Imię Nazwisko: Kanstantsin Zhuk

Nr albumu: 250934

Nazwa kursu: Projektowanie algorytmów i metody sztucznej inteligencji

Prowadzący: Prof. Marcin Ochman

Zajęcia: wtorek 15:15

**Projekt nr 2: Algorytmy sortowania**

# 1. Wprowadzenie

## **1.1 Opis projektu**

Projekt polegał na przeanalizowaniu trzech algorytmów sortowania pod względem ich wydajności. W projekcie analizowane będą następujące algorytmy:

* MergeSort
* QuickSort
* IntroSort

W projekcie będą rozważane następujące objętości danych:

* 10 000
* 50 000
* 100 000
* 500 000
* 1 000 000

oraz następujące stopnie posortowania:

* Wszystkie elementy losowe
* 25% • 50% • 75% • 95%
* 99%
* 99.7%
* Wszystkie elementy posortowane, ale w odwrotnej kolejności

**1.2 Opis algorytmów sortowania**

# MergeSort

Algorytm stosujący metodę divide and conquer. Działanie MergeSort’a można opisać w dwóch prostych krokach:

* Dziel rekurencyjnie na połówki cały zestaw danych aż do osiągnięcia jednoelementowych podproblemów
* Scalaj z powrotem kolejne podproblemy wraz z ich sortowaniem aż do osiągnięcia posortowanego zestawu danych

Złożoność obliczeniowa MergeSort’a:

* Najlepszy przypadek → 𝒪(𝑛 log 𝑛)
* Typowy przypadek → 𝒪(𝑛 log 𝑛)
* Najgorszy przypadek → 𝒪(𝑛 log 𝑛)

Złożoność pamięciowa:

## • 𝒪(𝑛)

MergeSort jest algorytmem **stabilnym**, co oznacza, że jeśli przed sortowaniem mamy elementy o takiej samej wartości, które są względem siebie ułożone w konkretnej kolejności to po sortowaniu te elementy będą nadal w takiej samej kolejności.

Złożoność pamięciowa MergeSort’a wynika z potrzeby posiadania dodatkowej tymczasowej struktury danych. Jest to fakt który decyduje o tym, że HeapSort (algorytm sortujący „w miejscu”) jest preferowanym algorytmem nad MergeSort’em.

# QuickSort

Algorytm, który również stosujący metodę divide and conquer. Działanie QuickSort’a można opisać w dwóch prostych krokach:

• Wybierz na zasadzie określonych zasad element rozdzielający (pivot), który rozdziela zestaw danych na dwie części. Ważne jest, że elementy większe od pivota przenoszone są do części po jego prawej stronie a mniejsze do części po lewej.

* Wykonuj rekursywnie powyższą czynność dla kolejnych fragmentów zestawu danych aż do osiągnięcia jednoelementowego zbioru który jest uważany za posortowany.

Złożoność obliczeniowa QuickSort’a:

* Najlepszy przypadek → 𝒪(𝑛 log 𝑛)
* Typowy przypadek → 𝒪(𝑛 log 𝑛)
* Najgorszy przypadek → 𝒪(𝑛2)

Złożoność pamięciowa:

## • 𝒪(log 𝑛) | 𝒪(𝑛)

Quick nie potrzebuje dodatkowej struktury danych żeby sortować, tzn. sortuje on „w miejscu”. Złożoność pamięciowa 𝒪(𝑛) wynika z występowania pesymistycznego przypadku quicka. QuickSort jest **niestabilny**. Z uwagi na możliwość manipulacji wyboru pivota, algorytm ten może działać z bardzo rożną wydajnością.

# IntroSort

Algorytm ten jest hybrydowy, tzn. że składa się on z kilku innych algorytmów sortowania. Na IntroSort’a składają się: QuickSort, HepSort, InsertionSort. Działanie IntroSort’a można opisać w kilku prostych krokach:

* Partycjonuj dane w taki sam sposób jak QuickSort, czyli wykorzystując element rozdzielający.
* Jeżeli głębokość rekurencji (dozwolona głębokość wywołań rekurencyjnych określana jako współczynnik 2 ∗ log 𝑛) jest równa 0 przełącz się na HeapSort’a.
* Jeżeli pozostała ilość danych w określonym fragmencie jest mniejsza od 16 to należy przełączyć się na InsertionSort’a i posortować fragment do końca.

Złożoność obliczeniowa IntroSort’a:

* Najlepszy przypadek → 𝒪(𝑛 log 𝑛)
* Typowy przypadek → 𝒪(𝑛 log 𝑛)
* Najgorszy przypadek → 𝒪(𝑛 log 𝑛)

Złożoność pamięciowa:

## • 𝒪(log 𝑛)

Intro również nie potrzebuje dodatkowej struktury danych żeby sortować oraz jest **niestabilny**. Algorytm ten został stworzony aby wyeliminować najgorszy przypadek złożoności obliczeniowej Quick’a.

**1.3 Porównanie algorytmów**

# Merge & Quick

|  |  |  |
| --- | --- | --- |
|  | **Merge** | **Quick** |
| **Złożoność pamięciowa** | 𝒪(𝑛) | 𝒪(log 𝑛)| 𝒪(𝑛) |
| **Najgorszy przypadek złożoności czasowej** | 𝒪(𝑛 log 𝑛) | 𝒪(𝑛2) |
| **Lepsza wydajność** | równa dla każdej wielkości zestawu | mniejsze zestawy danych |

Przyjmuje się, że z uwagi na dobrą kompatybilność z pamięcią podręczną QuickSort jest szybszy niż Merge. Kompatybilność ta wynika z faktu że Merge korzysta z dodatkowej struktury danych.

# Merge & Intro

|  |  |  |
| --- | --- | --- |
|  | **Merge** | **Intro** |
| **Złożoność pamięciowa** | 𝒪(𝑛) | 𝒪(log 𝑛) |
| **Najgorszy przypadek złożoności czasowej** | 𝒪(𝑛 log 𝑛) | 𝒪(𝑛 log 𝑛) |
| **Lepsza wydajność** | równa dla każdej wielkości zestawu | dobra dla każdej wielkości zestawu |

# Quick & Intro

|  |  |  |
| --- | --- | --- |
|  | **Quick** | **Intro** |
| **Złożoność pamięciowa** | 𝒪(log 𝑛)| 𝒪(𝑛) | 𝒪(log 𝑛) |
| **Najgorszy przypadek złożoności czasowej** | 𝒪(𝑛2) | 𝒪(𝑛 log 𝑛) |
| **Lepsza wydajność** | mniejsze zestawy danych | dobra dla każdej wielkości zestawu |

IntroSort jest szybszy od QuickSort’a ponieważ eliminuje jego najgorszy przypadek oraz przyspiesza dla małych zestawów danych dzięki użyciu InsertionSort’a.

## **1.4 Przewidywane wyniki**

Analizując algorytmy można stwierdzić że najszybszy powinien być IntroSort a najwolniejszy MergeSort. Jednak w przypadku dużych ilości danych, MergeSort powinien się lepiej sprawdzić od QuickSorta; również dla większego stopnia wstępnego posortowania Merge może się zachowywać lepiej niż Quick z uwagi na dużą głębokość rekurencji. Intro zaś powinien być najlepszym ze wszystkich algorytmów.

## **2. Przebieg testów**

Podczas testu zostały wzięte średnie wyniki z trzech prób każdego sortowania:

* 25%

* 50%

* 75%

* 95%

* 99%

* 99.7%

* Dane posortowane, ale w odwrotnej kolejności

## **3. Podsumowanie i wnioski**

MergeSort zwykle znajduje się pomiędzy QuickSort’em a IntroSort’em; dzieje się tak w przypadku kiedy dane nie wymagają za dużo porównań, tj. kiedy dane są już w jakimś stopniu posortowane. Dla sortowań 99,7% MergeSort został najwolniejszym.

Z pierwszych sortowań i sortowania danych w odwrotnej kolejności wynika, że Quick jest wolniejszy od Intro. W podpunktach 95% i 99% widać odstępstwo od reguły, być może jest to spowodowane sposobem implementacji QuickSort’a i IntroSort’a, albo błędem pomiarowym.

Przewidywania wyników na podstawie założeń teoretycznych okazały się mniej lub więcej odzwierciedlać realne wyniki testów.