Algorytmy i złożoność

Duży projekt № 1

Kiryl Paheryla

Najpierw kilka słów o tym, jak prowadziłem testy.

Aby obliczyć czas działania algorytmów, skorzystałem z biblioteki <chrono>.

Kod:

```
45  auto begin = std::chrono::steady_clock::now();
46  QuickSort(tab, 0, rozm-1);
47  auto end = std::chrono::steady_clock::now();
48  auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
49  cout << elapsed_ms.count() << endl;
```

Czas w milisekundach.

Aby ułatwić sobie przenoszenie danych do wykresu, skorzystałem z biblioteki <fstream>. Najpierw zapisywałem wszystkie dane do pliku czas.txt, a potem skopiowałem do wykresu.

Kod:

```
49  ofstream out("czas.txt", ios_base::app);
50  out << elapsed_ms.count() << endl;
51  out.close();
```

Porównanie różnych strategii wyboru elementu osiowego w sortowaniu szybkim.

Kod:

1. Wybór elementu skrajnego

```
 8  int PartitionElementuSkrajnego(int tab[], int start, int stop)
 9  {
10      int pivot = tab[stop];
11      int j = start;
12      for(int i = start; i < stop; i++)
13          if(tab[i] <= pivot) swap(tab[i], tab[j++]);
14      swap(tab[stop], tab[j]);
15      return j;
16  }
30  void QuickSort(int tab[], int start, int stop)
31  {
32      if(start >= stop) return;
33      int index = PartitionElementuSkrajnego(tab, start, stop);
34      QuickSort(tab, start, index-1);
35      QuickSort(tab, index+1, stop);
36  }
```

2. Wybór pseudolosowy

```
18  int PartitionPseudolosowy(int tab[], int start, int stop)
19  {
20      int los = rand()%(stop - start + 1) + start;
21      swap(tab[los], tab[stop]);
22      int pivot = tab[stop];
23      int j = start;
24      for(int i = start; i < stop; i++)
25          if(tab[i] <= pivot) swap(tab[i], tab[j++]);
26      swap(tab[stop], tab[j]);
27      return j;
28  }
30  void QuickSort(int tab[], int start, int stop)
31  {
32      if(start >= stop) return;
33      int index = PartitionPseudolosowy(tab, start, stop);
34      QuickSort(tab, start, index-1);
35      QuickSort(tab, index+1, stop);
36  }
```

3. Wybór mediany z 3 elementów

```
29  int WyborMedianyZ3elementow(int tab[], int start, int stop)
30  {
31      int middle = (start + stop) / 2;
32      if (tab[middle] < tab[start])
33          swap(tab[start], tab[middle]);
34      if (tab[stop] < tab[start])
35          swap(tab[start], tab[stop]);
36      if (tab[stop] < tab[middle])
37          swap(tab[middle], tab[stop]);
38      return middle;
39  }
41  int PartitionWyborMedianyZ3elementow(int tab[], int start, int stop)
42  {
43      int mediana = WyborMedianyZ3elementow(tab, start, stop);
44      swap(tab[mediana], tab[stop]);
45      int pivot = tab[stop];
46      int j = start;
47      for(int i = start; i < stop; i++)
48          if(tab[i] <= pivot)
49              swap(tab[i], tab[j++]);
50      swap(tab[stop], tab[j]);
51      return j;
52  }
54  void QuickSort(int tab[], int start, int stop)
55  {
56      if(start >= stop) return;
57      int index = PartitionWyborMedianyZ3elementow(tab, start, stop);
58      QuickSort(tab, start, index-1);
59      QuickSort(tab, index+1, stop);
```

4. Wybór mediany z 5 elementów

```
54  int InsertSort(int tab[], int temptab[], int rozm)
55  {
56      for(int i = 1; i < rozm; i++)
57      {
58          int temp = temptab[i];
59          int j = i - 1;
60          while(j >= 0 && tab[temp] < tab[temptab[j]])
61          {
62              temptab[j+1] = temptab[j];
63              j--;
64          }
65          temptab[j+1] = temp;
66      }
67      if(rozm = 5) return temptab[2];
68      //if(rozm = 7) return temptab[3];
69  }
70  int WyborMedianyZ5elementow(int tab[], int start, int stop)
71  {
72      int *temptab = new int[5];
73      temptab[0] = start; temptab[1] = stop;
74      for(int i = 2; i < 5; i++)
75          temptab[i] = rand() % (stop - start + 1) + start;
76      InsertSort(tab, temptab, 5);
77
78  }
79
80  int PartitionWyborMedianyZ5elementow(int tab[], int start, int stop)
81  {
82      int median = WyborMedianyZ5elementow(tab, start, stop);
83      swap(tab[median], tab[stop]);
84
85      int pivot = tab[stop];
86      int j = start;
87      for(int i = start; i < stop; i++)
88      {
89          if(tab[i] <= pivot)
90              swap(tab[i], tab[j++]);
91      }
92      swap(tab[stop], tab[j]);
93      return j;
94  }
122 void QuickSort(int tab[], int start, int stop)
123 {
124     if(start >= stop) return;
125     int index = PartitionWyborMedianyZ5elementow(tab, start, stop);
126     QuickSort(tab, start, index-1);
127     QuickSort(tab, index+1, stop);
128 }
```

5. Wybór mediany z 7 elementów

```
54  int InsertSort(int tab[], int temptab[], int rozm)
55  {
56      for(int i = 1; i < rozm; i++)
57      {
58          int temp = temptab[i];
59          int j = i - 1;
60          while(j >= 0 && tab[temp] < tab[temptab[j]])
61          {
62              temptab[j+1] = temptab[j];
63              j--;
64          }
65          temptab[j+1] = temp;
66      }
67      //if(rozm = 5) return temptab[2];
68      if(rozm = 7) return temptab[3];
69  }
97  int WyborMedianyZ7elementow(int tab[], int start, int stop)
98  {
99      int *temptab = new int[7];
100     temptab[0] = start; temptab[1] = stop;
101     for(int i = 2; i < 7; i++)
102         temptab[i] = rand() % (stop - start + 1) + start;
103     InsertSort(tab, temptab, 7);
104
105 }
106
107 int PartitionWyborMedianyZ7elementow(int tab[], int start, int stop)
108 {
109     int median = WyborMedianyZ7elementow(tab, start, stop);
110     swap(tab[median], tab[stop]);
111
112     int pivot = tab[stop];
113     int j = start;
114     for(int i = start; i < stop; i++)
115     {
116         if(tab[i] <= pivot)
117             swap(tab[i], tab[j++]);
118     }
119     swap(tab[stop], tab[j]);
120     return j;
121 }
123 void QuickSort(int tab[], int start, int stop)
124 {
125     if(start >= stop) return;
126     int index = PartitionWyborMedianyZ7elementow(tab, start, stop);
127     QuickSort(tab, start, index-1);
128     QuickSort(tab, index+1, stop);
129 }
```

Przypadki ich wypełniania:

1. elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy

```
48    for(int i = 0; i < rozm; i++) tab[i] = rand() + rozm;
```

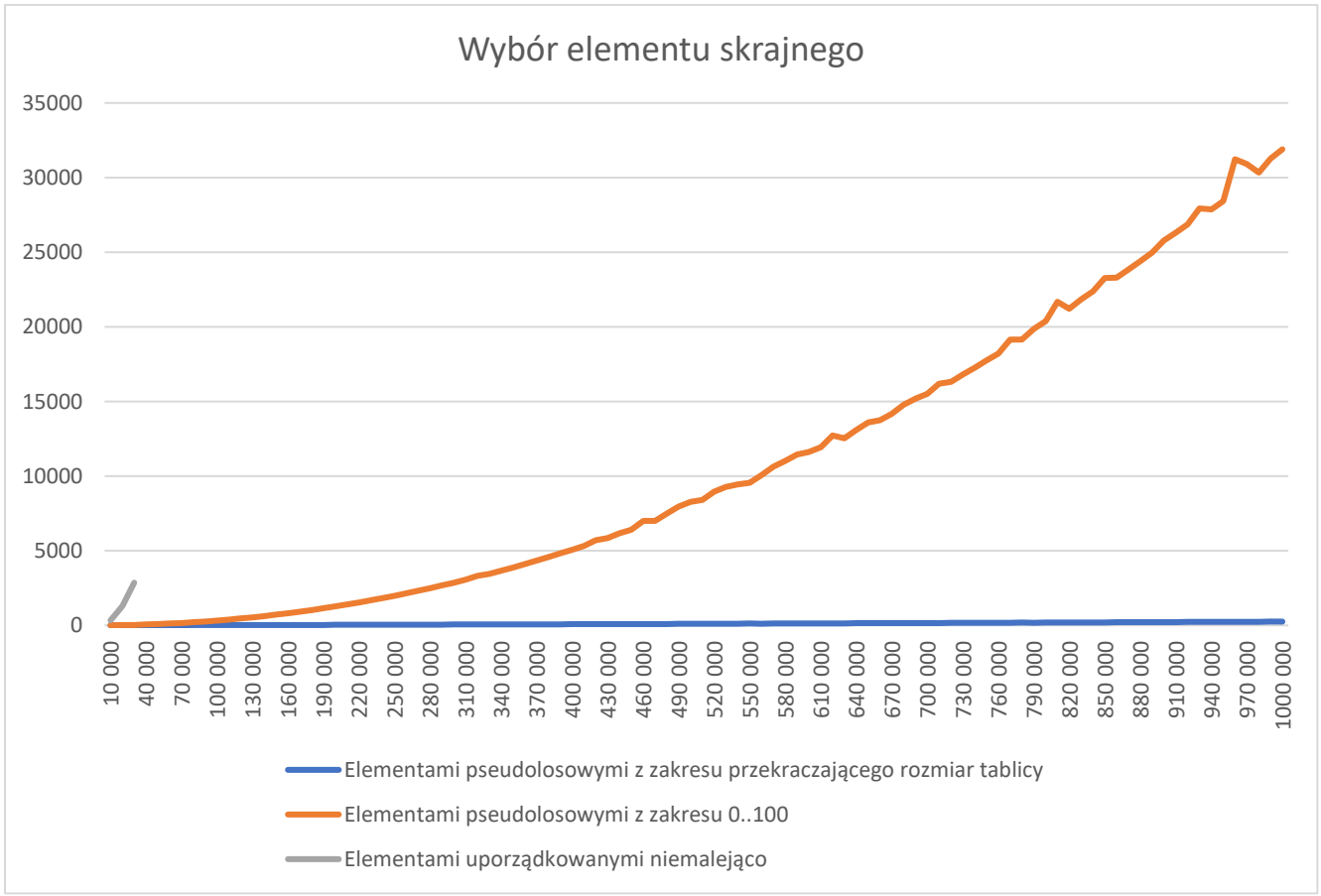
2. elementami pseudolosowymi z zakresu 0..100

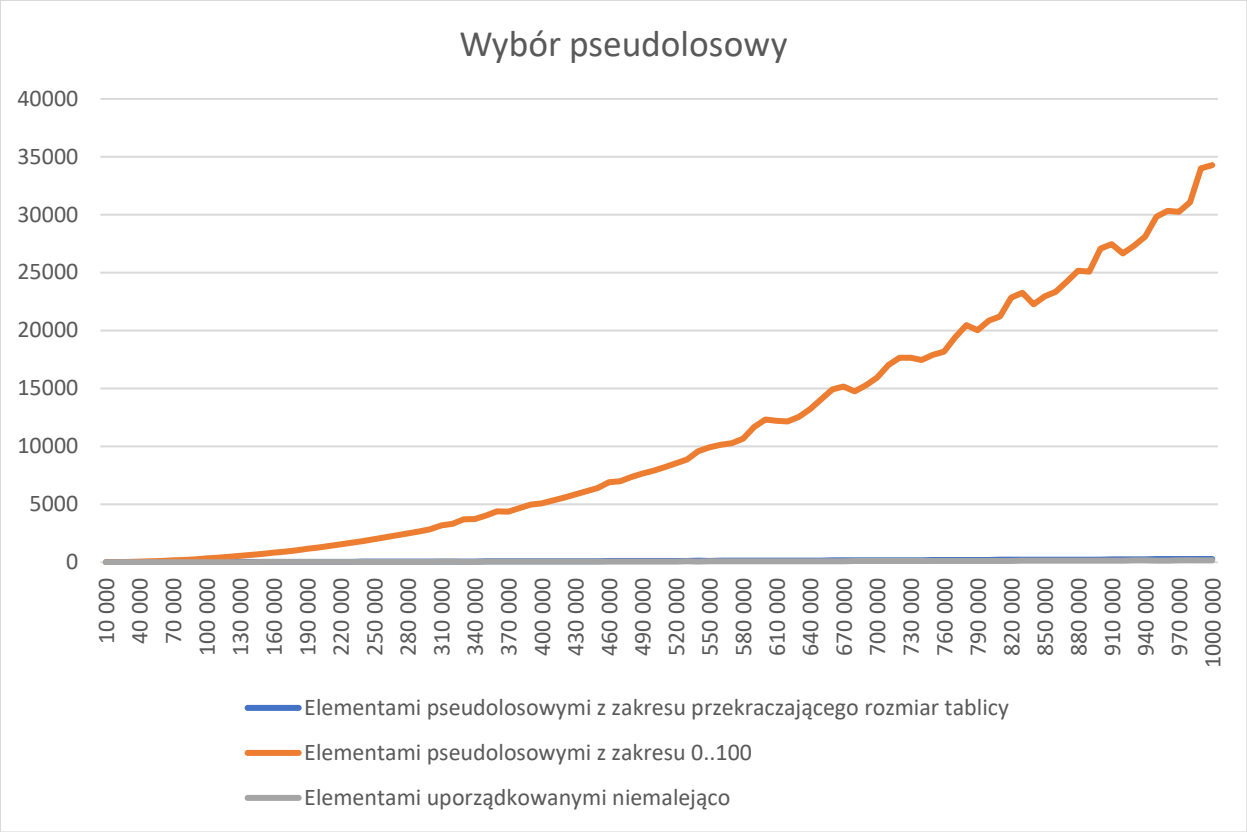
```
48    for(int i = 0; i < rozm; i++) tab[i] = rand() % (101);
```

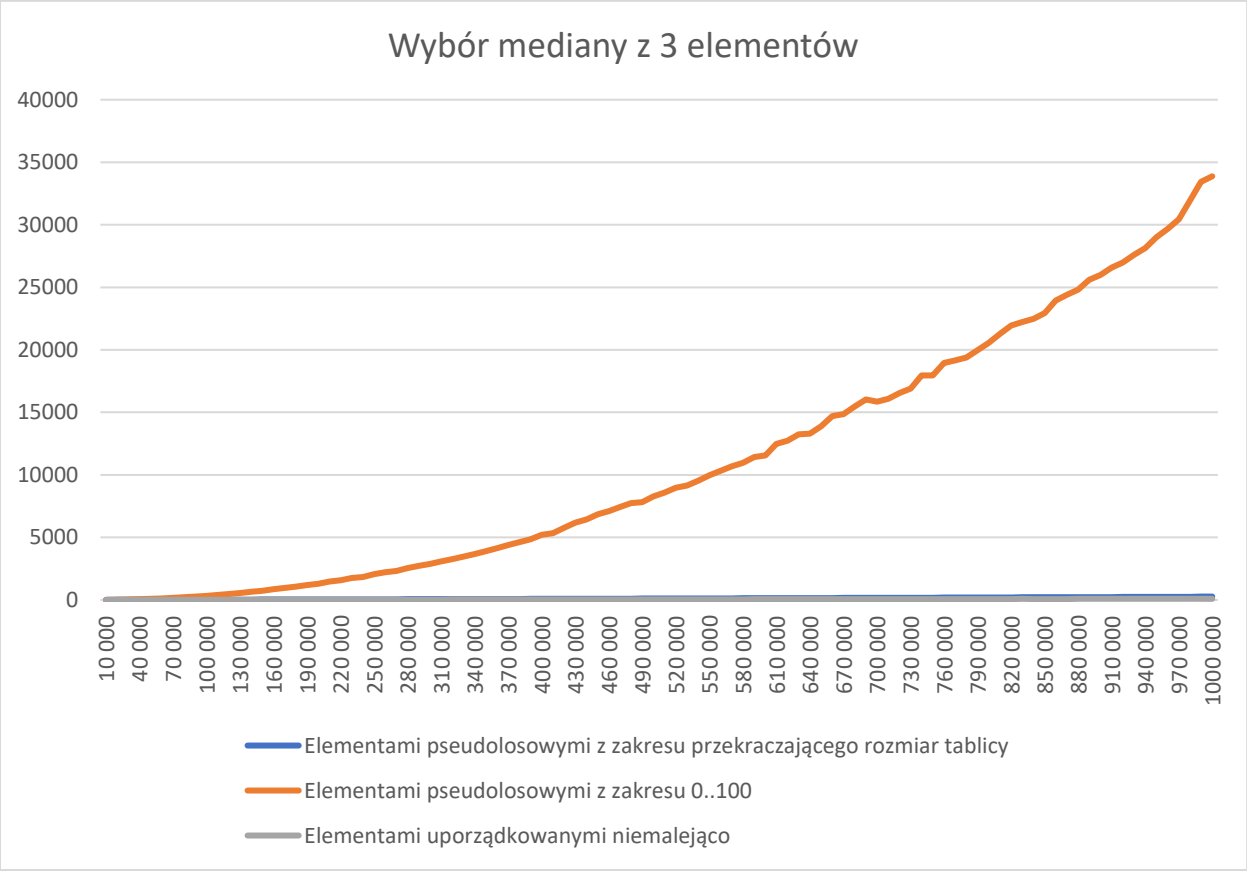
3. elementami uporządkowanymi niemalejąco

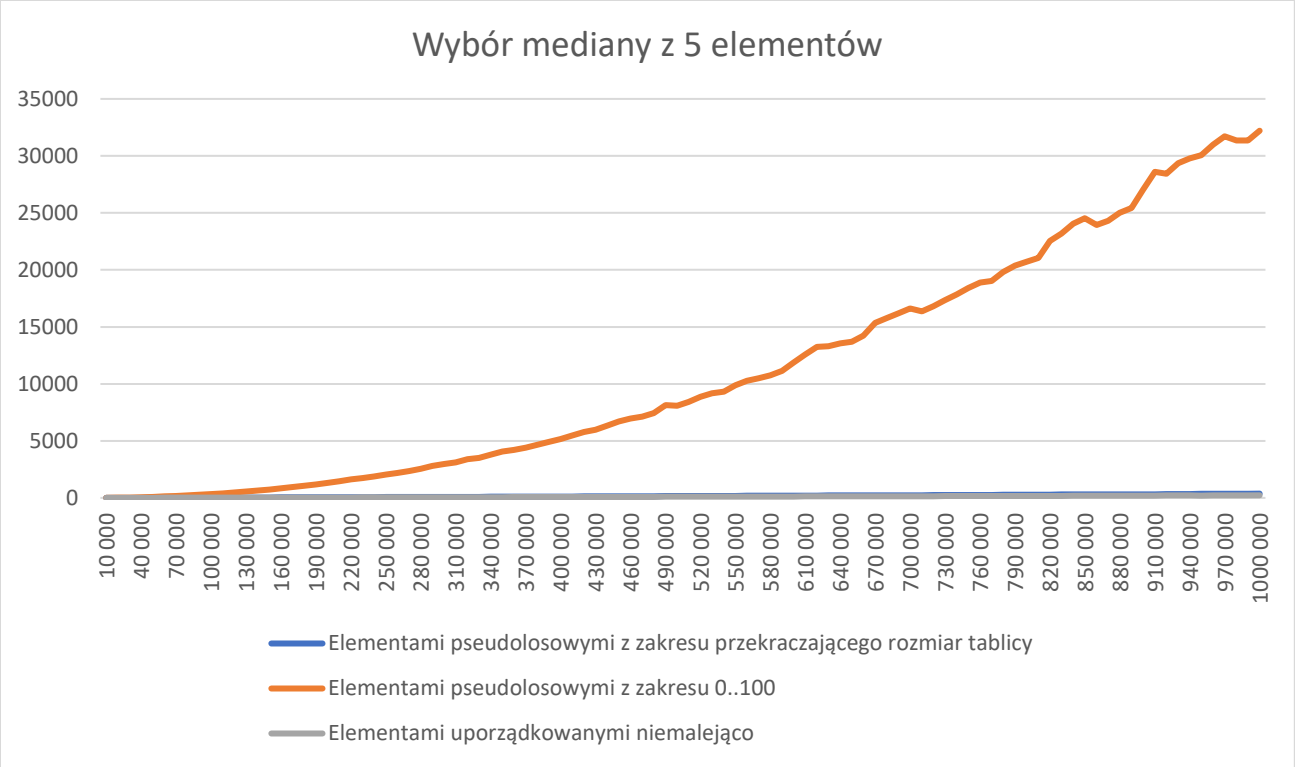
```
48    for(int i = 0; i < rozm; i++) tab[i] = i;
```

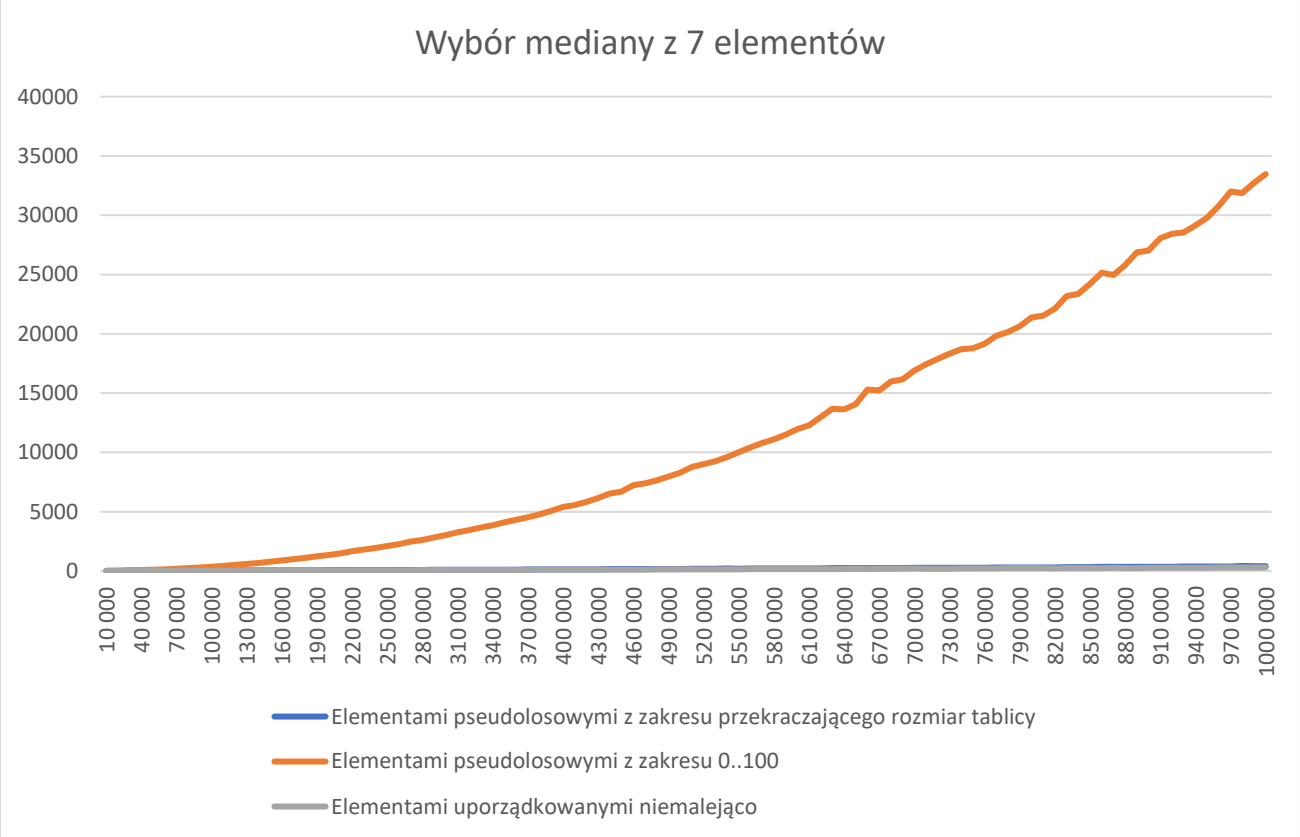
Wyniki testów na wykresach.

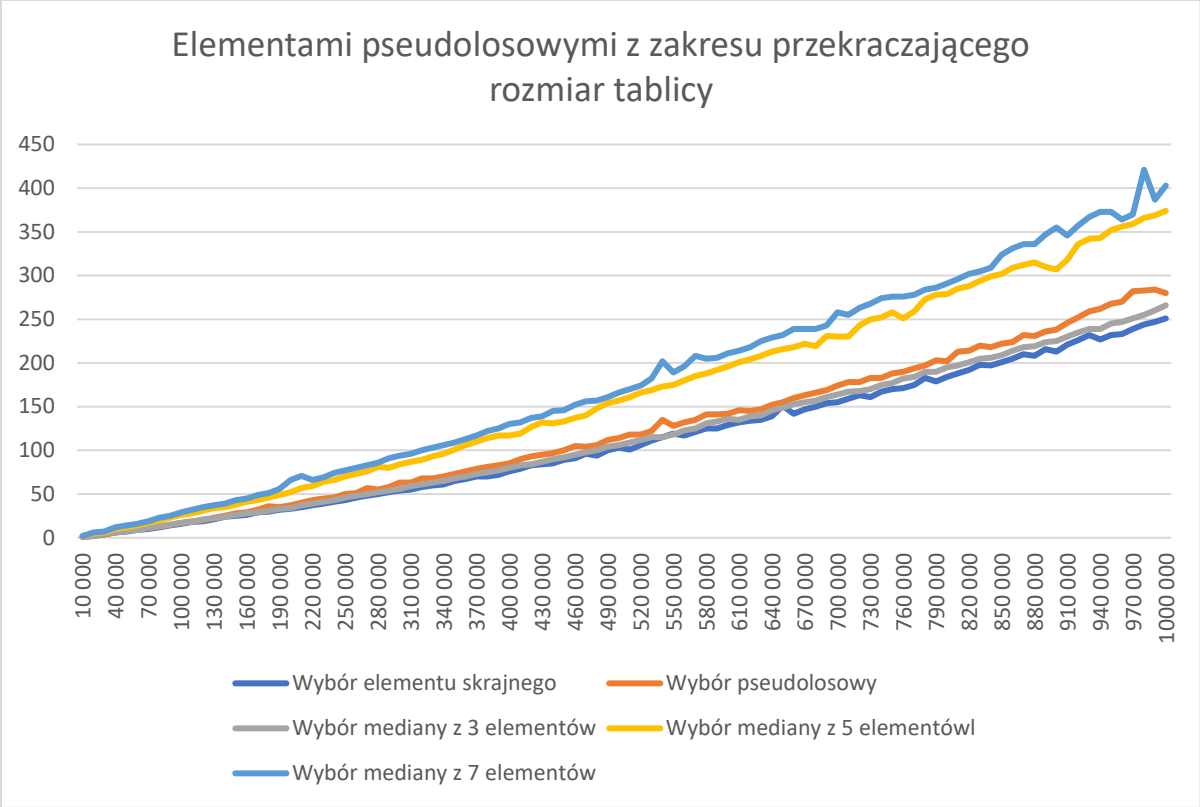


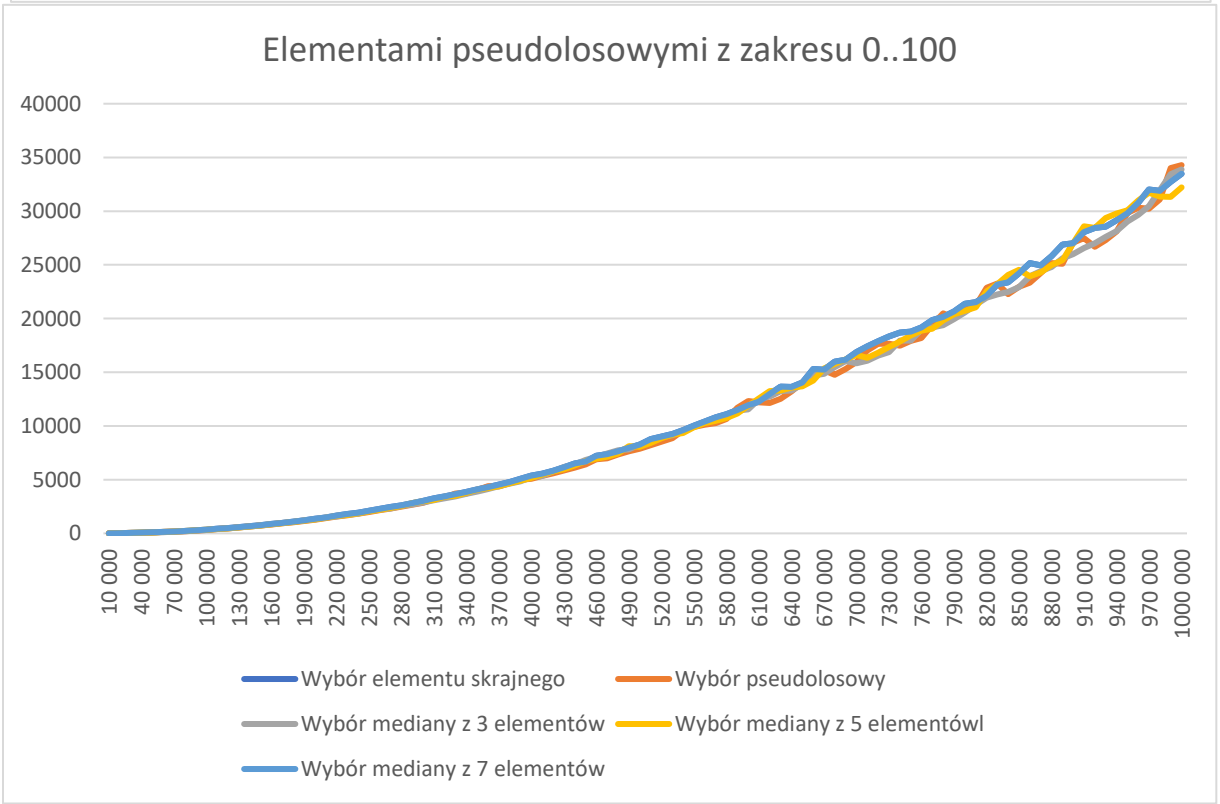

Wybór pseudolosowy
40000
35000
30000
25000
20000
15000
10000
5000
0
10 000
40 000
70 000
100 000
130 000
160 000
190 000
220 000
250 000
280 000
310 000
340 000
370 000
400 000
430 000
460 000
490 000
520 000
550 000
580 000
610 000
640 000
670 000
700 000
730 000
760 000
790 000
820 000
850 000
880 000
910 000
940 000
970 000
1000 000
Elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy
Elementami pseudolosowymi z zakresu 0..100
Elementami uporządkowanymi niemalejąco

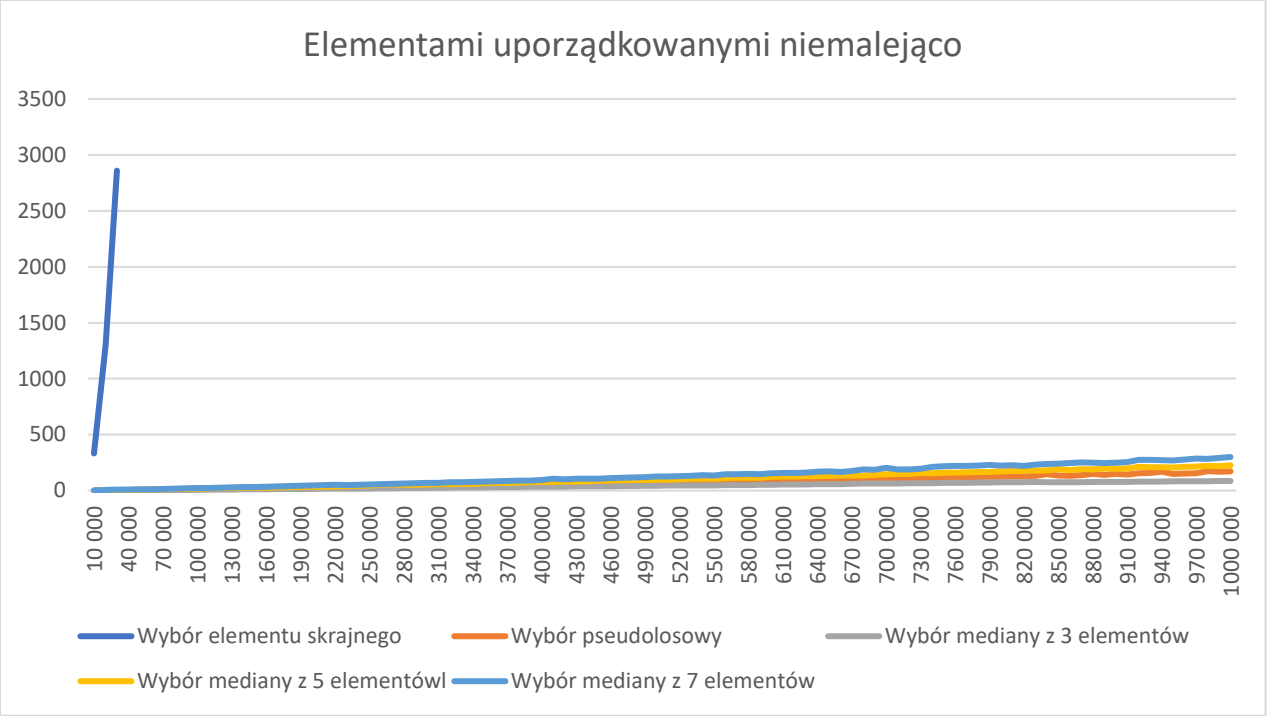

Wybór mediany z 3 elementów
40000
35000
30000
25000
20000
15000
10000
5000
0
10 000
40 000
70 000
100 000
130 000
160 000
190 000
220 000
250 000
280 000
310 000
340 000
370 000
400 000
430 000
460 000
490 000
520 000
550 000
580 000
610 000
640 000
670 000
700 000
730 000
760 000
790 000
820 000
850 000
880 000
910 000
940 000
970 000
1000 000
Elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy
Elementami pseudolosowymi z zakresu 0..100
Elementami uporządkowanymi niemalejąco


Wybór mediany z 5 elementów
35000
30000
25000
20000
15000
10000
5000
0
10 000
40 000
70 000
100 000
130 000
160 000
190 000
220 000
250 000
280 000
310 000
340 000
370 000
400 000
430 000
460 000
490 000
520 000
550 000
580 000
610 000
640 000
670 000
700 000
730 000
760 000
790 000
820 000
850 000
880 000
910 000
940 000
970 000
1000 000
Elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy
Elementami pseudolosowymi z zakresu 0..100
Elementami uporządkowanymi niemalejąco


Wybór mediany z 7 elementów
40000
35000
30000
25000
20000
15000
10000
5000
0
10 000
40 000
70 000
100 000
130 000
160 000
190 000
220 000
250 000
280 000
310 000
340 000
370 000
400 000
430 000
460 000
490 000
520 000
550 000
580 000
610 000
640 000
670 000
700 000
730 000
760 000
790 000
820 000
850 000
880 000
910 000
940 000
970 000
1000 000
Elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy
Elementami pseudolosowymi z zakresu 0..100
Elementami uporządkowanymi niemalejąco


Elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy
450
400
350
300
250
200
150
100
50
0
10 000
40 000
70 000
100 000
130 000
160 000
190 000
220 000
250 000
280 000
310 000
340 000
370 000
400 000
430 000
460 000
490 000
520 000
550 000
580 000
610 000
640 000
670 000
700 000
730 000
760 000
790 000
820 000
850 000
880 000
910 000
940 000
970 000
1000 000
Wybór elementu skrajnego
Wybór pseudolosowy
Wybór mediany z 3 elementów
Wybór mediany z 5 elementówl
Wybór mediany z 7 elementów


Elementami pseudolosowymi z zakresu 0..100
40000
35000
30000
25000
20000
15000
10000
5000
0
10 000
40 000
70 000
100 000
130 000
160 000
190 000
220 000
250 000
280 000
310 000
340 000
370 000
400 000
430 000
460 000
490 000
520 000
550 000
580 000
610 000
640 000
670 000
700 000
730 000
760 000
790 000
820 000
850 000
880 000
910 000
940 000
970 000
1000 000
Wybór elementu skrajnego
Wybór pseudolosowy
Wybór mediany z 3 elementów
Wybór mediany z 5 elementówl
Wybór mediany z 7 elementów



Wyniki testów

Podczas testów pojawił się problem. To był QuickSort przez wybór elementu skrajnego, a tablica generowana elementami uporządkowanymi niemalejąco. Od tablicy rozmiarem 40 000 program nie działał. Widzimy, że czas działania wynosi $\Theta(n^2)$. W szczególności dzieje się tak, gdy tablica jest początkowo sortowana.

Dla wyboru elementu skrajnego najlepiej tablica, która generowana elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy.

Dla wyboru pseudolosowego, mediany z 3 elementów, mediany z 5 elementów, mediany z 7 elementów najlepsza tablica, która generowana elementami uporządkowanymi niemalejąco lub elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy.

Najgorsza opcja dla wszystkich to tablica, która generowana elementami pseudolosowymi z zakresu 0..100.

Dla tablicy, która generowana elementami pseudolosowymi z zakresu przekraczającego rozmiar tablicy najszybszy wybór elementu skrajnego, a najgorszy wybór mediany z 7 elementów.

Dla tablicy, która generowana elementami pseudolosowymi z zakresu 0..100 najszybszy wybór mediany z 5 elementów, a najgorszy wybór pseudolosowy.

Dla tablicy, która generowana elementami uporządkowanymi niemalejąco najszybszy wybór mediany z 3 elementów, a najgorszy wybór elementu skrajnego.

Czas działania:

najgorzej: $T(N) = T(0) + T(N-1) + N = T(N-1) + N = O(N^2)$

Podciągi zawsze mają długości 0 i N-1 (el. Osiowy jest zawsze najmniejszy/największy). Np. dla posortowanego ciągu i pierwszej opcji wyboru el. osiowego.

najlepiej: $T(N) = 2T(N/2) + N = O(N\log N)$

Podział jest zawsze najlepszy (N/2). El. osiowy zawsze jest medianą.