Politechnika
Wrocławska,
Wydział Informatyki
i Telekomunikacji
Semestr IV Rok II

Zadanie projektowe nr 1

Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych

Wykonał:
Patryk Ignasiak 263889
Gr. Wtorek 15:15 P
Prowadzący:
Mgr. Inż. Antoni Sterna
Przedmiot:
Struktury danych i złożoność
obliczeniowa - Projekt

Wrocław, 17 kwietnia 2023

Spis treści

Wstęp

W tym projekcie zostały zaimplementowane takie struktury jak: Tablica dynamiczna, lista dwukierunkowa, kopiec binarny, drzewo przeszukiwań binarnych (brak testów czasowych) oraz drzewo czerwono-czarne. Na podanych strukturach zostały przetestowane poniższe procedury:

- Dodawanie

- Usuwanie

- Wyszukiwanie

Złożoności obliczeniowe na podstawie literatury:

- Tablica Dynamiczna
  - Dodawanie na początek – O(n)
  - Dodawanie na wybraną pozycję – O(n)
  - Dodawanie na koniec – O(n)
  - Usuwanie z początku – O(n)
  - Usuwanie z wybranej pozycji – O(n)
  - Usuwanie z końca – O(n)
  - Wyszukiwanie – O(n)
- Lista Dwukierunkowa
  - Dodawanie na początek – O(1)
  - Dodawanie na wybraną pozycję – O(n)
  - Dodawanie na koniec – O(1)
  - Usuwanie z początku – O(1)
  - Usuwanie z wybranej pozycji – O(n)
  - Usuwanie z końca – O(1)
  - Wyszukiwanie – O(n)
- Kopiec binarny
  - Dodawanie – O(n)
  - Usuwanie ze szczytu – O(n)
  - Wyszukiwanie – O(n)
  - Standardowe budowanie –O(n log n)
  - Algorytm Floyda – O(n)
- Drzewo przeszukiwań Binarnych
  - Dodawanie – O(n)
  - Usuwanie ze szczytu – O(n)
  - Wyszukiwanie – optymistyczne O(log n), pesymistyczne O(n)
  - Algorytm DSW – O(n)
- Drzewo czerwono-czarne
  - Dodawanie – O(log n)
  - Usuwanie ze szczytu – O(log n)

- Wyszukiwanie – O(log n)

Testy czasowe nie zostały przeprowadzone dla drzewa BST, problemem była potrzeba wykonania algorytmu DSW po każdej operacji co powodowało problemy podczas testów. Algorytm działa poprawnie, ale dla większych struktur powodował zawieszenie programu w testach automatycznych, a w półautomatycznych pojawiały się dziwne czasy jak np. 35ns dla drzewa z milionem węzłów. Z pozostałymi strukturami nie było problemów, aby przeprowadzić testy do nawet 2,5 miliona danych. Dla kopca 2,25 miliona, ponieważ później również generowały się niepoprawne czasy. Możliwe, że wynika to z wykorzystania „Chrono" do pomiarów.

Opis eksperymentu

Eksperymenty na poszczególnych strukturach odbywały się przy użyciu własnoręcznie napisanego algorytmu. Algorytm testował poszczególne operacje 20 razy dla każdego z 19 rozmiarów struktur. Rozmiary: 10, 50, 100, 500, 1 000, 5 000, 10 000, 50 000, 100 000, 250 000, 500 000, 750 000, 1 000 000, 1 250 000, 1 500 000, 1 750 000, 2 000 000, 2 250 000, 2 500 000.

Czasy były mierzone z wykorzystaniem „CHRONO" z dokładnością do nanosekund i zapisywane w tablicy typu „ long long" po zakończeniu wszystkich testów, z wykorzystaniem metody „saveTimeTestResult" pochodzącej z klasy „Data", średnie czasy (czyli podzielone przez 20) zostały zapisane w plikach „.txt". Następnie z wykorzystaniem programu „Excel" uzyskane wyniki zostały obrobione.

Po załadowaniu losowo wygenerowanych danych do struktury testy czasu odbywały się w sposób przedstawiony poniżej:

```
int randomNumber = randomInt();           //losowa wartość do dodania
auto start = chrono::steady_clock::now(); //zapisanie czasu startowego
array->pushFront(randomNumber);           //testowana procedura
auto end = chrono::steady_clock::now();   //zapisanie czasu końcowego
auto duration = end - start;              //obliczenie czasu trwania

//zapisanie do tablicy
times[i][0] += chrono::duration_cast<chrono::nanoseconds>(duration).count();
```

Poniżej zostały przedstawione wyniki pomiarów. Pomiary przeprowadzone zostały na laptopie Lenovo ThinkPad T14. Podstawowe parametry:

- RAM: 32GB

- Procesor: AMD Ryzen 7 PRO 4750U

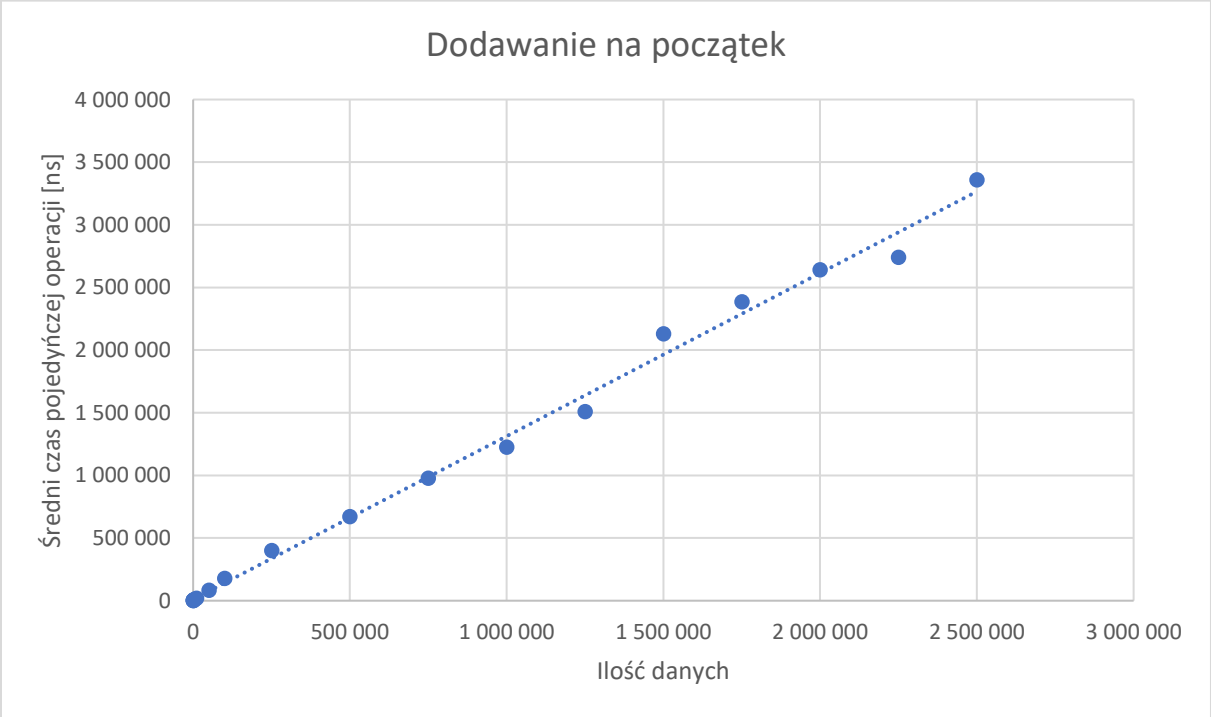
Tablica dynamiczna

Tabela czasów

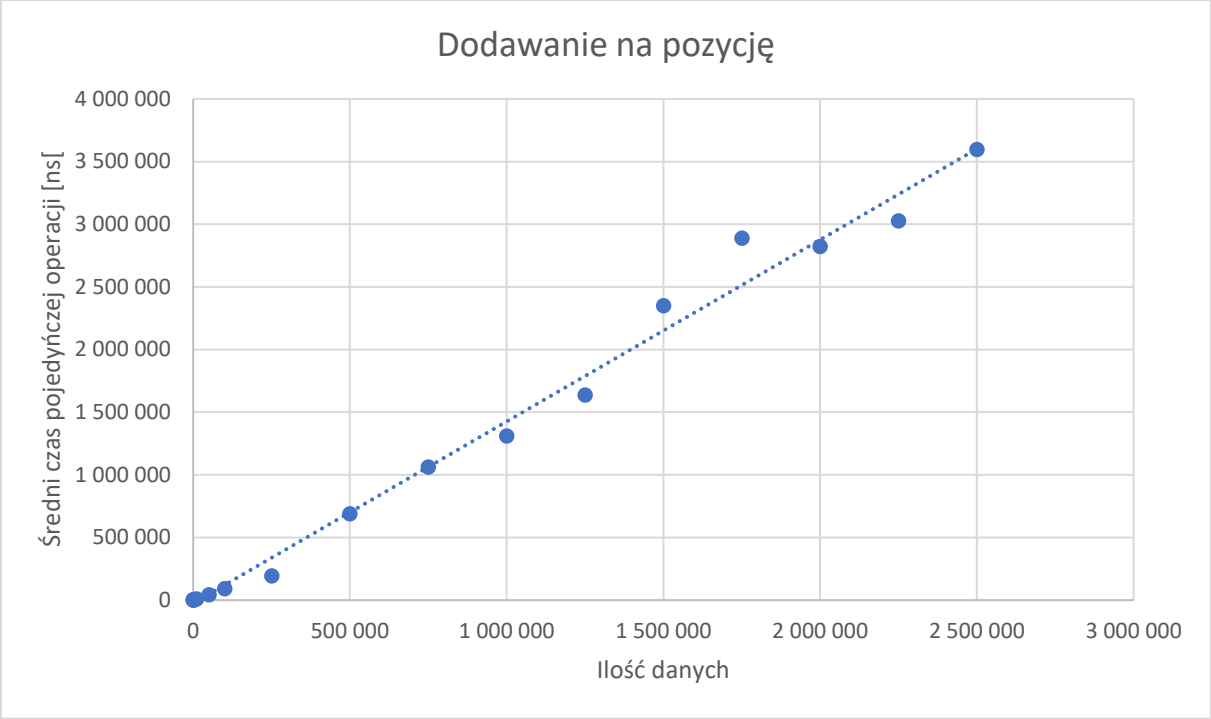
Ilość danych	Dodawanie na początek	Dodawanie na pozycję	Dodawanie na koniec	Usuwanie z początku	Usuwanie z pozycji	Usuwanie z końca	Wyszukiwanie
10	305	190	165	275	265	260	65
50	760	295	395	400	215	210	70
100	1 505	330	410	555	365	235	110
500	2 315	665	545	1 875	655	535	275
1 000	6 685	1 040	815	2 070	880	825	430
5 000	10 880	4 700	3 200	3 355	4 085	3 145	2 370
10 000	18 550	8 660	6 095	6 485	7 860	6 040	4 630
50 000	83 130	41 020	30 690	30 695	38 760	30 100	20 815
100 000	177 710	89 475	64 115	75 240	80 270	62 560	38 245
250 000	398 410	193 050	134 250	138 660	174 790	134 375	73 885
500 000	668 870	687 185	612 770	614 020	674 910	667 240	168 055
750 000	976 360	1 061 315	929 515	926 305	1 015 430	935 655	277 890
1 000 000	1 224 610	1 308 190	1 175 760	1 201 350	1 285 985	1 159 460	414 470
1 250 000	1 507 445	1 635 890	1 462 265	1 445 550	1 570 325	1 438 845	510 405
1 500 000	2 129 520	2 348 560	1 963 200	1 953 535	2 218 845	1 974 140	720 640
1 750 000	2 383 465	2 887 210	2 376 690	2 341 930	2 605 990	2 448 420	796 895
2 000 000	2 640 255	2 822 260	2 465 110	2 475 585	2 669 870	2 476 505	859 805
2 250 000	2 739 040	3 026 335	2 627 900	2 624 930	2 955 620	2 657 725	865 210
2 500 000	3 357 015	3 596 650	3 325 635	3 248 710	3 267 215	3 342 490	884 955

Dodawanie

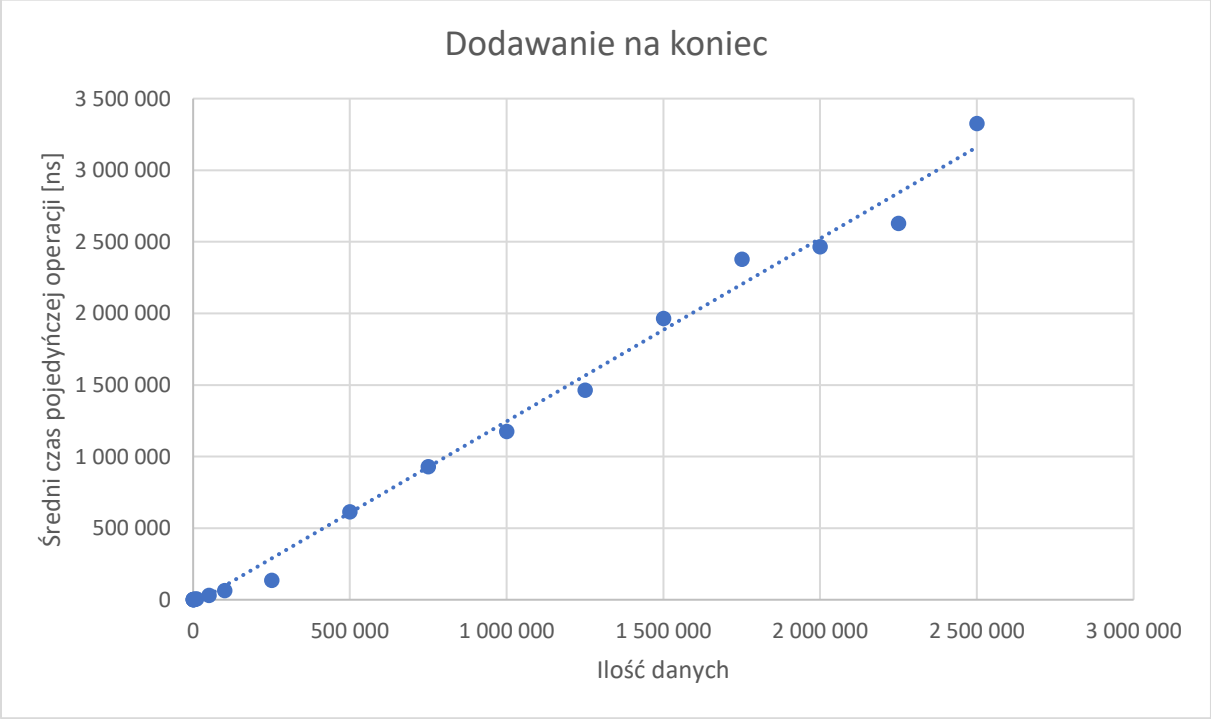
Na początek



Na wybraną pozycję

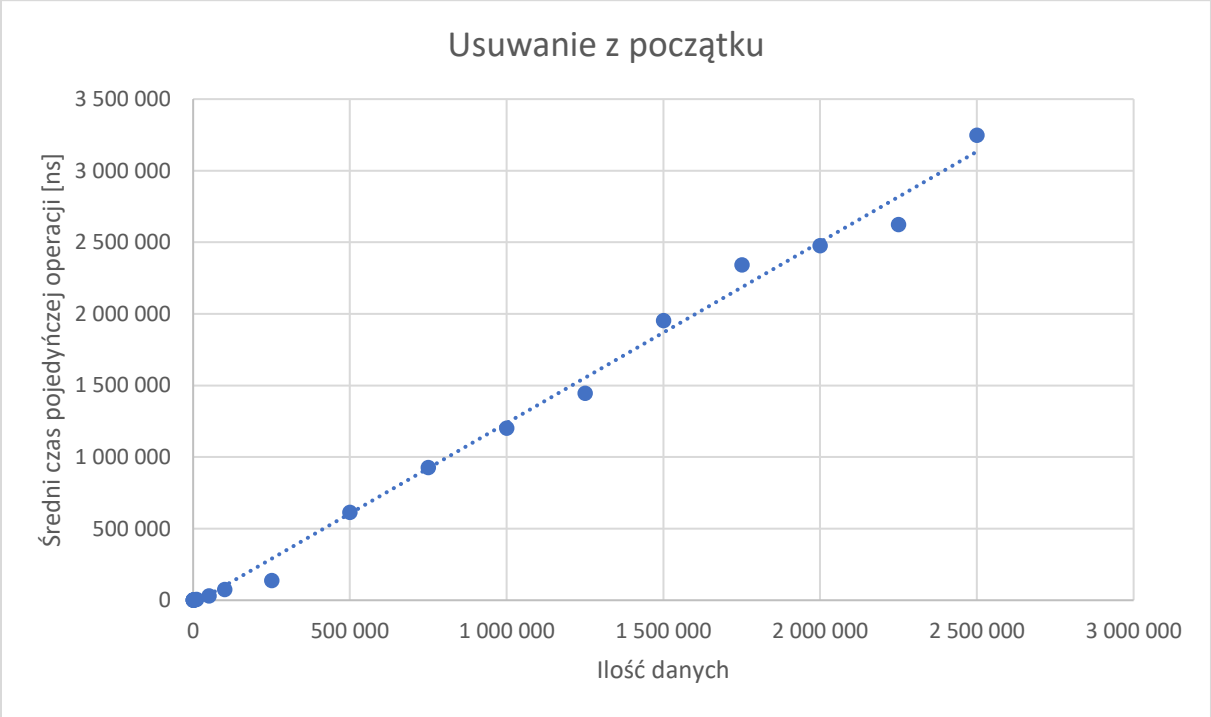


Na koniec

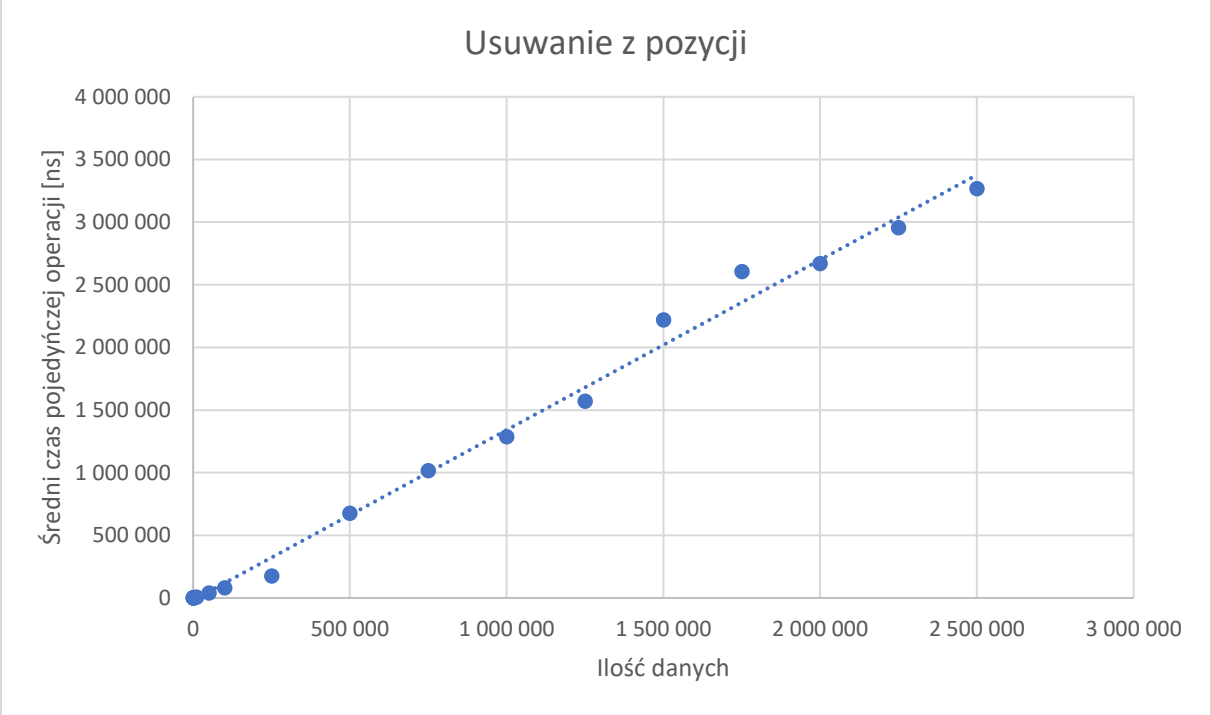


Usuwanie

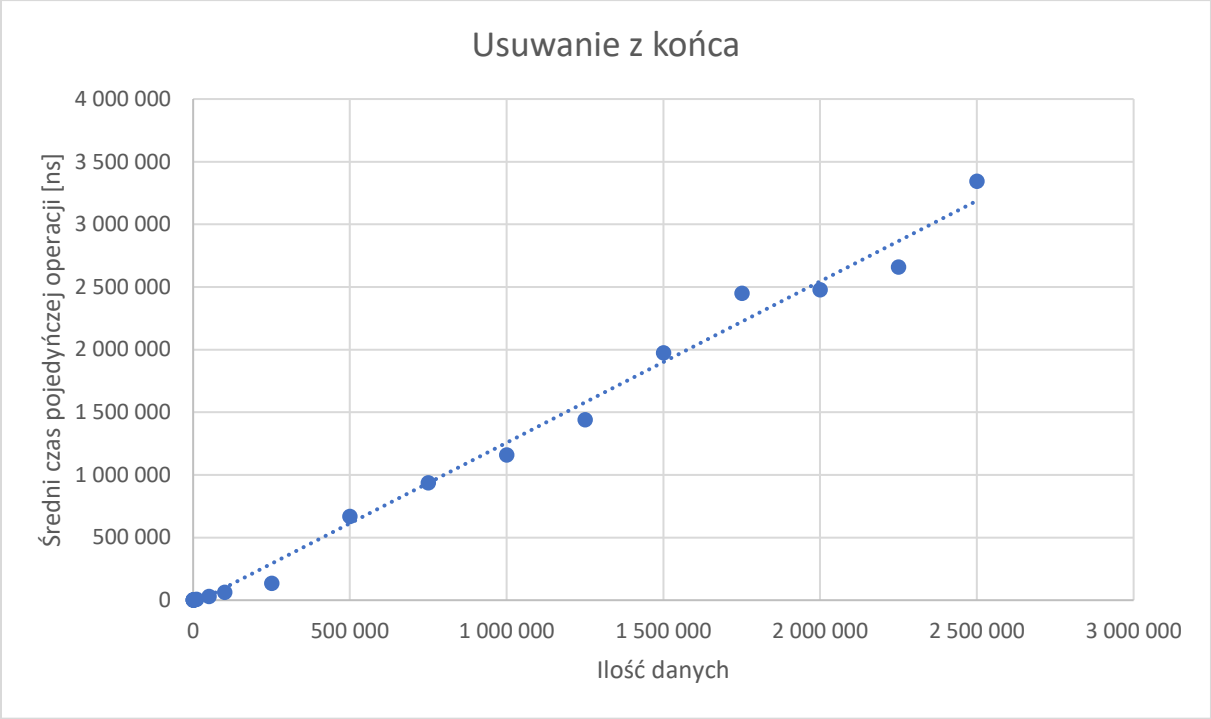
Z początku



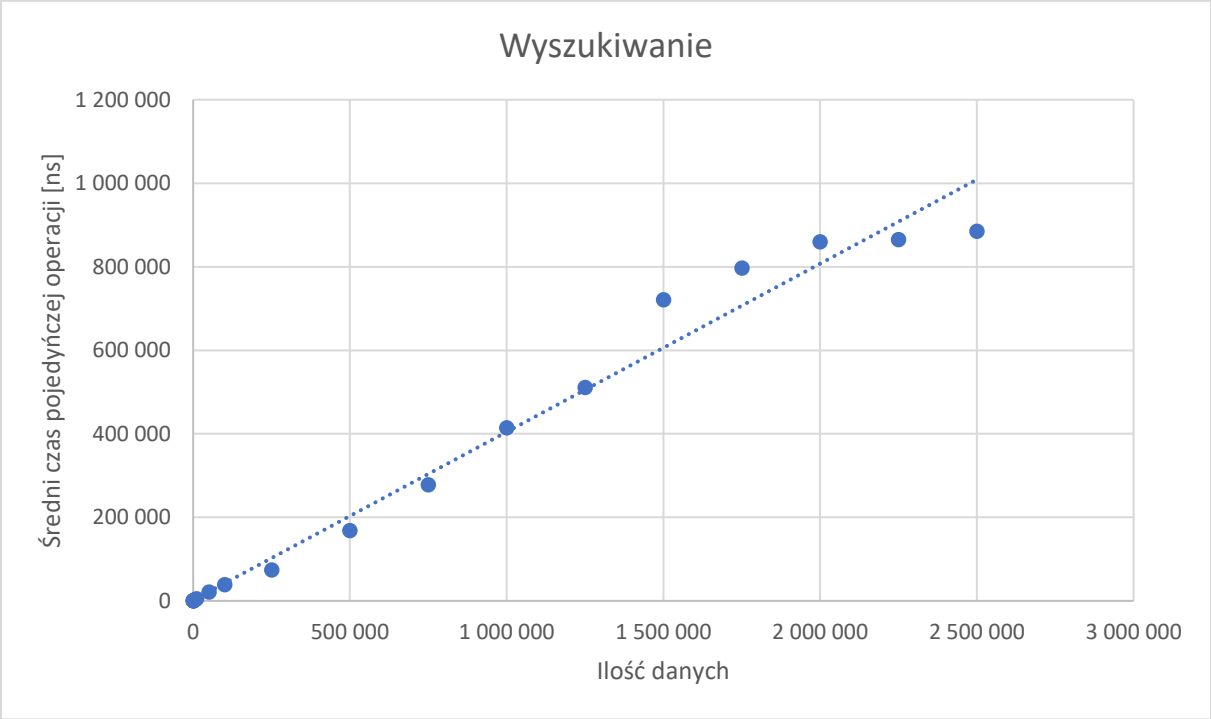
Z wybranej pozycji



Z końca



Wyszukiwanie



Podsumowanie

Dodawanie i usuwanie w tablicy dynamicznej, jak można zauważyć na powyższych wykresach, są liniowo zależne od rozmiaru tablicy. Związane jest to z potrzebą relokacji tablicy dynamiczne w pamięci. Podczas wyszukiwania w tablicy dynamicznej zachodzi natomiast nieco inny proces. Zwiększamy wielkość tablicy, ale zakres losowania (INT32_MIN do INT32_MAX) zostaje cały czas ten

sam. Dzięki temu prawdopodobieństwo na znalezienie losowej wartości wzrasta. Na wykresie od 2 000 000 wartości wzwyż mamy praktycznie stały czas wyszukiwania wiąże się to z procesem opisanym powyżej.

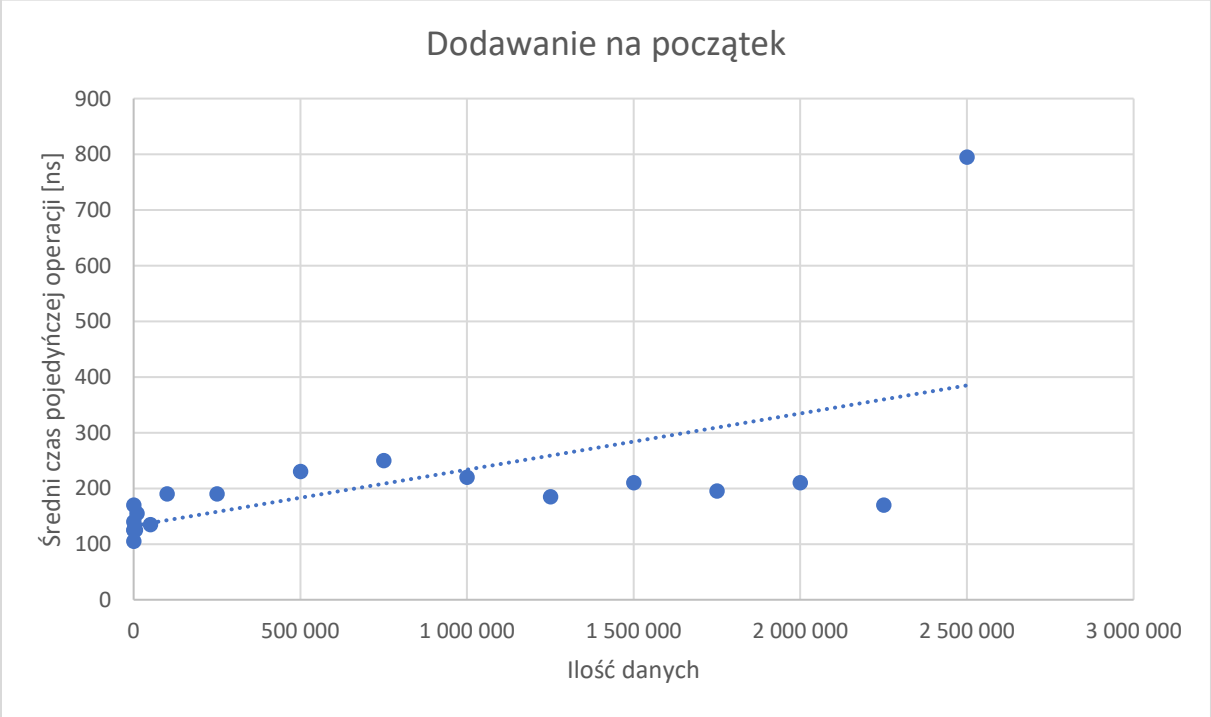
Lista dwukierunkowa

Tabela czasów

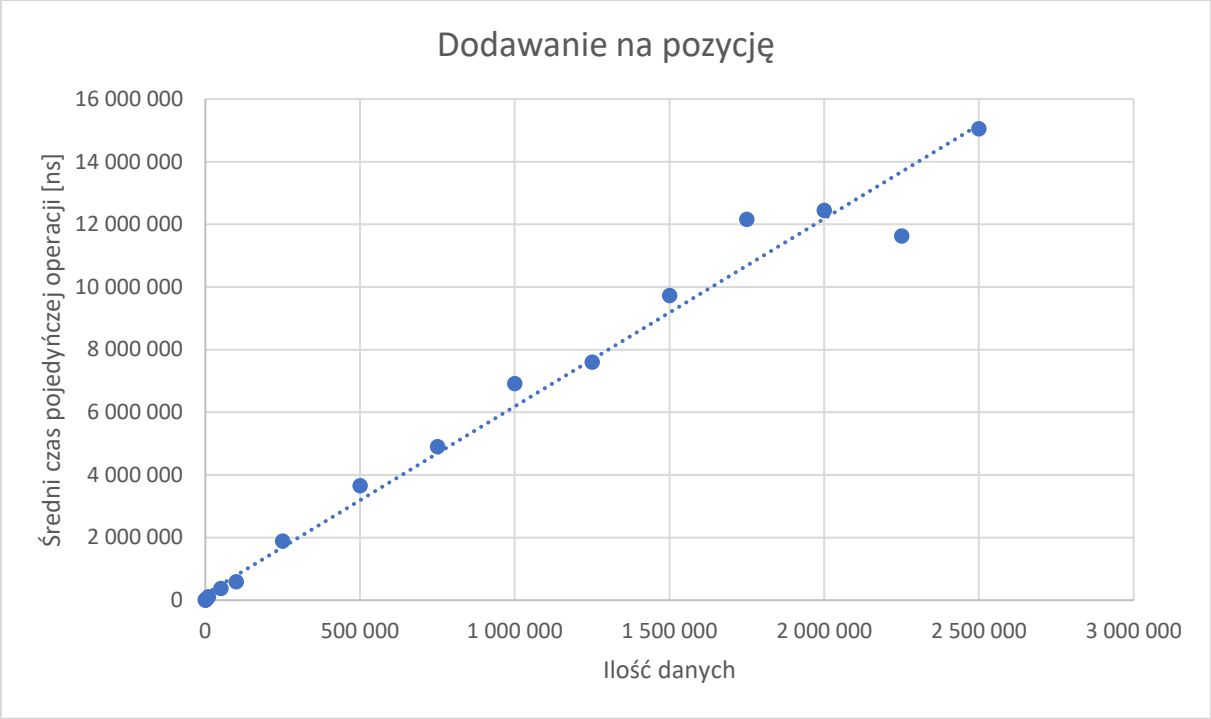
Ilość danych	Dodawanie na początek	Dodawanie na pozycję	Dodawanie na koniec	Usuwanie z początku	Usuwanie z pozycji	Usuwanie z końca	Wyszukiwanie ze zwracaniem indeksu	Wyszukiwanie ze zwracaniem elementu	Wyszukiwanie - procesor Intel
10	140	395	100	105	295	75	30	865	80
50	105	1 245	75	145	645	70	75	2 185	265
100	170	3 285	90	205	1 460	85	250	4 635	530
500	125	11 860	70	170	1 745	50	850	11 290	3 090
1 000	125	16 730	65	135	3 790	65	2 565	23 145	11 805
5 000	125	50 935	95	205	12 970	80	9 625	81 565	47 195
10 000	155	106 465	85	240	37 115	85	29 060	183 780	147 120
50 000	135	369 225	55	455	154 825	55	160 110	781 490	693 450
100 000	190	584 190	80	555	395 200	75	1 883 065	1 974 785	2 452 915
250 000	190	1 882 680	60	775	1 306 590	65	4 213 650	4 638 375	6 338 110
500 000	230	3 652 910	80	960	2 952 595	70	9 307 230	10 028 095	13 211 100
750 000	250	4 898 125	70	1 080	4 305 975	70	16 759 595	17 846 065	25 806 985
1 000 000	220	6 907 140	65	860	6 338 160	70	29 719 010	28 121 910	41 719 420
1 250 000	185	7 593 960	70	1 160	8 795 740	70	40 817 705	40 468 435	58 480 080
1 500 000	210	9 718 395	80	955	9 593 565	80	55 776 610	56 337 915	79 381 425
1 750 000	195	12 152 450	70	1 050	10 439 140	65	73 836 795	74 093 650	124 118 365
2 000 000	210	12 441 010	165	1 235	14 510 295	70	95 149 075	94 208 130	140 451 875
2 250 000	170	11 627 875	145	1 140	14 803 935	60	116 130 585	117 128 615	189 577 420
2 500 000	795	15 050 495	110	225	17 967 125	80	141 716 125	143 689 490	209 552 075

Dodawanie

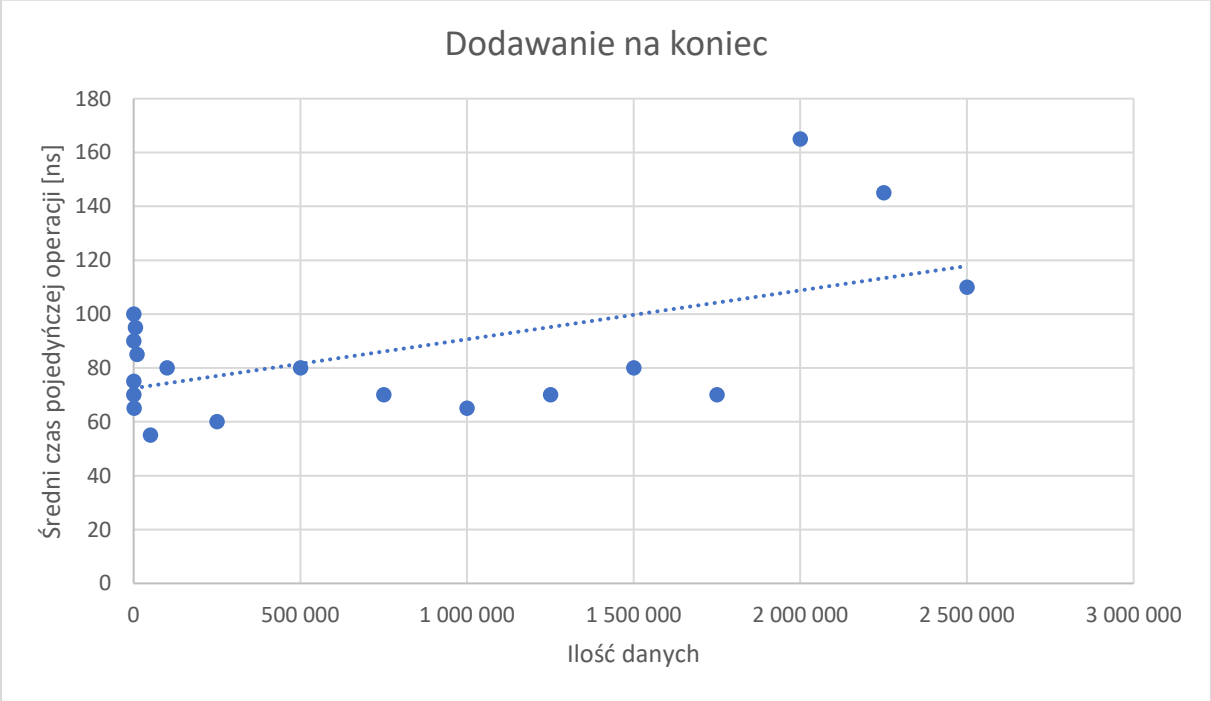
Na początek



Na wybraną pozycję



Na koniec



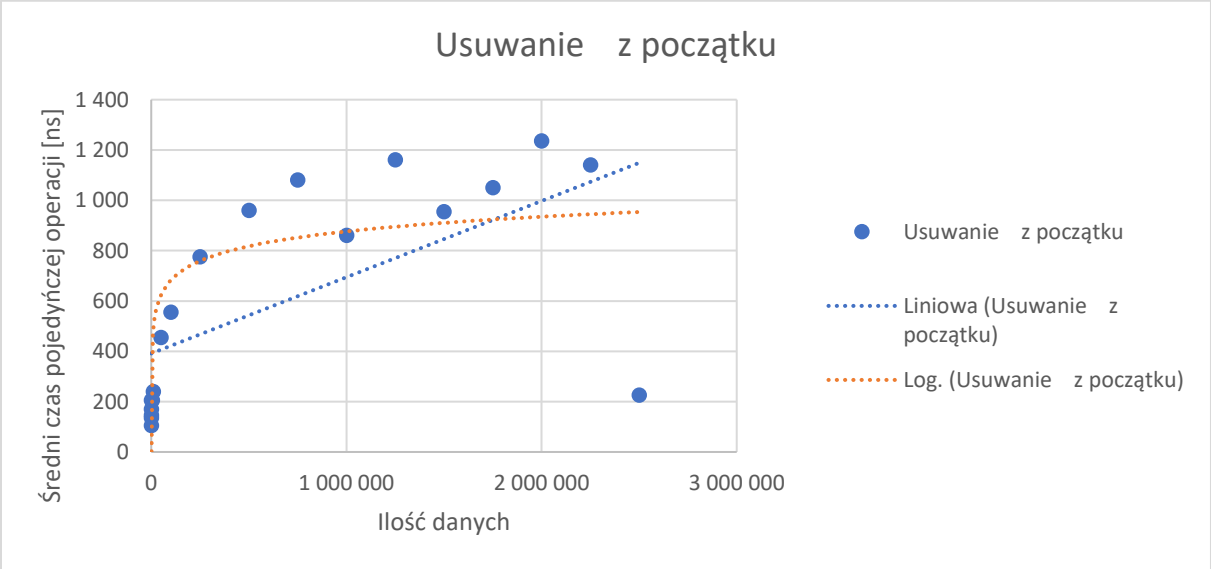
Podsumowanie

Z racji zastosowania listy dwukierunkowej (posiada głowę i ogon) dodawanie na początek i koniec powinno mieć złożoność jednostkową O(1). Jednak dla usuwania z początku, przypomina bardziej złożoność O(log n). Może to być spowodowane jakimś procesem, który zaczął działać w tle, gdyż dla 2,5 miliona

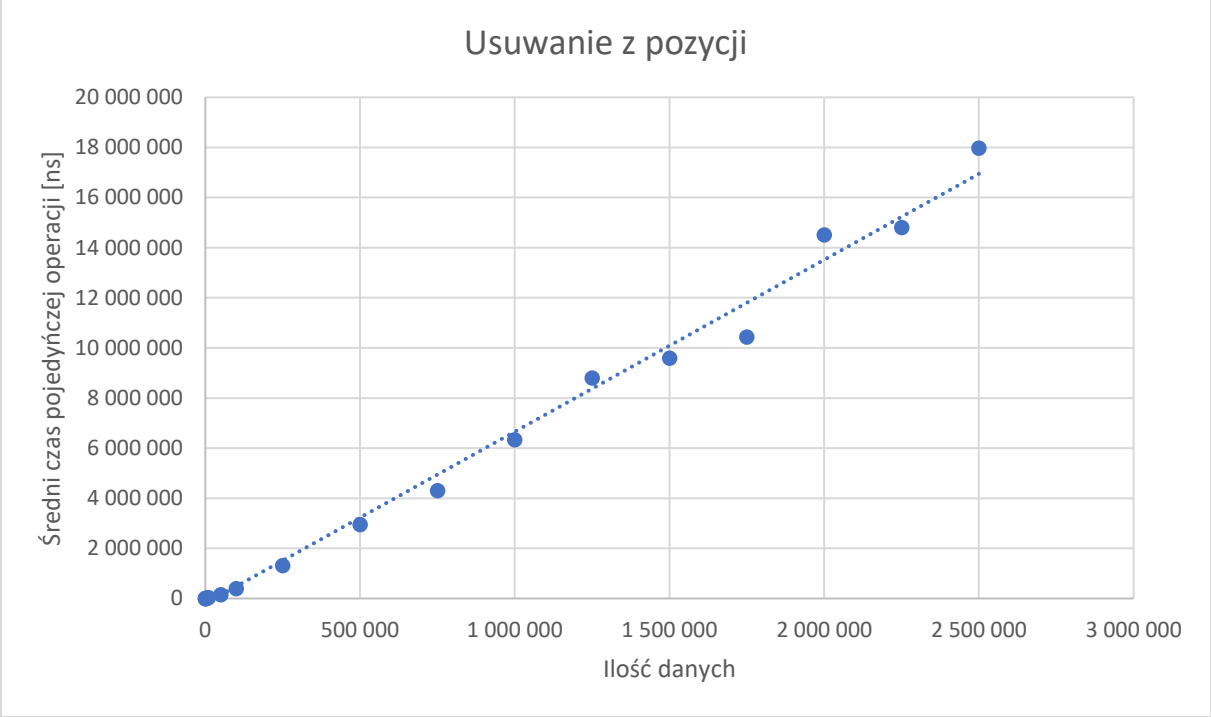
wróciło do poprawnych wartości. Natomiast dodanie na pozycję wymaga zastosowania iteracji, zatem ma złożoność O(n).

Usuwanie

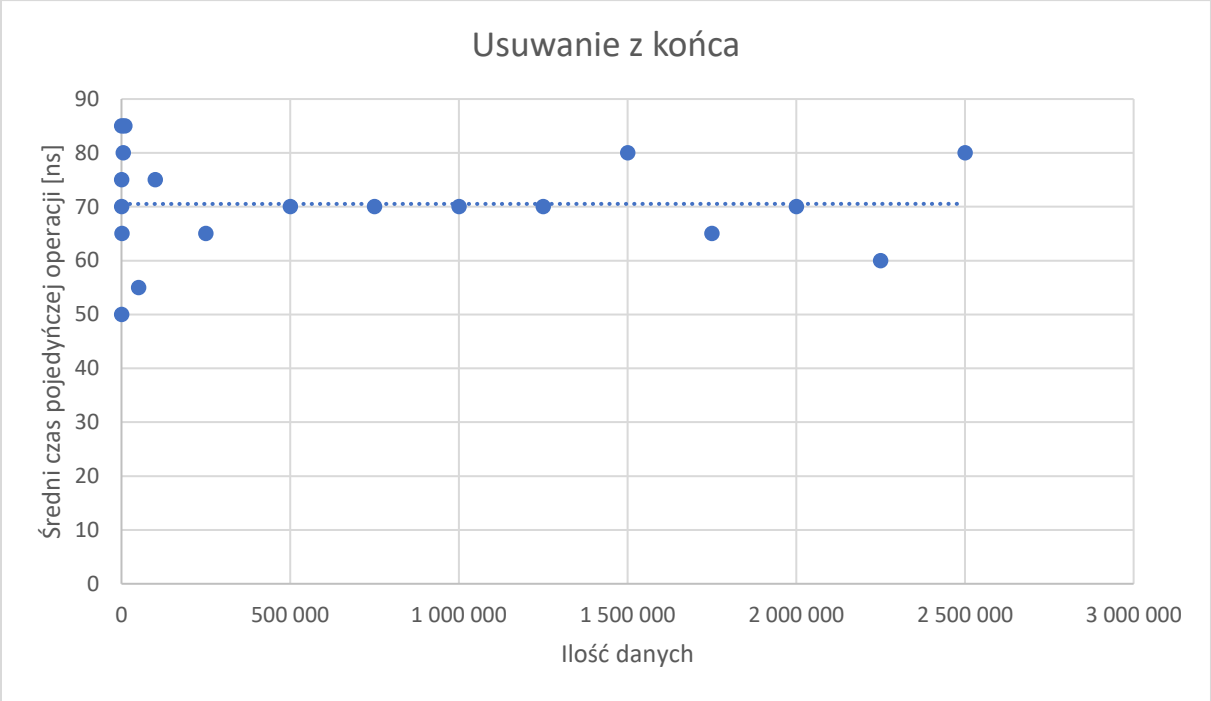
Z początku



Z wybranej pozycji



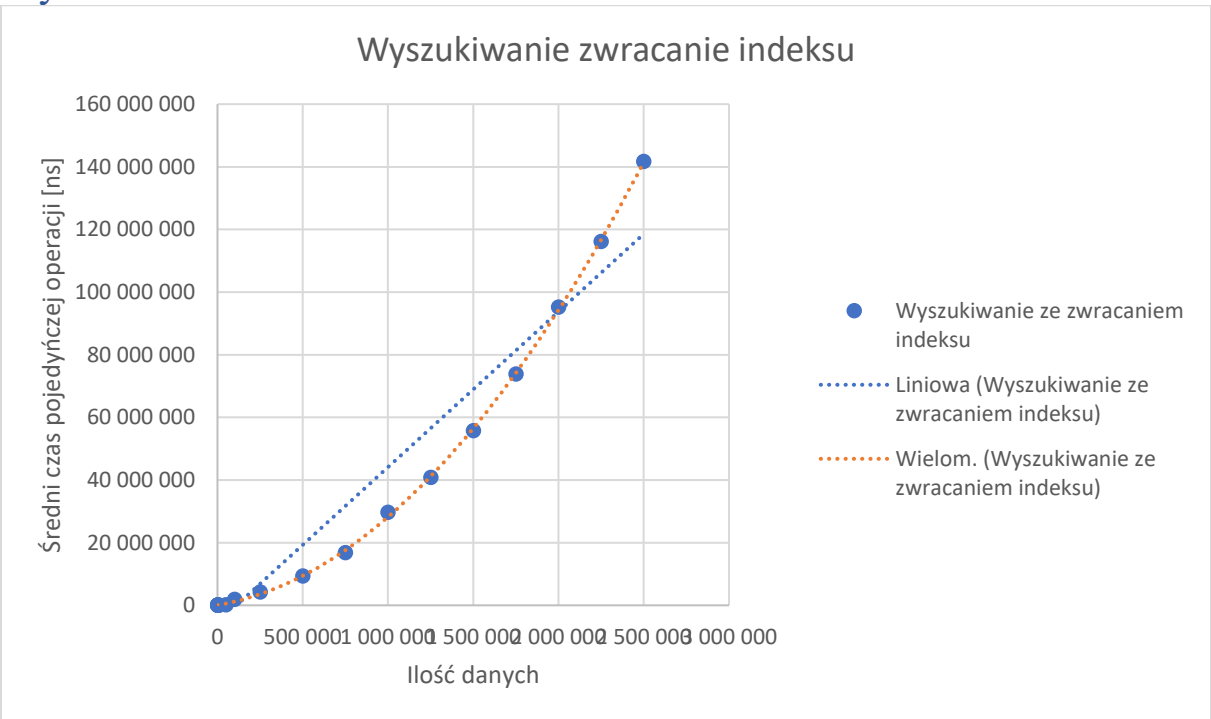
Z końca

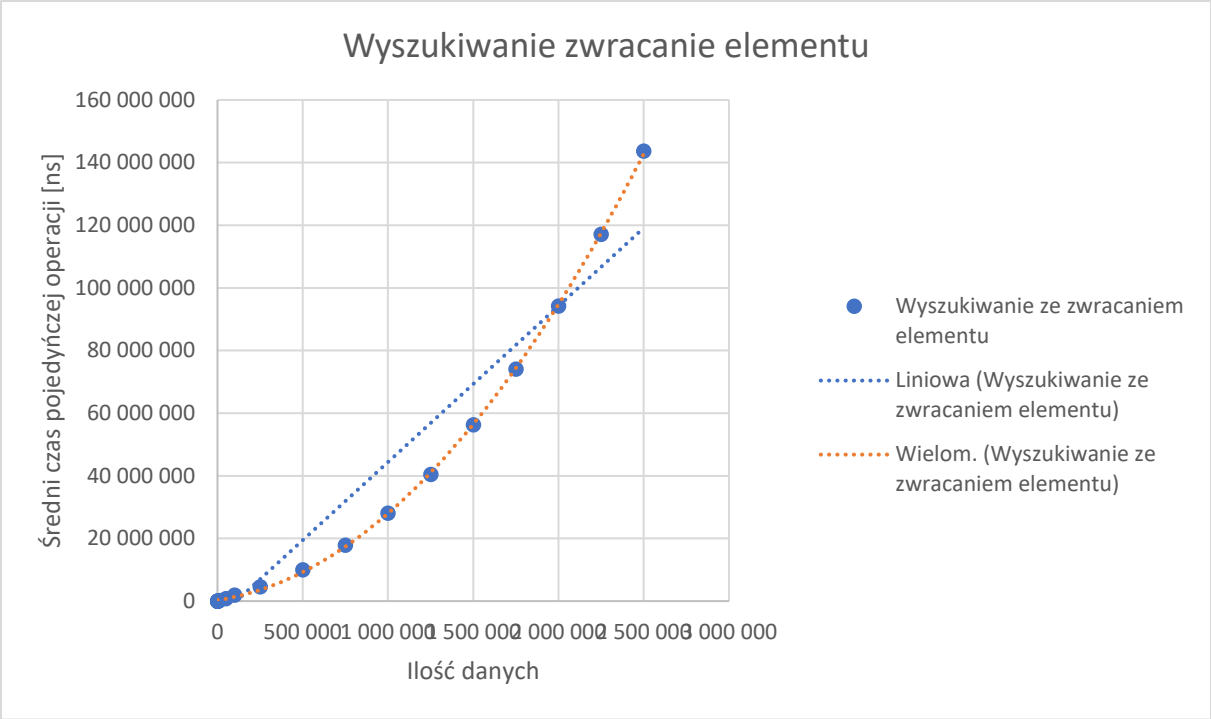


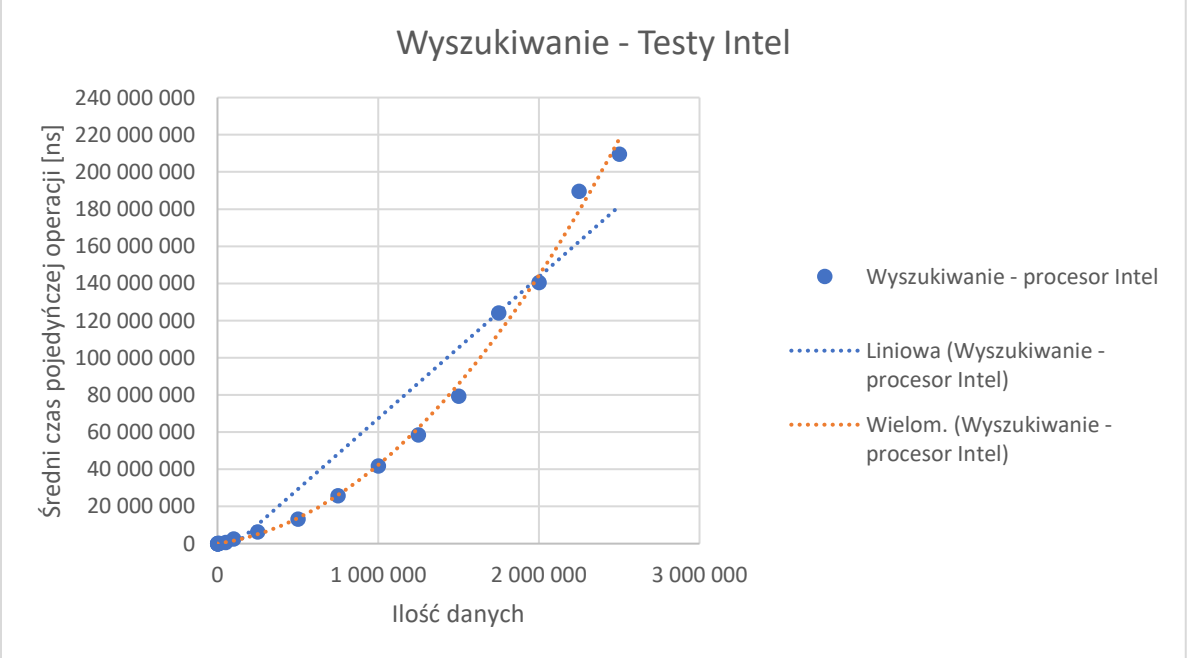
Podsumowanie

Podobnie jak w dodawaniu usuwanie na początek i koniec ma złożoność jednostkową O(1). Natomiast usuwanie z wybranej pozycji wymaga zastosowania iteracji, zatem ma złożoność O(n).

Wyszukiwanie







Podsumowanie

Zastosowany algorytm wyszukiwania iteracyjnego z założenia powinien wychodzić liniowy, jednakże testy wykazały, że jego złożoność wychodzi $O(n^2)$. W pierwszej kolejności myślałem, że wynika to z dodatkowej operacji, jaką jest liczenie w celu zwrócenia pozycji, jednak dodatkowe testy wykazały, że nie ma to związku. Wykonałem dodatkowe testy na komputerze z procesorem firmy Intel (pierwsze odbywały się na AMD), ale uzyskałem identyczny rezultat, a gorsze wyniki wynikały ze słabszej jednostki. Ostatecznie nie udało mi się znaleźć wytłumaczenia tej sytuacji.

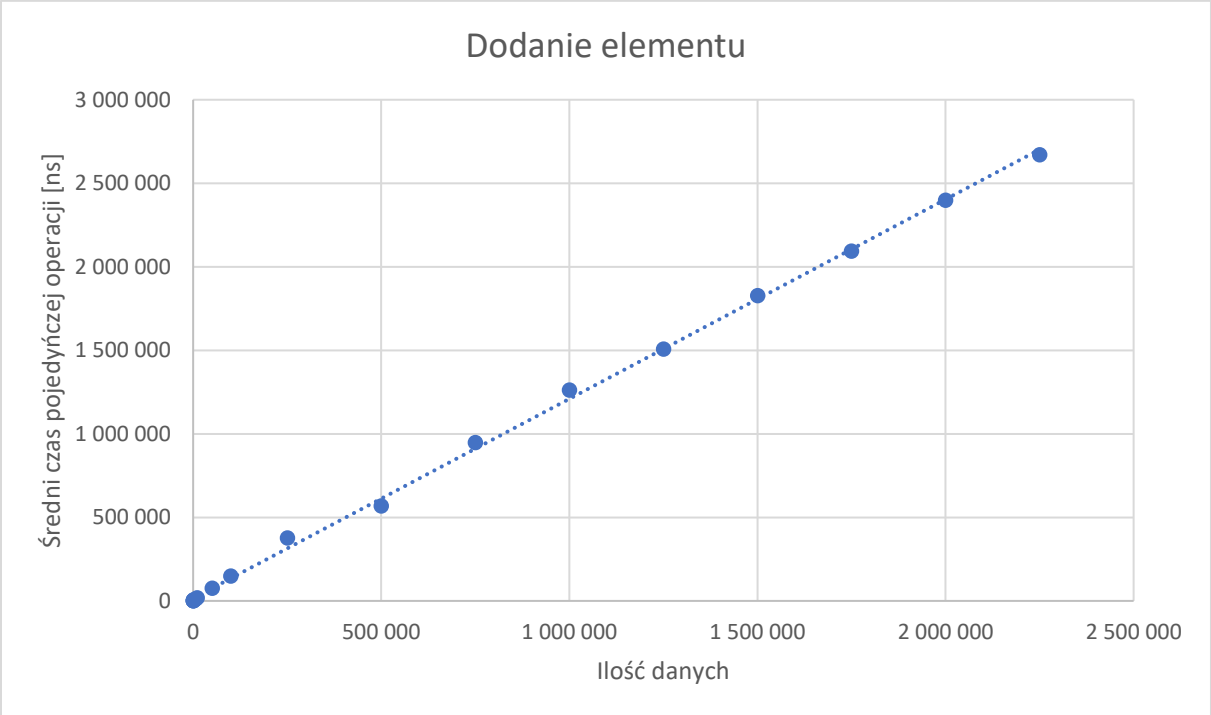
Kopiec binarny

Tabela czasów

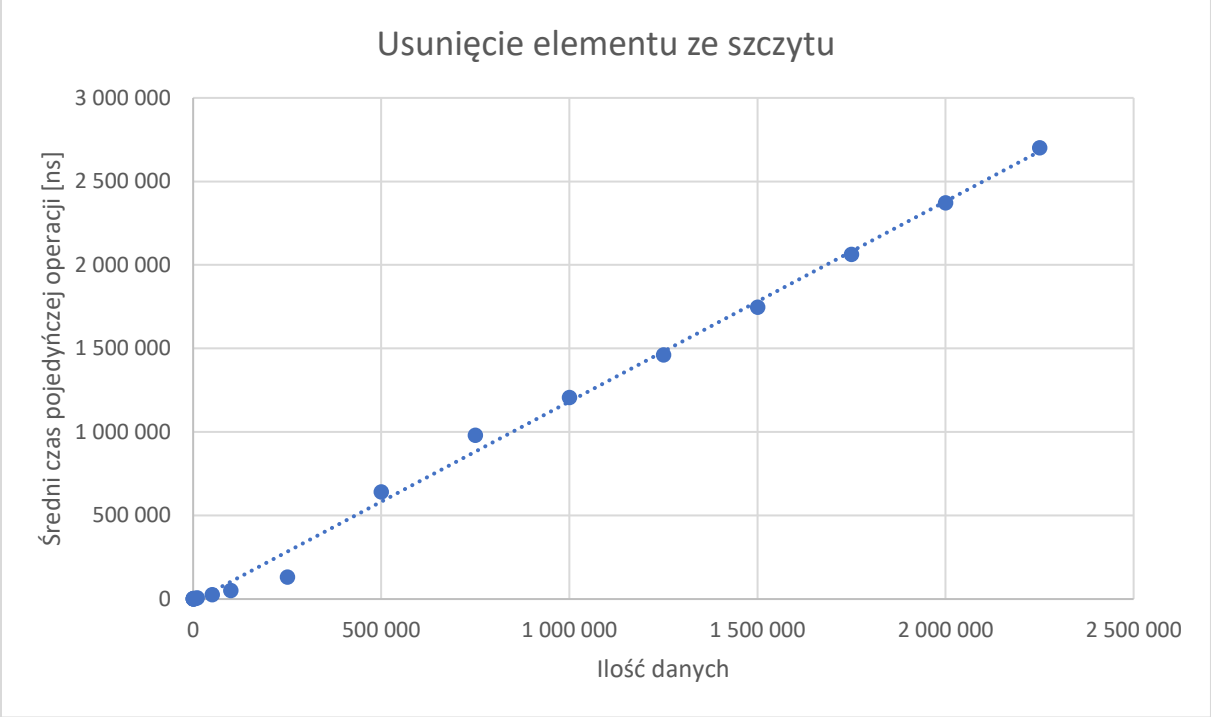
Ilość danych	Dodanie elementu	Usunięcie elementu ze szczytu	Wyszukiwanie	Algorytm Floyda	Kopcowanie
10	330	355	65	1 070	1 210
50	925	400	75	555	1 845
100	1 160	585	80	735	3 035
500	2 605	1 875	230	2 895	14 910
1 000	6 030	1 785	485	5 805	34 290
5 000	8 630	2 965	2 160	31 505	205 590
10 000	17 740	5 935	4 370	66 680	442 110
50 000	75 030	25 820	20 150	346 565	2 400 070
100 000	148 615	50 560	29 180	661 050	5 070 865
250 000	377 270	129 710	64 095	1 558 045	13 340 640
500 000	567 355	639 910	168 120	3 357 815	28 266 970
750 000	947 550	979 550	272 735	5 269 560	42 968 475
1 000 000	1 262 650	1 206 060	425 575	6 582 385	58 643 205
1 250 000	1 506 790	1 459 715	523 430	8 888 350	78 238 935
1 500 000	1 827 550	1 746 035	598 520	10 924 685	96 438 235
1 750 000	2 093 515	2 062 665	692 395	14 165 115	113 901 250
2 000 000	2 397 545	2 371 310	789 095	16 304 795	126 068 385
2 250 000	2 669 590	2 701 090	862 590	17 637 965	148 992 375

Podstawowe operacje

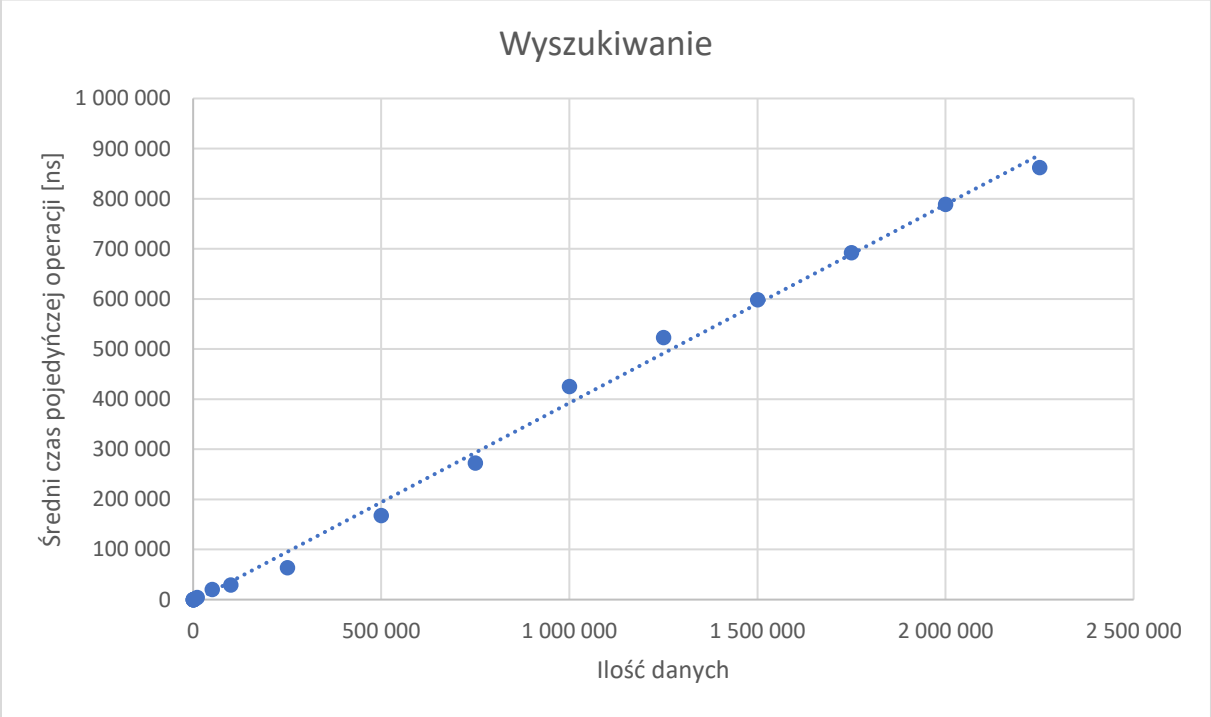
Dodawanie



Usuwanie elementu ze szczytu



Wyszukiwanie

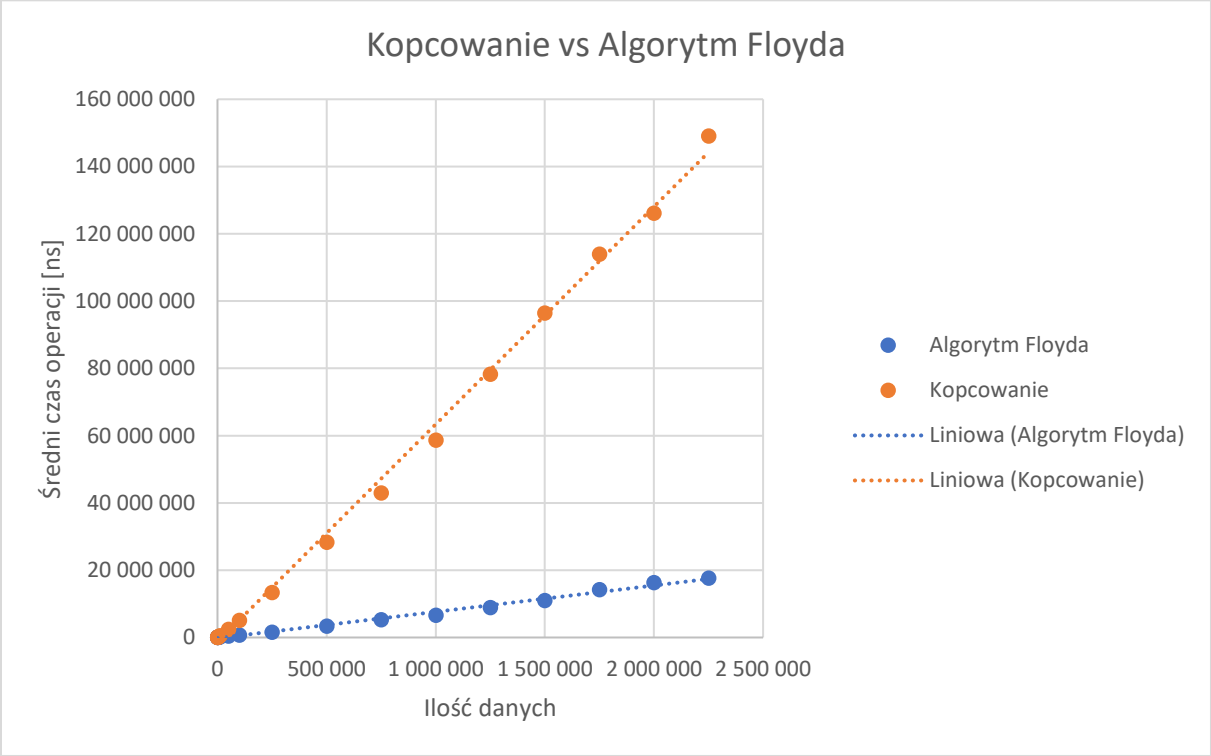


Podsumowanie

Wszystkie operacje na kopcu są liniowe, związane jest to z wykorzystaniem tablicy dynamicznej do przechowywania danych. Taki sposób realizacji kopca wymaga relokowania tablicy przy dodawaniu i usuwaniu danych. Wyszukiwanie odbywa się w sposób identyczny jak dla tablicy, jednakże z racji, że został zaimplementowany kopiec maksymalny to wyszukiwanie jest trochę szybsze,

ponieważ sprawdzamy na początku czy podana wartość jest większa od wartości na szczycie.

Budowanie kopca – porównanie algorytmów



Podsumowanie

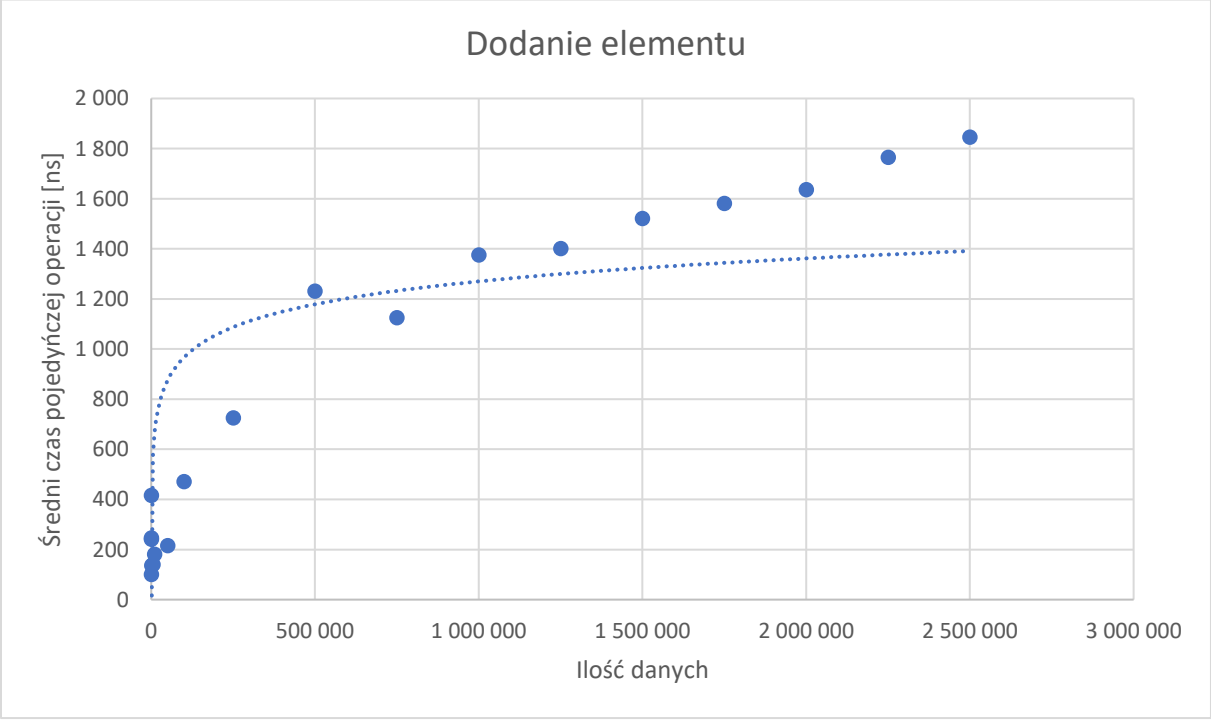
Analizując działanie algorytmów, łatwo zauważyć, że algorytm Floyda wykonuje ok. 2 razy mniej przebiegów pętli niż standardowy algorytm kopcowania. Złożoność standardowego algorytmu wynosi O(n log n) zatem można by się spodziewać, iż algorytm Floyda będzie miał taką samą złożoność. Jednakże jak możemy zauważyć na załączonym wykresie jest on praktycznie liniowy (z niewielkimi odchyłami od prostej), zatem możemy stwierdzić, że złożoność tego algorytmu wynosi O(n).

Drzewo czerwono-czarne

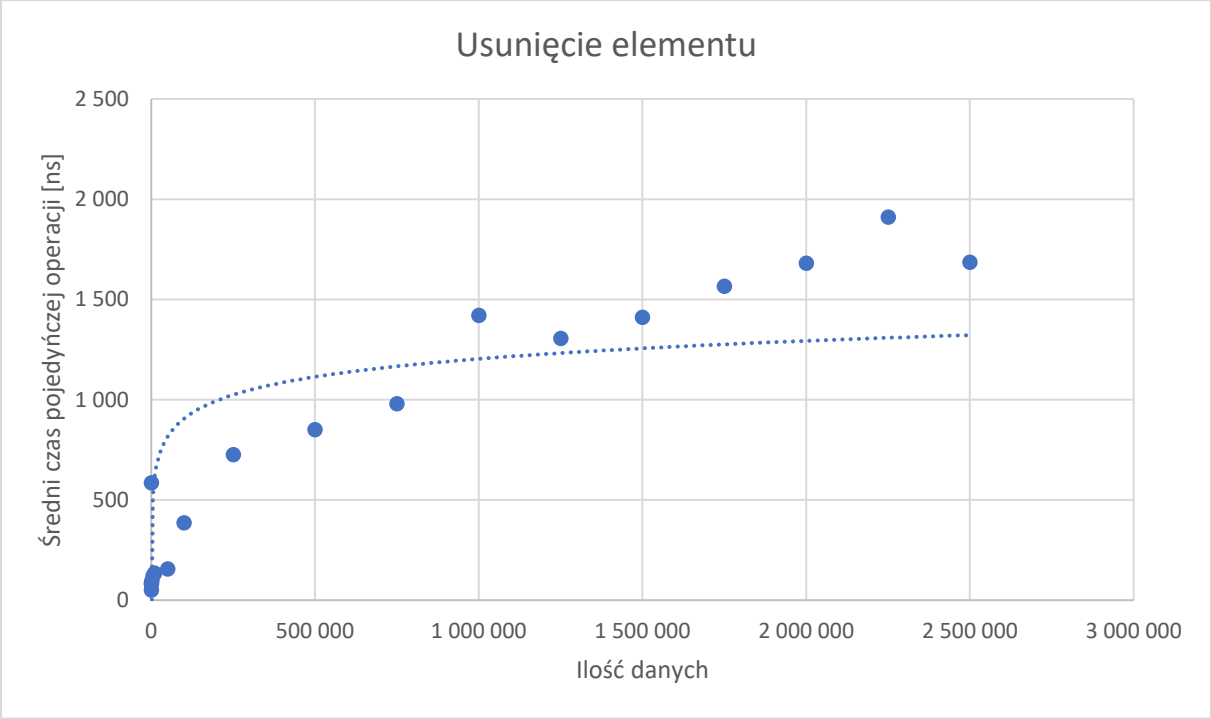
Tabela czasów

Ilość danych	Dodanie elementu	Usunięcie elementu	Wyszukiwanie
10	415	585	90
50	245	50	40
100	240	85	65
500	100	80	70
1 000	135	95	85
5 000	140	120	90
10 000	180	135	120
50 000	215	155	140
100 000	470	385	315
250 000	725	725	630
500 000	1 230	850	880
750 000	1 125	980	1 015
1 000 000	1 375	1 420	1 285
1 250 000	1 400	1 305	1 245
1 500 000	1 520	1 410	1 360
1 750 000	1 580	1 565	1 690
2 000 000	1 635	1 680	1 445
2 250 000	1 765	1 910	1 820
2 500 000	1 845	1 685	1 715

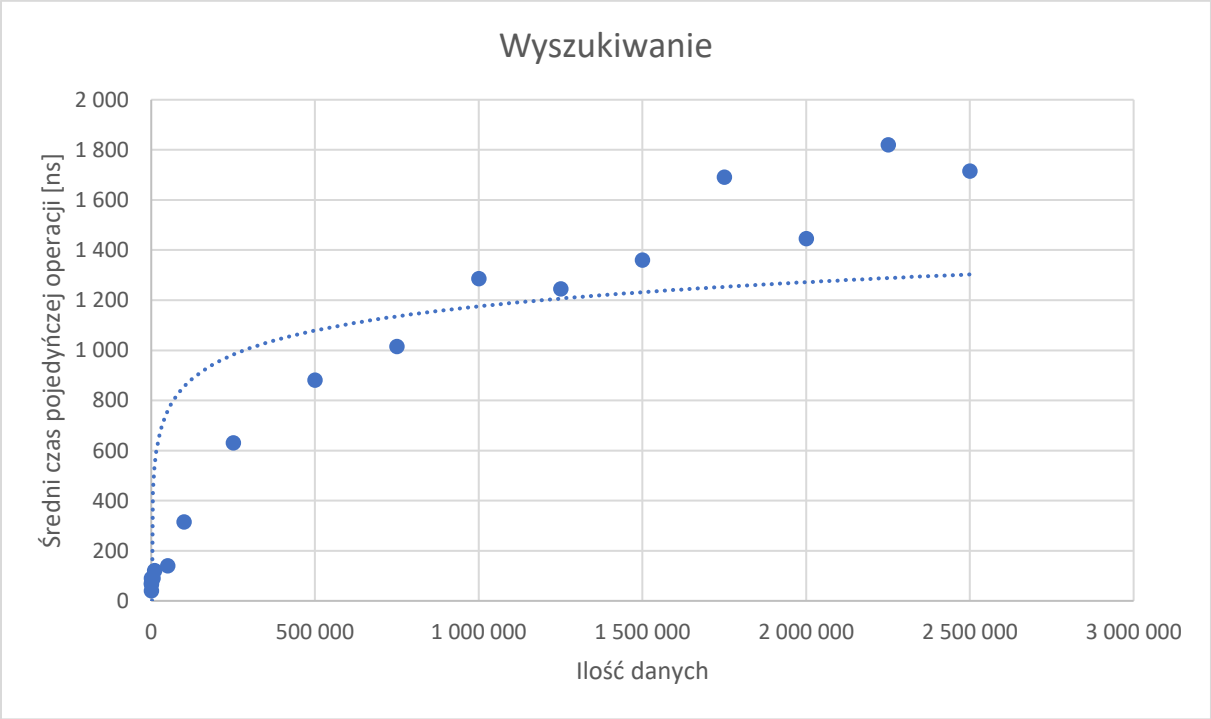
Dodawanie



Usuwanie elementu



Wyszukiwanie



Podsumowanie

Operacje na drzewie czerwono-czarnym mają złożoność O(log n), a ich czasy w porównaniu do innych struktur są znakomite. Jedynie dodawanie/usuwanie na początek/koniec listy daje nieznacznie lepsze czasy, ponieważ te operacje mają złożoność O(1). Drzewo czerwono-czarne wydaje się być najlepszą strukturą jeśli

zależy nam na szybkim wyszukiwaniu, oraz dobrym dodawaniu i usuwaniu wartości.

Bibliografia

1. „Wprowadzenie do algorytmów” Clifford Stein, Ron Rivest i Thomas H. Cormen
2. https://home.math.uni.lodz.pl/horzel/wp-content/uploads/sites/3/2020/03/3_drzewa_RB_splay.pdf
3. http://jaroslaw.mierzwa.staff.iiar.pwr.edu.pl/sdizo/
4. https://en.cppreference.com/w/cpp/chrono/parse
5. https://eduinf.waw.pl/inf/alg/001_search/0113.php
6. http://www.algorytm.org/