176 178 179 187 190 191 194 200 205 206 208 211 218 218 219 223 225 226 229 229 230 236 238 240 246 256 262 269 270 271 273 277 277 280 281 286 290 294 296 298
- Dane wyjściowe (sortowanie przez kopcowanie): 1 2 3 6 12 13 16 18 23 24 24 27 28 34 35 38 42 48 48 54 55 58 59 64 64 64 64 65 70 71 76 80 81 82 89 92 96 96 98 101 102 106 119 119 120 127 128 130 132 136 140 145 151 152 158 158 160 167 172 175 176 178 179 187 190 191 194 200 205 206 208 211 218 218 219 223 225 226 229 229 230 236 238 240 246 256 262 269 270 271 273 277 277 280 281 286 290 294 296 298