|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Piotr Mikołajczyk | | | | |
| **AISDE**  **LAB 4** | 272018  Nr Indeksu | | 2021.05.04 – 2021.05.07  Data wykonania ćwiczenia | 2021.04.07  Nominalna data oddania sprawozdania |
| Punkty do wykonania | Zadanie polega na porównaniu dwóch implementacji kolejki priorytetowej: kopcowej i wykorzystującej selekcję, przy czym każda będzie występowała w dwóch wersjach: leniwej i nadgorliwej. Przypominamy, że w nadgorliwej kolejce priorytetowej operacje wstawienia i wyjęcia wiążą się z uporządkowaniem kolejki tak, by zminimalizować czas potrzebny na kolejne wyjęcie elementu o największym priorytecie. Natomiast w leniwej kolejce priorytetowej minimalizowany jest czas potrzebny na wstawienie nowego elementu, zaś odpowiednie operacje porządkujące są wykonywane dopiero na etapie wyjęcia elementu o największym priorytecie.  Należy zbadać przynajmniej jeden scenariusz użycia kolejki. Przykładowe scenariusze to:   1. Pewna liczba początkowych wstawień, po których wyjmowane są wszystkie elementy (lub tylko ich część). 2. Jak pkt 1, ale dla kluczy (tj. priorytetów) elementów z zakresu o wiele mniejszego od ich liczby (np. sto liczb jednocyfrowych). 3. Jak pkt 1, ale wstawiane są elementy o coraz większym (lub coraz mniejszym) priorytecie. 4. Sekwencja operacji, przy czym ich charakter (wstawienie/wyjęcie) jest losowy. Priorytety elementów mogą być przy tym losowe, rosnące lub malejące. W kodzie programu jest dostępna klasa RandomGenerator, której można użyć nie tylko do losowania wartości kluczy, lecz także do losowania binarnych flag, na podstawie których można wybrać rodzaj operacji: wstawienie lub wyjęcie. Domyślne prawdopodobieństwa zera i jedynki są równe, lecz łatwo można sprawić, by było inaczej. 5. Pewna ustalona liczba wyjęć z kolejki i/lub wstawień do niej. Początkowo kolejka zawiera pewną liczbę elementów, np. 10, 100, 1000 itd. Sprawdzamy, jak stan początkowy kolejki wpływa na liczbę koniecznych do przeprowadzenia operacji. UWAGA: operacji potrzebnych do wstępnego wypełnienia kolejki nie wliczamy do statystyk! 6. Kombinacja powyższych scenariuszy lub własna propozycja – zwłaszcza jeśli potrafimy uzasadnić, w jakiej sytuacji scenariusz taki mógłby wystąpić (np. kolejka pacjentów oczekujących na szczepienie). | | |

SPRAWOZDANIE:

Analiza właściwości struktur danych algorytmów

zastosowanych do implementacji kolejki priorytetowej

W ramach laboratorium 4 z przedmiotu AISDE, przeprowadzono badanie wpływu ułożenia wartości priorytetów w kolejce na szybkość działania algorytmów. W tym celu tworzono coraz większe kolejki ze stałą wartością, malejącą i rosnącą wraz z kolejnymi elementami kolejki, przy pomocy algorytmów kopcowej i wykorzystującą selekcje w wersjach leniwej i nadgorliwej.

Poniżej przedstawiono kod testowy dla której testowano każdy wariant kolejki.

*// Proste, "silowe" testowanie kolejki*

*void QueueTest(int x, int b,int IterNum,int flag) {*

*SimpleObject<int> a = SimpleObject<int>(x);*

*if (flag == 0) {*

*for (int i = 0; i < IterNum; i++) {*

*// Priorytet staly kazdego elementu*

*theQueue.put(a);*

*a.setValue(b); // wartosci priorytetow w kolejce - badanie dla roznych priorytetow*

*}*

*} else if (flag == 1) {*

*for (int i = 0; i < IterNum; i++) {*

*// priorytet rosncy dla kolejnych pozycji*

*theQueue.put(a);*

*a.setValue(b + i);*

*}*

*} else {*

*for (int i = 0; i < IterNum; i++) {*

*// priorytet malejcy dla kolejnych pozycji*

*theQueue.put(a);*

*a.setValue(b - i);*

*}*

*}*

*}*

*int main (int argc, char \* const argv[]) {*

*RandomGenerator gen(272018); // tu nalezy wstawic numer indeksu*

*std::string Comparisions;*

*std::string Copyings;*

*std::string TempCompare;*

*std::string TempCopying;*

*std::string DataInfo;*

*std::string CopyingsArray;*

*std::string ComparisonsArray;*

*int x = 1;*

*int b = 1;*

*int IterNum = 1;*

*int flag =2;*

*// Wybor implementacji kolejki:*

*for (int i = 0; i <= 14; i++){*

*QueueTest(x,b,IterNum,flag);*

*theQueue.printDataTable("po wczytaniu 1");*

*try {*

*std::cout << "get: " << theQueue.get().getValue() << "\n";*

*}*

*catch (QueueException &exc) {*

*std::cout << "get: empty\n";*

*}*

*theQueue.printDataTable("po get");*

*// std::cout << "Comparisons: " << SimpleObject<int>::getComparisons() << "\n";*

*// std::cout << "Copyings: " << SimpleObject<int>::getCopyings() << "\n";*

*TempCompare.append("Comparisons: ");*

*TempCompare.append(std::to\_string(SimpleObject<int>::getComparisons()));*

*TempCompare.append(" - Iter : ");*

*TempCompare.append(std::to\_string(i));*

*TempCompare.append("\n");*

*Comparisions.append(TempCompare);*

*TempCopying.append("Copyings: ");*

*TempCopying.append(std::to\_string(SimpleObject<int>::getCopyings()));*

*TempCopying.append(" - Iter : ");*

*TempCopying.append(std::to\_string(i));*

*TempCopying.append("\n");*

*Copyings.append(TempCopying);*

*TempCompare.clear();*

*TempCopying.clear();*

*DataInfo.append("\n");*

*DataInfo.append("Iter : ");*

*DataInfo.append(std::to\_string(i));*

*DataInfo.append(" Info: x: ");*

*DataInfo.append(std::to\_string(x));*

*DataInfo.append(" b: ");*

*DataInfo.append(std::to\_string(b));*

*DataInfo.append(" Iter Number: ");*

*DataInfo.append(std::to\_string(IterNum));*

*DataInfo.append(" Queue Data Size: ");*

*DataInfo.append(std::to\_string(theQueue.getDataSize()));*

*DataInfo.append("\n");*

*ComparisonsArray.append(std::to\_string(SimpleObject<int>::getComparisons()));*

*ComparisonsArray.append(" ");*

*CopyingsArray.append(std::to\_string(SimpleObject<int>::getCopyings()));*

*CopyingsArray.append(" ");*

*//DataInfo.append("\n");*

*x =1; //gen.getRandom(0,272018);*

*b =1; //gen.getRandom(0,272018);*

*IterNum = pow(2,i);*

*}*

*ComparisonsArray.append(" - Comparisions:");*

*ComparisonsArray.append("\n");*

*CopyingsArray.append(" - Copying");*

*CopyingsArray.append("\n");*

*std::cout << Comparisions;*

*std::cout << Copyings;*

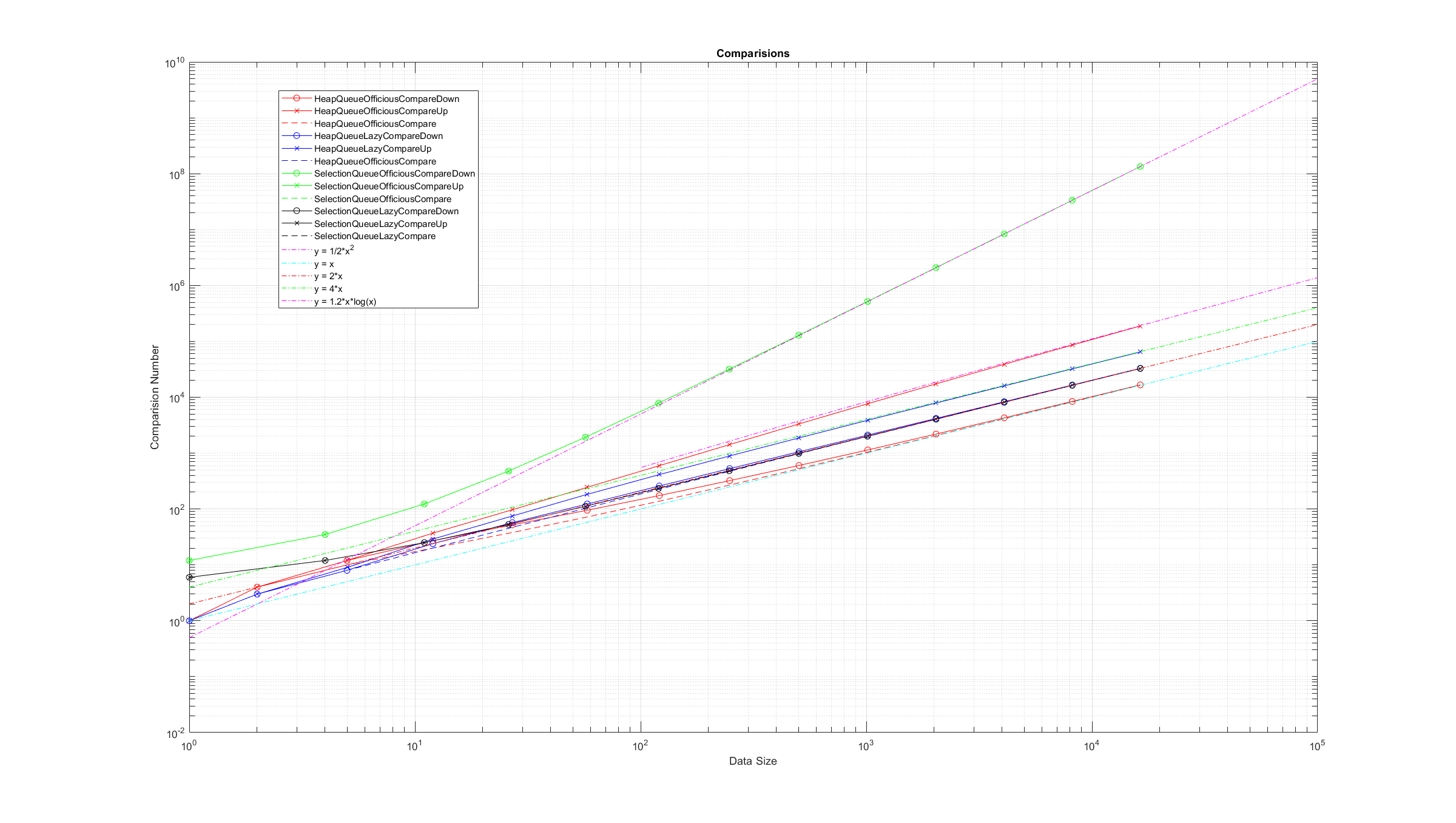
*std::cout << DataInfo;*

*std::cout << ComparisonsArray;*

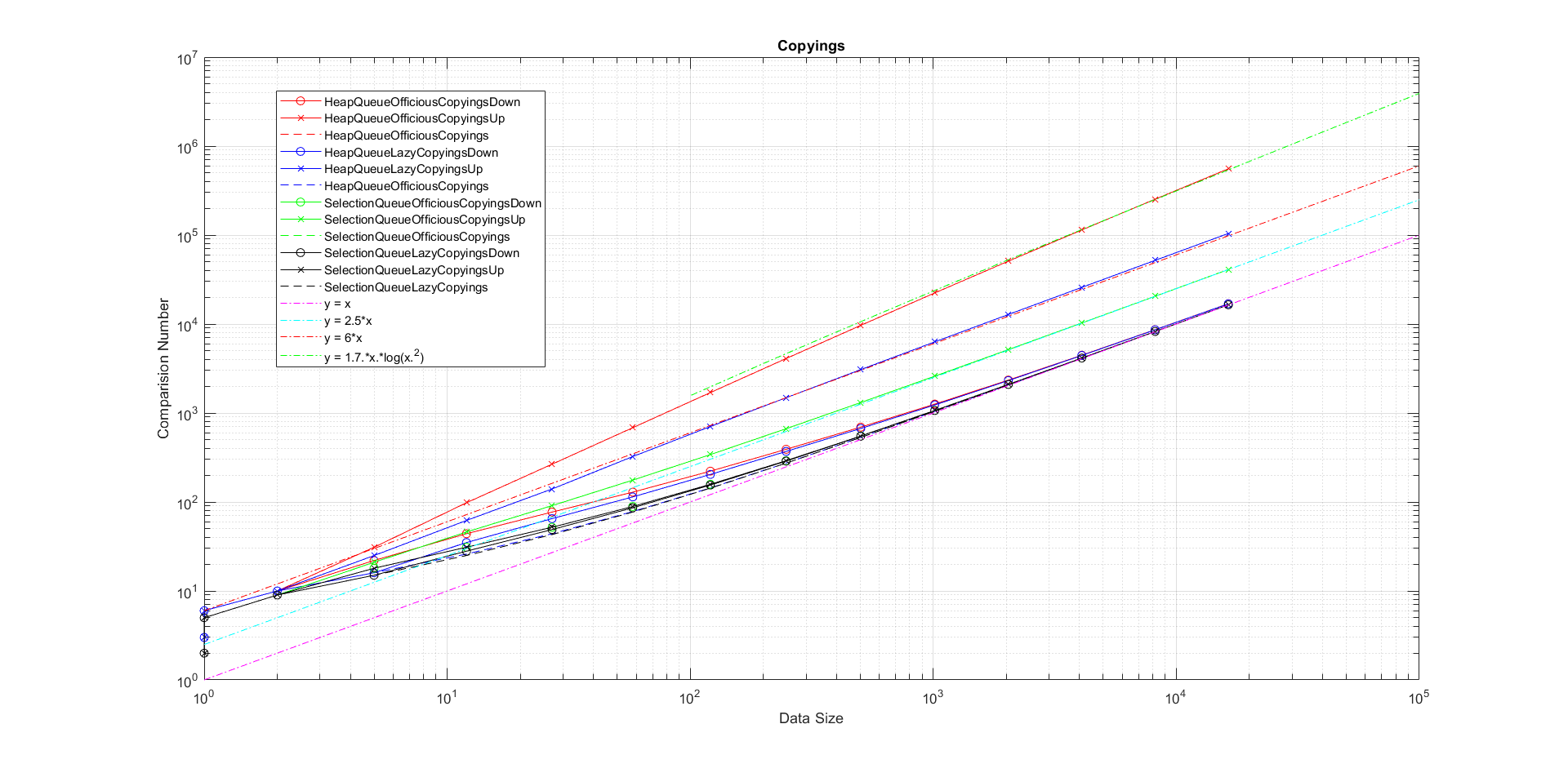
*std::cout << CopyingsArray;*

*return 0;*

*}*



*Rys. 1 – Złożoność obliczeniowa porównań algorytmów kolejki*



*Rys. 2 – Złożoność obliczeniowa kopiowania algorytmów kolejki*

Wnioski:

Zdecydowanie ze wszystkich algorytmów najwyżej wypadł algorytm kolejkowania przez selekcje

w wersji nadgorliwej. Na rysunku 1 liczba porównań nie zależnie od priorytetu elementów wchodzi w trend wykładniczy. Dzieje się tak ponieważ kolejka ta korzysta z zasady sortowania przez selekcje która ma złożoność kwadratową – nie nadaje się dla dużych zbiorów danych. Kolejka kopcowa w wersji nadgorliwej dla rosnących wartości priorytetu ma złożoność liniowo logarytmiczną. Natomiast reszta przypadków posiada złożoność liniową. Podwyższanie priorytetów dla kolejnych pozycji w kolejce skutkuje znaczącym podwyższaniem się złożoności obliczeniowych, natomiast obniżanie wartości priorytetów lub utrzymywanie stałej wartości dla wszystkich elementów, daje w przybliżeniu złożoność liniową lub liniowo logarytmiczną - wykluczając porównania dla kolejki przez selekcje w wersji nadgorliwej, która jest złym wyborem dla każdego wariantu nadawania elementom priorytetów.

*W sprawozdaniu należy zamieścić formułę Oświadczam, że niniejsza praca stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu Algorytmy i Struktury Danych została wykonana przez mnie samodzielnie.*