SDiZO TN, wt Marta Głowacka, 234999

## Struktury Danych i Złożoność Obliczeniowa

## ZADANIE PROJEKTOWE NR 1

**Badanie efektywności operacji na danych w podstawowych strukturach danych**

1. **Informacje wstępne o zadaniu projektowym**

Projekt polegał na zaimplementowaniu i zmierzeniu czasu wykonywania poszczególnych operacji na strukturach danych typu:

* Tablica
* Lista dwukierunkowa
* Kopiec binarny typu maksimum
* Drzewo czerwono-czarne.

Operacje, których czas należało zmierzyć to: dodawanie, usunięcie i wyszukanie elementu ze struktury. Dodatkowo dla tablic i list należało zrealizować dodawanie i usuwanie elementu na początku, końcu i dowolnym miejscu.

Należało przyjąć poniższe zasady:

* podstawowym elementem struktur jest 4 bajtowa liczba całkowita ze znakiem, czyli typ integer,
* wszystkie struktury danych powinny być alokowane dynamicznie (w przypadku tablic powinny zajmować możliwie jak najmniej miejsca tzn. powinny być relokowane przy dodawaniu/usuwaniu elementów),
* nie wolno korzystać z gotowych bibliotek np. STL, Boost lub innych
* dopuszczalnymi językami programowanie są języki kompilowane do kodu natywnego – mój projekt został napisany w C++ w wersji obiektowej
* Klasy struktur powinny być uniwersalne, tzn. funkcje operacji na danych nie powinny korzystać z bibliotek typu np. iostream
* do dokładnego pomiaru czasu skorzystałam z funkcji std::chrono::high\_resolution\_clock
* należy sprawdzić poprawność zbudowanej struktury: wczytywanie danych z pliku, wyświetlenie struktury na ekranie, menu konsolowe

1. **Wstęp teoretyczny**

Aby pokazać różne złożoności czasowe operacji, należy najpierw uświadomić sobie, czym jest złożoność obliczeniowa algorytmów. Przytoczmy najpierw definicję problemu :

Złożoność obliczeniowa algorytmu to koszt jego realizacji, czyli ilość zasobów komputera niezbędnych do jego wykonania.[[1]](#footnote-1)

Złożoność obliczeniową dzielimy na złożoność pamięciową oraz czasową algorytmów.

* **Złożoność czasowa** (ang. time complexity) to czas niezbędny do rozwiązania problemu. Wyrażana zazwyczaj jest w podstawowych jednostkach czasu (sekundy, milisekundy, mikrosekundy) bądź też w liczbie taktów procesora. Zależy od liczby danych wejściowych
* **Złożoność pamięciowa** (ang. space complexity) określa, ile pamięci zostanie zajętych przez dane w trakcie rozwiązywania problemu. Wyrażana jest przeważnie w bajtach. Złożoność ta również zależy od liczby danych wejściowych, na których wykonywane są operacje.

Istnieje również inna kategoria podziału złożoności obliczeniowej:

1. Złożoność optymistyczna, czyli zużycie zasobów dla najkorzystniejszego zbioru danych np. gdy wartość poszukiwana pojawia się na samym początku struktury.
2. Złożoność średnia, czyli typowe zużycie zasobów dla losowego zbioru danych. Wyznacza się ją, licząc średnią wielokrotnych pomiarów. Np. wartość poszukiwana pojawia się w środku struktury.
3. Złożoność pesymistyczna, zużycie zasobów dla najmniej korzystnego zbioru danych, np. gdy wartość poszukiwana pojawia się na samym końcu struktury.

Aby sprawnie poruszać się w terminologii złożoności czasowych operacji, należy jeszcze przybliżyć, czym jest klasa złożoności obliczeniowej. Określaja ona rząd funkcji T(n). Aby określić ten rząd, używamy notacji omikrona, czyli dużego O.

Funkcje f(n) i g(n) są funkcjami N->R. Są one czasami przebiegu dwóch algorytmów działających na danych wejściowych o rozmiarze n.

Powiemy, że f rośnie nie szybciej niż g. Oznaczamy to:

F(n) = O((g(n))

Jeżeli istnieją stałe c>0 i n0 ∈ N takie, że f(n) ≤ c \* g(n) dla wszystkich n≥n0[[2]](#footnote-2)

Istnieją również inne notacje np. Omega, Theta.

Najczęściej stosowanymi klasami złożoności obliczeniowej są:

O(1) - stała klasa czasowej złożoności obliczeniowej. Algorytm wykonuje stałą liczbę operacji bez względu na rozmiar danych n.

O(n) - liniowa klasa czasowej złożoności obliczeniowej. Algorytm wykonuje stałą liczbę operacji dla każdej danej n.

O(n2) - kwadratowa klasa czasowej złożoności obliczeniowej. Dla każdej danej n algorytm wykonuje proporcjonalną do n liczbę operacji.[[3]](#footnote-3)

Ο(log(n)) - złożoność logarytmiczna, czas rozwiązania zależy od wyniku logarytmu[3](http://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/#fn:podstawa) z wielkości danych wejściowych. [[4]](#footnote-4)

W literaturze często spotykamy się z podawaniem złożoności jedynie dla przypadku średniego (typowego) albo pesymistycznego.

Tabela dla niektórych operacji struktur danych i złożoność obliczeniowa:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Typ struktury danych**  **i algorytmu** | Przypadek najgorszy-N / Przypadek średni-Ś | | | | | |
| **wstawianie** | | **wyszukiwanie** | | | **wybór** |
| **N** | **Ś** | **N** | **Ś** | | **N** |
|  |  |  | **traf.** | **chyb.** |
| tablica uporządkowana, tablica nieuporządkowana z wyborem miejsca wstawiania | n | n/2 | n | n/2 | n/2 | 1 |
| tablica nieuporządkowana (np. stos) | 1 | 1 | n | n/2 | n | n lg n |
| lista uporządkowana | n | n/2 | n | n/2 | n/2 | n |
| lista nieuporządkowana (np.stos) | 1 | 1 | n | n/2 | n | n lg n |
| wyszukiwanie binarne (tablica) | n | n/2 | lg n | lg n | lg n | 1 |
| binarne drzewo poszukiwań | n | lg n | n | lg n | lg n | n |
| drzewo czerwono-czarne | lg n | lg n | lg n | lg n | lg n | lg n |

*wg R.Sedgewick: Algorytmy w C++*

Posiłkując się powyższą tabelą oraz informacjami z innych źródeł można przestawić tabele dla poszczególnych struktur danych dla przypadku pesymistycznego.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operacja** | **Tablica** | **Lista** | **Kopiec** | **Drzewo czerwono-czarne** |
| Dodanie wartości | O(n) | O(1) | O(1) | O(log(n)) |
| Usunięcie wartości | O(n) | O(n) | O(1) | O(log(n)) |
| Wyszukanie wartości | O(n) | O(n) | O(n) | O(log(n)) |
| Dostęp do elementu | O(1) | O(n) | O(n) | O(log(n)) |

1. **Plan eksperymentu**

Cały program został napisany w C++ obiektowo z zastosowaniem polimorfizmu i dziedziczenia. Dane do struktur wczytywane są z pliku, mogą być również do niego zapisane.

Pierwsza liczba w pliku struktura.txt określa rozmiar struktury (liczbę elementów, którą należy wczytać z pliku). Z tego uniwersalnego pliku mogą korzystać wszystkie struktury, gdyż każda bazuje na podobnym schemacie: wymaga prowadzenia liczby danych i samych danych: gdyż każda z nich zawiera pole size oraz kontener, który te wartości gromadzi (np. tablica w przypadku kopca lub węzły w przypadku drzew).

Do dokładnego pomiaru czasu skorzystałam z funkcji std::chrono::high\_resolution\_clock z biblioteki <chrono>. W tym celu utworzyłam klasę Watch, która włącza stoper, zatrzymuje go, a także zwraca czas, jaki upłynął między tymi akcjami.

1. **Struktury – tabele i wykresy**

Testy zostały przeprowadzone dla struktur zawierających:

* 100
* 500
* 1000
* 2000

elementów. Tam, gdzie miało to sens, niektóre operacje sprawdziłam również dla struktur, w którym wcześniej nie znajdował się żadnej element.

Jeden test dotyczył wyszukiwania w strukturach, w których wartości mieszczą się w zakresie od 1 do 100, drugi w zakresie od 1000 do 20 000 (dla tablic i list).

Pierwszy wykres przedstawiony jest dla wartości z pierwszego zakresu, drugi dla drugiego zakresu.

Program wyliczył i podał wartość średnią wyliczoną dla czasów ze 100 testów, generując przy każdym teście nowy zestaw danych.

Aby zwiększyć przejrzystość danych, w tabeli umieszczono wartości czasowe bez jednostek. Wszystkie czasy podane są w µs.

**Tablica**

* Dodanie wartości na początek

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Push front | | | | | | | | | |
| Zakres wartosci 1-100 | | | | | Zakres wartosci 1000-200000 | | | | |
| Rozmiar struktury | | | | | Rozmiar struktury | | | | |
| 0 | 100 | 500 | 1000 | 2000 | 0 | 100 | 500 | 1000 | 2000 |
| 4,04886 | 22,43993 | 85,69761 | 167,24 | 448,16 | 4,24 | 29,30 | 80,90 | 164,52 | 461,84 |

* Usuwanie wartości z początku

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pop front | | | | | | | | | |
| Zakres wartosci 1-100 | | | | | Zakres wartosci 1000-20000 | | | | |
| Rozmiar struktury | | | | | Rozmiar struktury | | | | |
| 0 | 100 | 500 | 1000 | 2000 | 0 | 100 | 500 | 1000 | 2000 |
| 18,27 | 85,24 | 18,27 | 85,24 | 18,27 | 85,24 | 18,27 | 85,24 | 18,27 | 85,24 |

* Wyszukiwanie

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Search | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 1,04 | 1,24 | 1,13 | 1,18 | 12,23 | 22,85 | 38,49 | 49,46 |

* Dodawanie na koniec

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Push back | | | | | | | | | |
| Zakres wartosci 1-100 | | | | | Zakres wartosci 1000-20000 | | | | |
| Rozmiar struktury | | | | | | | | | |
| 0 | 100 | 500 | 1000 | 2000 | 0 | 100 | 500 | 1000 | 2000 |
| 3,39 | 22,99 | 82,3 | 165,8 | 450,87 | 4,96 | 24,81 | 88,72 | 162,12 | 451,27 |

* Usuwanie z końca

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pop back | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury |  |  |  | Rozmiar struktury |  |  |  |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 20,69 | 97,79 | 181,71 | 497,60 | 23,58 | 103,55 | 181,76 | 480,49 |

* Dodawanie wybranego elementu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Push selected | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 0,04 | 0,08 | 0,04 | 0,21 | 0,03 | 0,04 | 0,08 | 0,08 |

* Usuwanie wybranego elementu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pop selected | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 0,04 | 0,03 | 0,05 | 0,08 | 0,04 | 0,04 | 0,06 | 0,27 |

**Lista**

* Dodanie wartości na początek

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Push front | | | | | | | | | |
| Zakres wartosci 1-100 | | | | | Zakres wartosci 1000-20000 | | | | |
| Rozmiar struktury | | | | | Rozmiar struktury | | | | |
| 0 | 100 | 500 | 1000 | 2000 | 0 | 100 | 500 | 1000 | 2000 |
| 2,07 | 0,65 | 0,76 | 0,75 | 0,72 | 1,96 | 0,65 | 0,77 | 0,68 | 0,78 |

* Usuwanie wartości z początku

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pop front | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 1,06 | 2,10 | 1,20 | 1,53 | 0,93 | 1,21 | 1,14 | 1,17 |

* Wyszukiwanie

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Search | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 1,97 | 1,12 | 1,21 | 1,10 | 2,02 | 1,34 | 1,11 | 0,70 |

* Dodawanie na koniec

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Push back | | | | | | | | | |
| Zakres wartosci 1-100 | | | | | Zakres wartosci 1000-20000 | | | | |
| Rozmiar struktury | | | | | | | | | |
| 0 | 100 | 500 | 1000 | 2000 | 0 | 100 | 500 | 1000 | 2000 |
| 0,61 | 0,67 | 0,94 | 0,86 | 1,1 | 1,04 | 0,76 | 0,87 | 0,93 | 1,02 |

* Usuwanie z końca

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pop back | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury |  |  |  | Rozmiar struktury |  |  |  |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 1,13 | 1,10 | 1,26 | 1,62 | 0,70 | 0,74 | 0,98 | 1,74 |

* Dodawanie wybranego elementu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Push selected | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 0,68 | 0,26 | 0,04 | 0,04 | 0,55 | 0,08 | 0,07 | 0,12 |

* Usuwanie wybranego elementu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pop selected | | | | | | | |
| Zakres wartosci 1-100 | | | | Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | | Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 | 100 | 500 | 1000 | 2000 |
| 0,68 | 0,26 | 0,04 | 0,04 | 1,08 | 0,16 | 0,09 | 0,23 |

**Kopiec**

* Dodanie wartości do kopca

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Push | | | | |
| Zakres wartosci 1-100 | | | | |
| Rozmiar struktury | | | | |
| 0 | 100 | 500 | 1000 | 2000 |
| 1,6 | 1,78 | 2,31 | 4,01 | 7,24 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Push | | | | |
| Zakres wartosci 1000-20000 | | | | |
| Rozmiar struktury | | | | |
| 0 | 100 | 500 | 1000 | 2000 |
| 1,53 | 1,81 | 2,76 | 4,35 | 7,3 |

* Usuwanie wartości z kopca

|  |  |  |  |
| --- | --- | --- | --- |
| Pop | | | |
| Zakres wartosci 1-100 | | | |
| Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 |
| 2,10 | 3,02 | 4,81 | 8,98 |

|  |  |  |  |
| --- | --- | --- | --- |
| Search | | | |
| Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 |
| 2,11 | 3,34 | 4,2 | 9,02 |

* Wyszukanie wartości w kopcu

|  |  |  |  |
| --- | --- | --- | --- |
| Search | | | |
| Zakres wartosci 1-100 | | | |
| Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 |
| 0,75 | 0,73 | 0,67 | 0,56 |

|  |  |  |  |
| --- | --- | --- | --- |
| Search | | | |
| Zakres wartosci 1000-20000 | | | |
| Rozmiar struktury | | | |
| 100 | 500 | 1000 | 2000 |
| 0,77 | 0,69 | 0,73 | 0,77 |

**Wnioski**

Czas wykonania danego algorytmu zależy od wielu czynników:

1. Jakości kodu napisanego przez programistę i wygenerowanego przez kompilator
2. Szybkości sprzętu, na którym jest on wykonywany
3. Rozmiaru danych wejściowych
4. Użytego algorytmu[[5]](#footnote-5)

W tabelach dla kopca (typu O(n)) przestawionych w sprawozdaniu można zauważyć wzrost czasu operacji wraz z rośnięciem wartości w strukturze. Jest to związane z budową definicyjną tej struktury. Jednak różnice są niewielkie i często obarczone błędami. Przyczyną tego może być zbyt mała odległość wartości 1-100 i 1000-20000 od siebie. Ta, okazuje się, mała różnica powodowała niewielkie wzrosty wartości czasu wyrażonego w mikrosekundach.

Gdy rozmiar struktury (dane początkowe) był większy i wykonywano na tej strukturze działanie o złożoności obliczeniowej O(n) – wykres był rosnący. Można wysnuć wniosek, że komputer musi zrobić więcej porównań, aby wstawić wartość w konkretne miejsce. Szczególnie widać to na wykresie funkcji push back, gdyż czas bezpośrednio zależy tu od początkowej wielkości struktury.

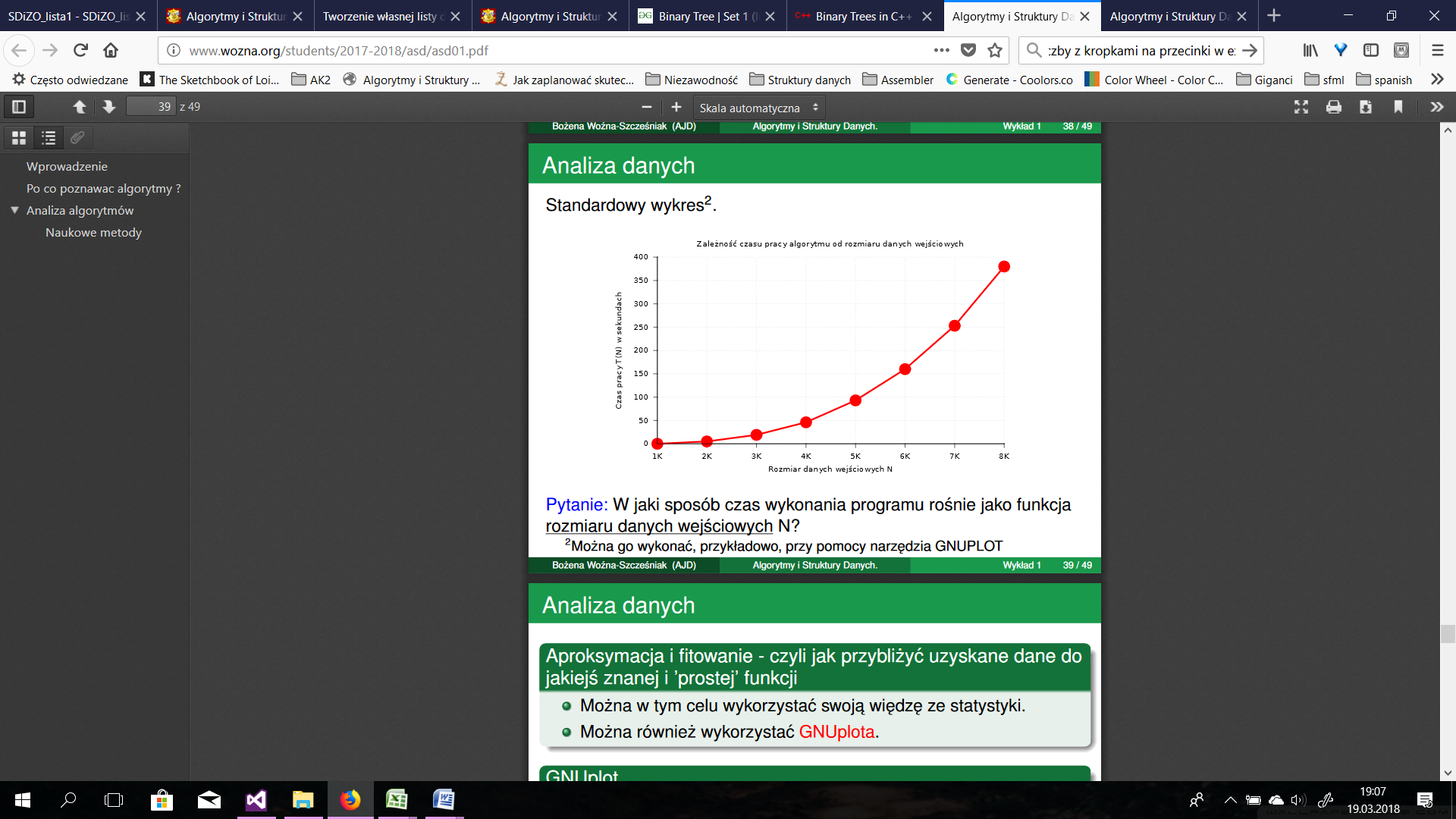
Powstrzymałabym się natomiast z wyciągania wniosków z wykresów wyszukiwania i wstawiania/usuwania elementu do/ze struktury. Za każdym razem (w każdej ze 100 prób) generowane były inne dane. Łatwo się domyślić, że szukana wartość pojawia się w losowych miejscach. Wielkość struktury ma tu więc mniejsze znaczenie – należałoby wykonać o wiele więcej testów na różnych wielkościowo strukturach.

Czas pomiaru był afektowany przez pamięć zajmowaną i zwalnianą w danym momencie działania programu. Pamięć była efektywnie zwalniana po każdym cząstkowym pomiarze, ale przy generowaniu struktur wielkości 2000 elementów z wartościami z przedziału 1000-20000 ma to znaczenie.

Aby efektywnie i z większą dokładnością wyznaczyć czas pojedynczej operacji, należałoby użyć poważniejszych narzędzi statystycznych, większej mocy obliczeniowej (by operacje na wielkich strukturach nie wykonywały się bardzo długo), a także przeprowadzić więcej prób. Należałoby również odjąć czas realokacji i alokacji elementów.

Wybór odpowiedniej struktury uzależniony jest od zastosowania. Jeśli potrzebujemy kontenera na elementy znajdujące się blisko siebie w pamięci oraz w kolejności – użyjemy tablicy (indeksowanie pętli). Jeśli potrzebujemy kontenera na dane, które często będziemy usuwać ze środka i dodawać w różne miejsca, jednocześnie nie zależy nam na kolejności – użyjemy listy (lista osób jadących na wycieczkę). Jeżeli zależy nam na dodatkowym posortowaniu wartości i zbudowaniu określonej hierarchii – wybierzemy kopiec (foldery na komputerze). Jeśli chcemy osiągnąć niską [złożonością obliczeniową](https://pl.wikipedia.org/wiki/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa) elementarnych operacji takich, jak wstawianie, wyszukiwanie czy usuwanie elementów z drzewa, wybierzemy drzewo binarne (czerwono-czarne).

Zaznaczając jeszcze raz, wszystkie jednostki czasu podane zostały w mikrosekundach, aby zwiększyć czytelność tabel (na wykresach zaznaczono jednostki).



<http://www.wozna.org/students/2017-2018/asd/asd01.pdf>

**Źródła**

Wykłady dr hab. Bożeny Woźnej-Szcześniak:

* <http://www.wozna.org/students/2017-2018/asd/asd01.pdf>
* <http://www.wozna.org/students/2017-2018/asd/asd02.pdf>

Wykład dr Tomasza Kapłona:

<http://tomasz.kaplon.staff.iiar.pwr.wroc.pl/epo/2017_2018/w2.pdf>

Wykład dr Jarosława Mierzwy:

jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/sdizo/zlozonosc\_lista\_kopiec.zip

Oraz:

<http://www.cs.otago.ac.nz/staffpriv/mike/Papers/MinMaxHeaps/MinMaxHeaps.pdf>

<https://arxiv.org/ftp/cs/papers/0007/0007043.pdf>

<https://www.cs.uaf.edu/2009/spring/cs311/slides/cs311_20090417_heap_pq_stl.pdf>

<http://eduinf.waw.pl/inf/alg/001_search/0113.php>

<https://pl.wikibooks.org/wiki/Struktury_danych/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa>

<http://eduinf.waw.pl/inf/utils/010_2010/0216.php>

http://home.agh.edu.pl/~byrska/IC/10ZlozonoscObliczeniowa.pdf

http://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/

1. http://www.wozna.org/students/2017-2018/asd/asd02.pdf [↑](#footnote-ref-1)
2. http://www.wozna.org/students/2017-2018/asd/asd02.pdf [↑](#footnote-ref-2)
3. http://eduinf.waw.pl/inf/utils/010\_2010/0216.php [↑](#footnote-ref-3)
4. http://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/ [↑](#footnote-ref-4)
5. <https://pl.wikibooks.org/wiki/Struktury_danych/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa> [↑](#footnote-ref-5)