SPRAWOZDANIE Z 4 LABORATORIÓW Z PROGRAMOWANIA KOMPUTERÓW

Autor: Łukasz Wojczuk

Treść zadania:

Dla poniższych kontenerów:

- set,
- map,
- unordered_map,
- vector,
- queue,
- stack.

sporządź charakterystyki czasu wykonania wymienionych niżej operacji, w funkcji liczby elementów.

Charakterystyki powinny być przygotowane dla następujących operacji:

- dodawanie elementów do kolekcji,
- usuwanie elementów z kolekcji,
- wyszukiwanie elementu w kolekcji,

Wykonanie:

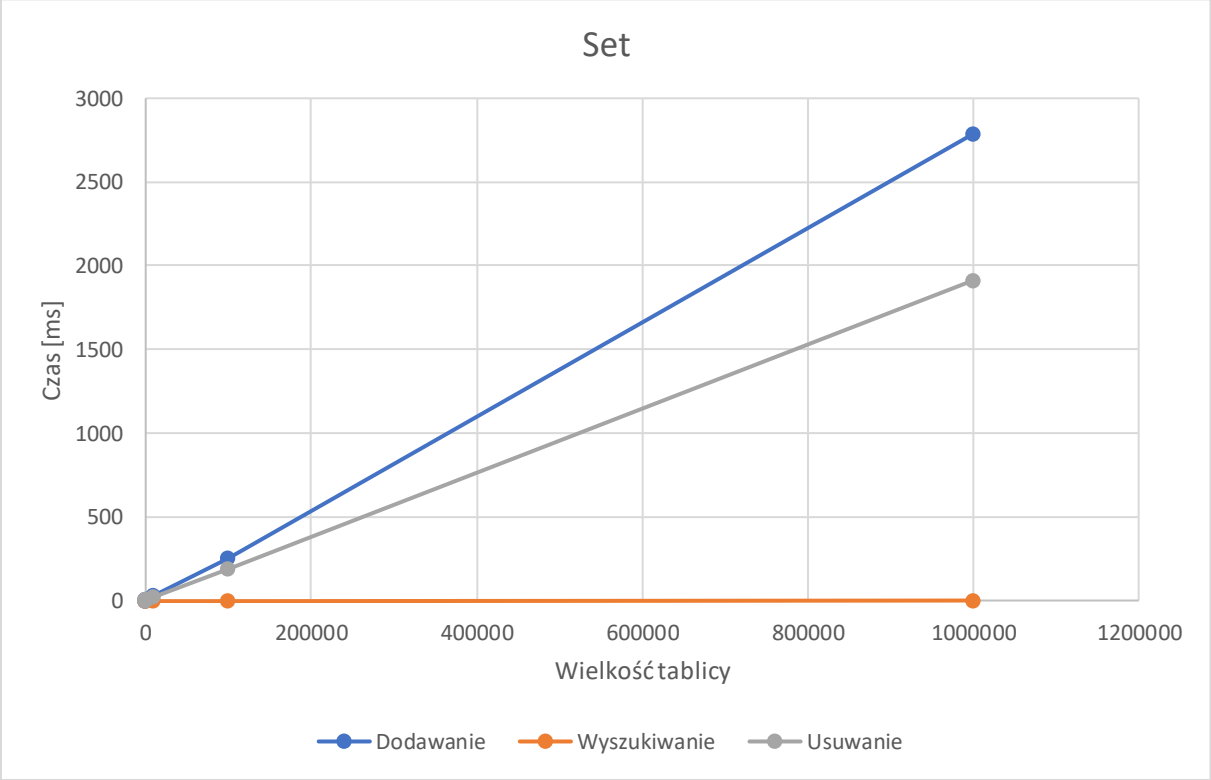
czynność\wielkość tablicy	1	10	100	1000	10000	100000	1000000
dodawanie set	0,0194	0,06	0,323	2,22	25,064	252	2787
wyszukiwanie set	0,0062	0,0256	0,0144	0,0162	0,0143	0,0131	0,005
usuwanie set	0,003	0,0294	0,216	1,793	18,11	189	1912
dodawanie map	0,0037	0,0147	0,173	2,521	31,476	253	1781
wyszukiwanie map	0,0012	0,0013	0,0017	0,0029	0,0029	0,0029	0,0049
usuwanie map	0,0027	0,0123	0,189	2,224	15,256	165	1344
dodawanie umap	0,0067	0,0251	0,201	2,305	17,653	187	1188
wyszkuwianie umap	0,0012	0,0009	0,0011	0,0013	0,0014	0,0024	0,0017
usuwanie umap	0,0015	0,0046	0,036	0,432	4,067	71,144	435
dodawanie vector	0,0041	0,0168	0,0882	0,775	12,177	926	46415
wyszukiwanie vector	0,0001	0,0002	0,0005	0,0036	0,0778	0,458	3,397
usuwanie vector	0,0016	0,0063	0,0466	1,185	9,278	751	nieskonczone
dodawanie queue	0,0034	0,007	0,0361	0,298	3,147	30,492	231
usuwanie queue	0,001	0,003	0,02	0,191	2,897	22,67	146
dodawanie stack	0,0028	0,0065	0,033	0,316	2,892	31,403	228
usuwanie stack	0,0007	0,0026	0,0183	0,309	1,918	19,812	138
							ms

Rys.1 Tabela z czasami wykonywania odpowiednich czynności

Kontener set:

```cpp
//set dodawanie

std::set<int> s;
sw.Start();
for (size_t i = 0; i < size; ++i) s.insert(i);
sw.Stop();
std::cout << "dodawanie setu: " << sw.ToString() << std::endl;

//set wyszukiwanie

sw.Start();
s.find(500'000);
sw.Stop();
std::cout << "wyszukiwanie setu: " << sw.ToString() << std::endl;

//set usuwanie
sw.Start();
for (size_t i = 0; i < size; ++i) s.erase(i);
sw.Stop();
std::cout << "usuwanie setu: " << sw.ToString() << std::endl;
```

Rys.2 Kod źródłowy kontenera set

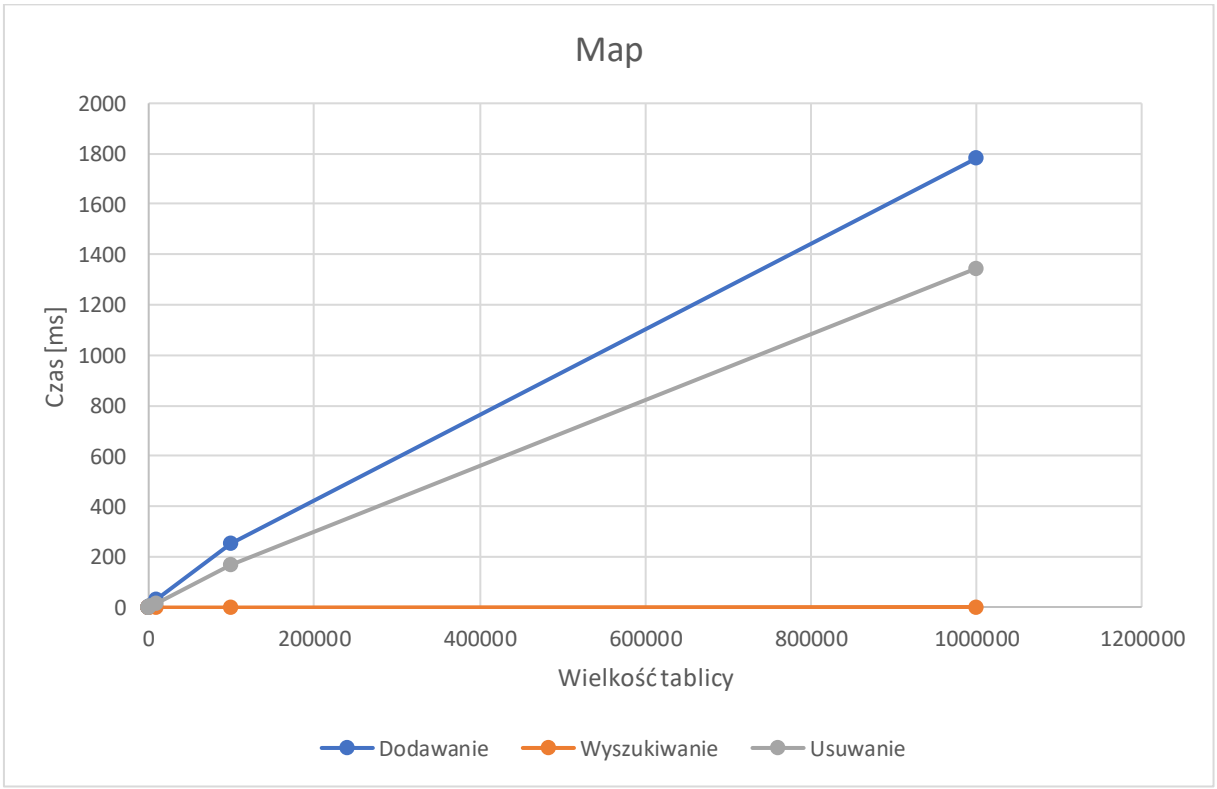


Rys.3 Wykres zależności czasu wykonywania czynności od wielkości tablicy dla kontenera set

Kontener map:

```cpp
//map dodawanie

std::map<int, int> m;
sw.Start();
for (size_t i = 0; i < size; ++i) m.insert(std::pair<int, int>(i, i));
sw.Stop();
std::cout << "dodawanie mapy: " << sw.ToString() << std::endl;


//map wyszukiwanie

sw.Start();
m.find(500000);
sw.Stop();
std::cout << "wyszukiwanie mapy: " << sw.ToString() << std::endl;

//map usuwanie

sw.Start();
for (size_t i = 0; i < size; ++i) m.erase(i);
sw.Stop();
std::cout << "usuwanie mapy: " << sw.ToString() << std::endl;
```

Rys.4 Kod źródłowy kontenera map

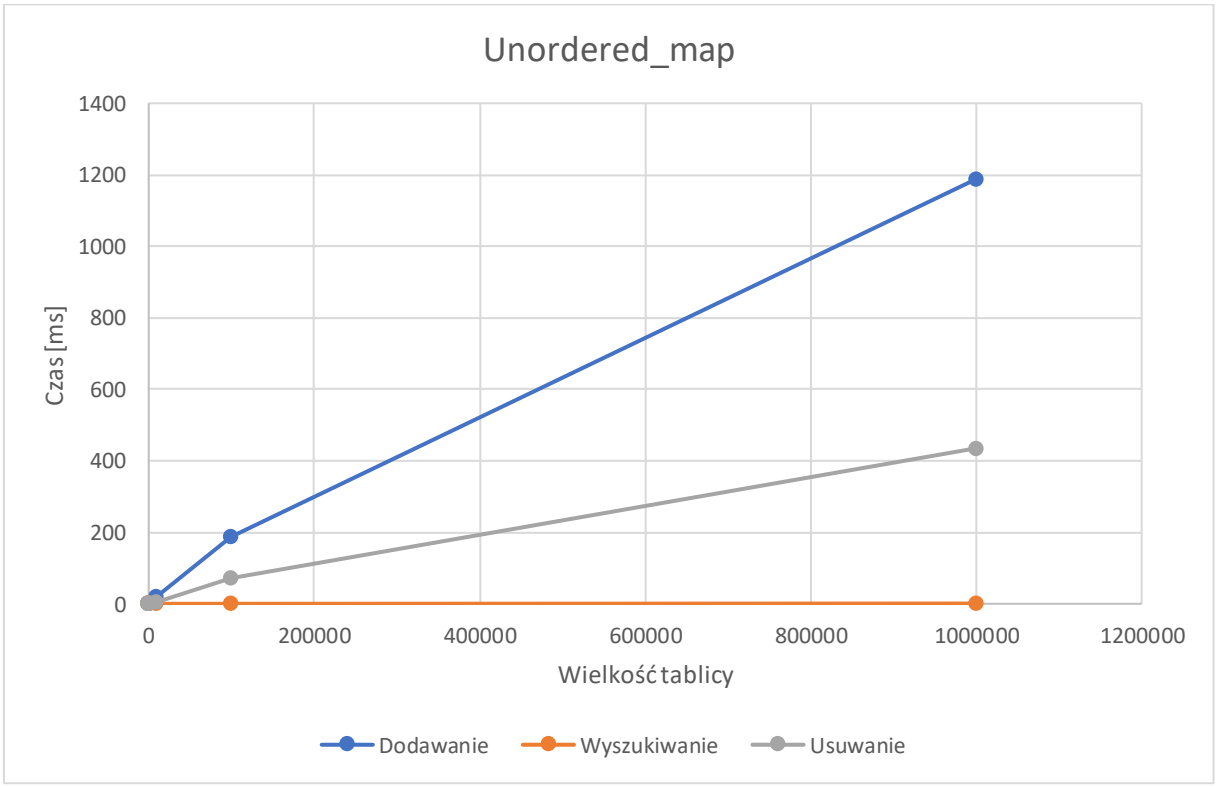


Rys.5 Wykres zależności czasu wykonywania czynności od wielkości tablicy dla kontenera map

Kontener unordered_map:

```cpp
// dodawanie unordered_map

std::unordered_map<int, int> um;
sw.Start();
for (size_t i = 0; i < size; ++i) um.insert(std::pair<int, int>(i, i));
sw.Stop();
std::cout << "dodawanie unordered_mapy: " << sw.ToString() << std::endl;

//unordered_map wyszukiwanie

sw.Start();
um.find(500000);
sw.Stop();
std::cout << "wyszukiwanie unordered_mapy: " << sw.ToString() << std::endl;

//unordered_map usuwanie

sw.Start();
for (size_t i = 0; i < size; ++i) um.erase(i);
sw.Stop();
std::cout << "usuwanie unordered_mapy: " << sw.ToString() << std::endl;
```

Rys.6 Kod źródłowy kontenera unordered_map

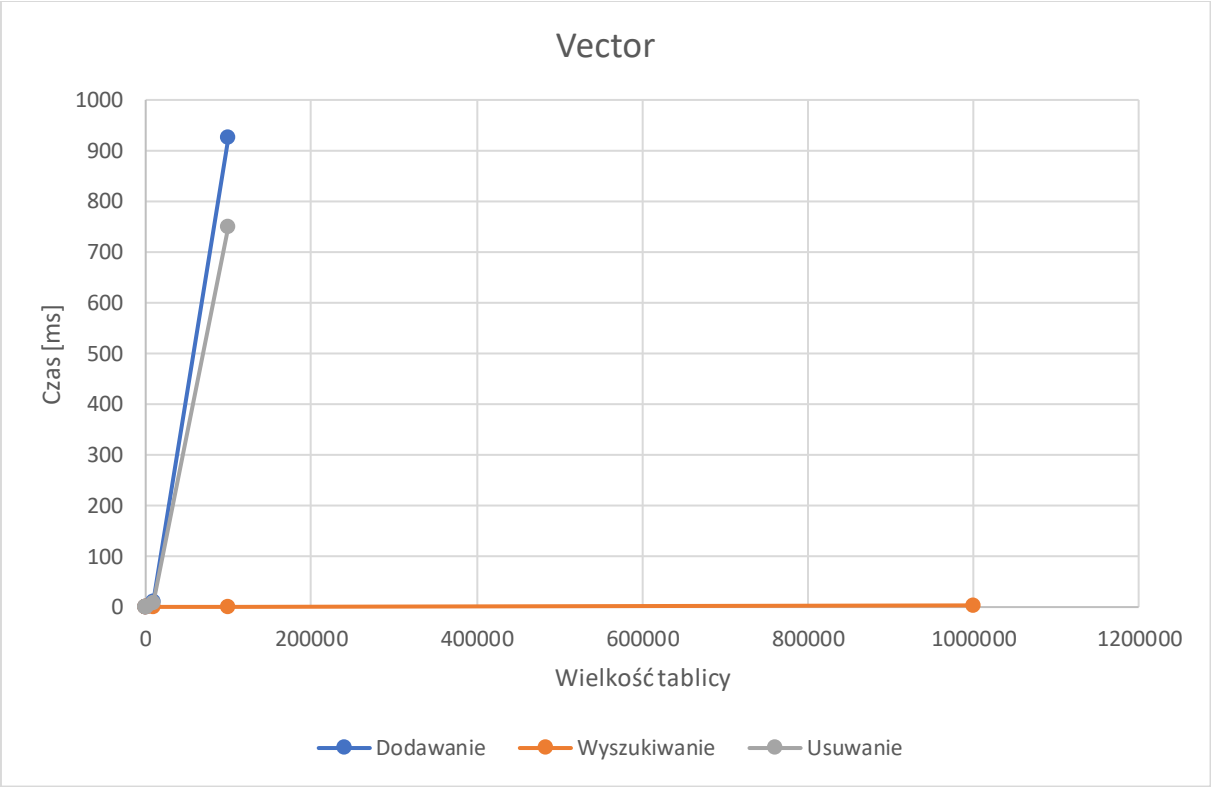


Rys.7 Wykres zależności czasu wykonywania czynności od wielkości tablicy dla kontenera unorderd_map

Kontener vector:

```
//dodawanie do wektora
sw.Start();
std::vector<int> v;
for (size_t i = 0; i < size; ++i) v.insert(v.begin(),i);
sw.Stop();
std::cout << "dodawanie wektora: " << sw.ToString() << std::endl;

//wyszukiwanie wektora

sw.Start();
for (size_t i = 0; i < size; ++i)
{
    if (v[i] == size / 2)
        break;
}
sw.Stop();
std::cout << "wyszukiwanie wektora: " << sw.ToString() << std::endl;

//usuwanie wektora
sw.Start();
for (size_t i = 0; i < size; ++i) v.erase(v.begin());
sw.Stop();
std::cout << "usuwanie wektora: " << sw.ToString() << std::endl;
```

Rys.8 Kod źródłowy kontenera vector

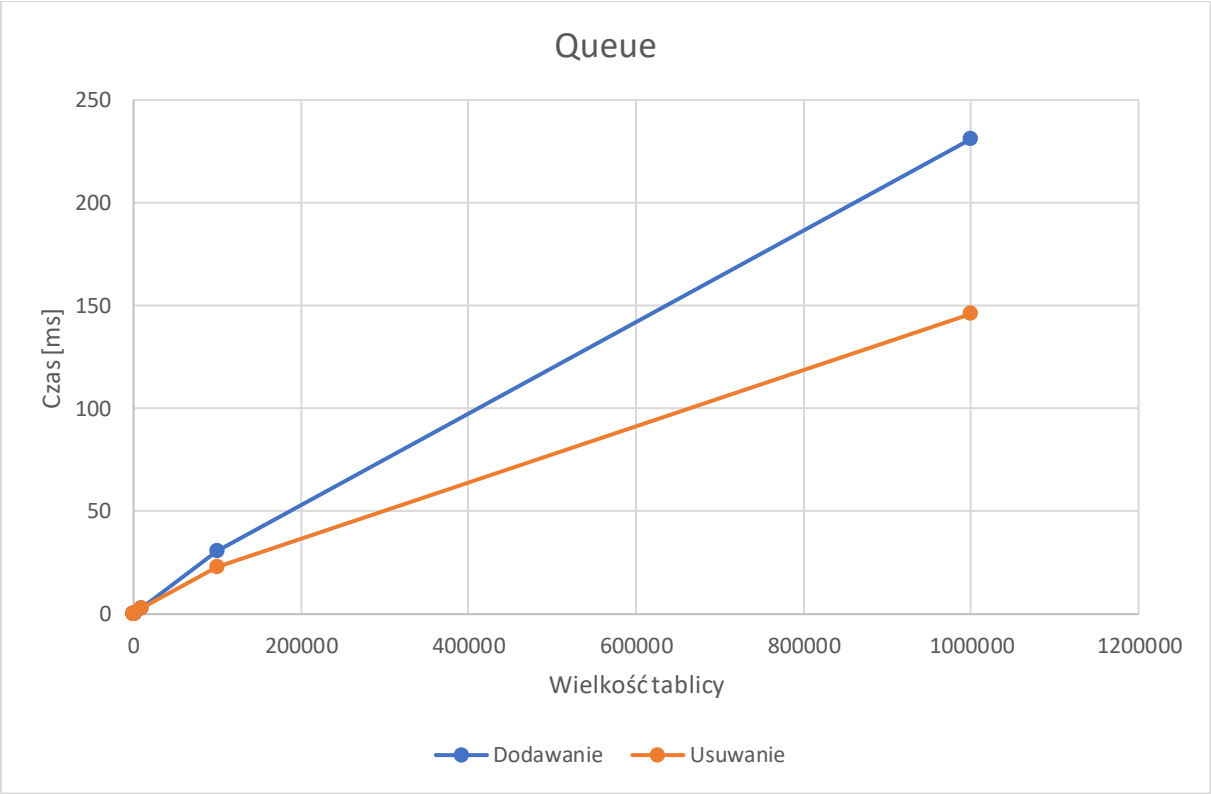


Rys.9 Wykres zależności czasu wykonywania czynności od wielkości tablicy dla kontenera vector

Kontener queue:

```
//dodawanie queue

std::queue<int> q;
sw.Start();
for (size_t i = 0; i < size; ++i) q.push(i);
sw.Stop();
std::cout << "dodawanie queue: " << sw.ToString() << std::endl;

//usuwanie queue

sw.Start();
for (size_t i = 0; i < size; ++i) q.pop();
sw.Stop();
std::cout << "usuwanie queue: " << sw.ToString() << std::endl;
```

Rys.10 Kod źródłowy kontenera queue

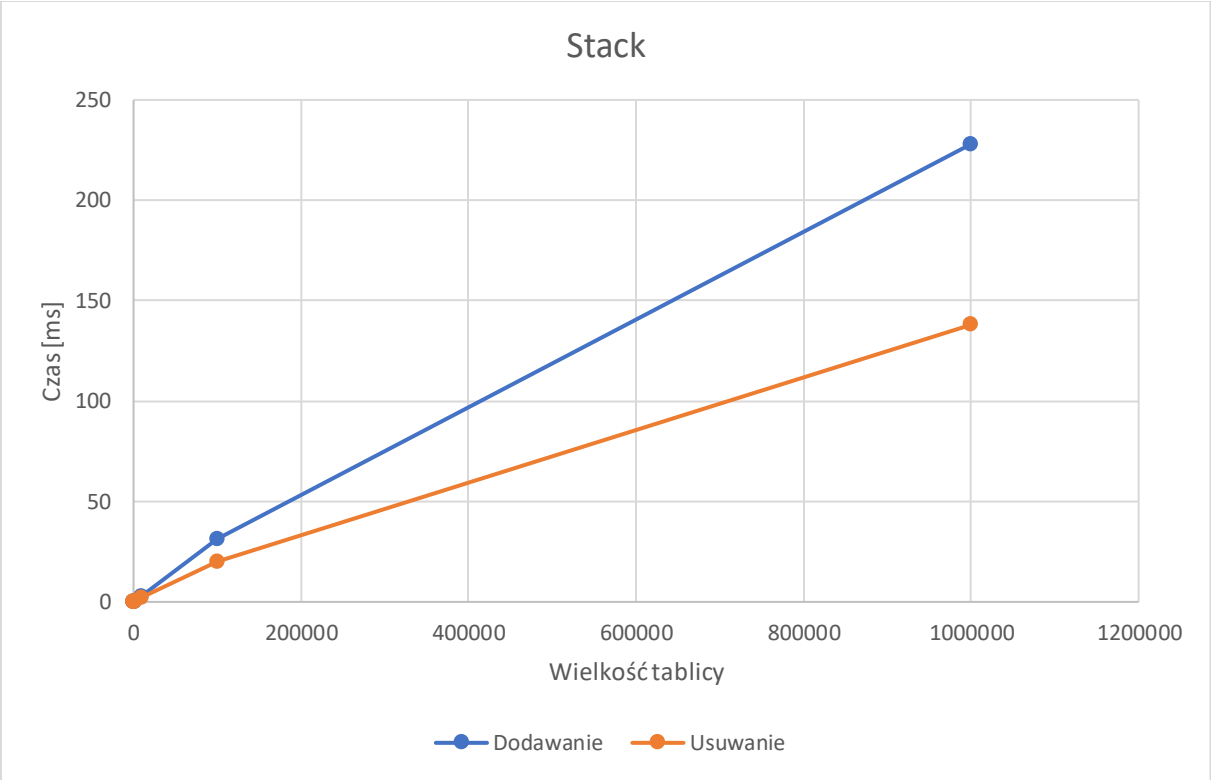


Rys.11 Wykres zależności czasu wykonywania czynności od wielkości tablicy dla kontenera queue

Kontener stack:

```
//dodawanie stack
std::stack<int> st;
sw.Start();
for (size_t i = 0; i < size; ++i) st.push(i);
sw.Stop();
std::cout << "dodawanie stack: " << sw.ToString() << std::endl;

//usuwanie stack

sw.Start();
for (size_t i = 0; i < size; ++i) st.pop();
sw.Stop();
std::cout << "usuwanie stack: " << sw.ToString() << std::endl;
```

Rys.12 Kod źródłowy kontenera stack



Rys.13 Wykres zależności czasu wykonywania czynności od wielkości tablicy dla kontenera stack

Wnioski:

Ze wszystkich przetestowanych kontenerów, najlepiej czasowo wychodzą kontenery stack i queue, z minimalną przewagą kontenera stack. Posiadają one jednak duży minus, nie mamy opcji wyszukiwania parametrów zapisanych w tych kontenerach. Spośród reszty kontenerów, które już posiadają opcję wyszukiwania najszybciej działają kontenery odpowiednio: unordered_map, map, set, vector. Czasy wszystkich kontenerów poza kontenerem vector są od 5 do 10 razy dłuższe od kontenerów stack i queue. Kontener vector przy użyciu funkcji insert oraz erase jest znacząco wolniejszy od pozostałych, dlatego na wykresie nie ma podanych danych dla największej testowanej tablicy. Dodawanie dla wielkości miliona danych w tablicy trwało ponad 46 sekund, przy czym dla pozostałych kontenerów najdłuższy czas wyniósł niecałe 3 sekundy. Czasu usuwania miliona danych z kontenera vector nie udało się ustalić, lecz najdłuższa próba wynosiła 3 godziny i mimo tak długiego czasu działania programu, usuwanie nie zostało zakończone. Jeżeli użytkownik nie ma potrzeby wyszukiwania danych z kontenerów, to najszybszą opcją będą kontenery stack i queue, lecz gdy użytkownik będzie potrzebował użycia funkcji wyszukiwania danych, to najrozsądniejszym wyborem wydaje się kontener unordered_map.