PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI

Patrycja Polek
Kiełbasa Kamil

23 marca 2018

Spis treści

1 Wstęp

Celem naszego projektu była implementacja algorytmów sortowanie: QuickSortu, HeapSortu oraz MergeSortu, a także porównanie ze sobą ich złożoności obliczeniowej w różnych przypadkach ułożeń sortowanych elementów. Na podstawie tej wiedzy napisaliśmy algorytm hybrydowy,który stwierdzał jaki rodzaj sortowania zastosować, by czas działania był jak najkrótszy.

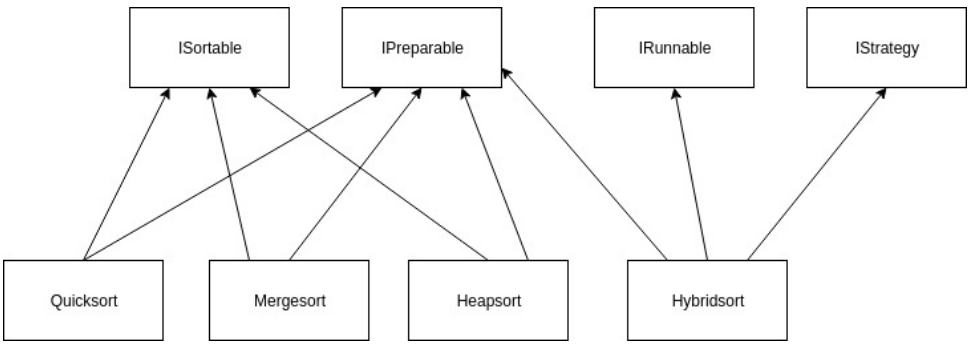
2 Kod źródłowy

2.1 Schemat powiązań klas

```
template <class T>
class Hybridsort : public IRunnable, public IStrategy, public IPreparable {
public:
    explicit Hybridsort();
    ~Hybridsort();

    // virtual methods
    void run();
    void clear() {}
    void strategy();
    void prepare(int data_size);
    void prepare(int data[], int data_size) {}

    // methods
    std::list<ISortable*>& getSortableList() { return list_sortable; }
    std::list<IPreparable*>& getPreparableList() { return list_preparable; }
    void print() const;

private:
    std::list<ISortable*> list_sortable;
    std::list<IPreparable*> list_preparable;
    Quicksort<T> *q;
    Mergesort<T> *m;
    Heapsort<T>  *h;

    T data_size;
    T *tab;

    T test_data_size;
    T *test_tab;

    std::vector<double> vec_time;
    std::vector<std::string> vec_string;
    std::string the_fastest;
};
```

(a) Hybridsort

Rysunek 1: Diagram

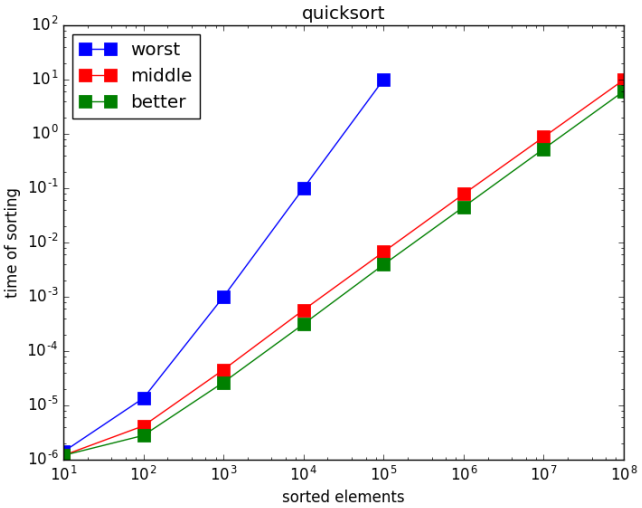


(a) Hybridsort

Rysunek 2: Diagram

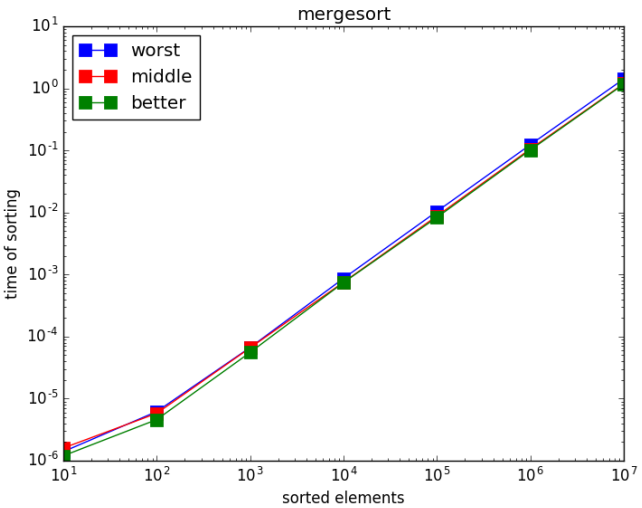
3 Wykresy

3.1 Wybrane przypadki dla danych algorytmów sortowania



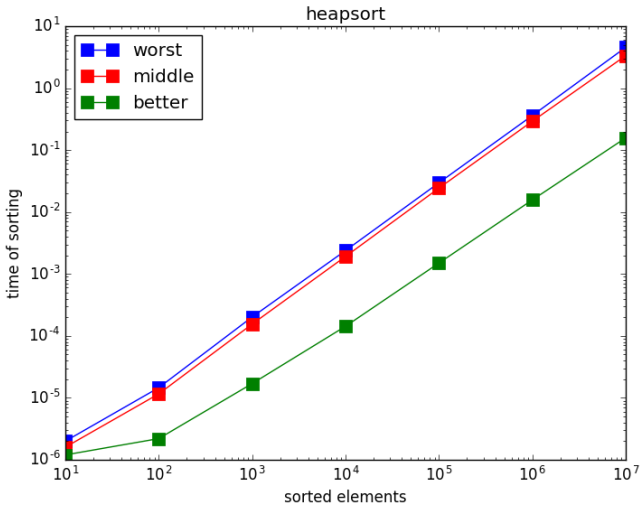
(a) Quicksort

Rysunek 3: Przypadki dla algorytmu sortowania Quicksort



(a) Mergesort

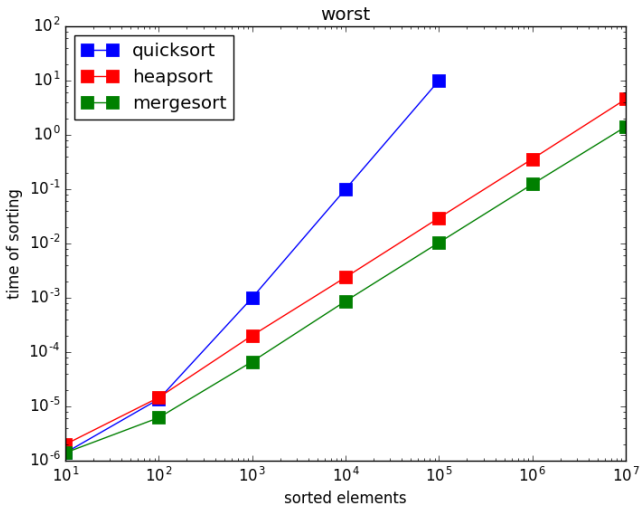
Rysunek 4: Przypadki dla algorytmu sortowania Mergesort



(a) Heapsort

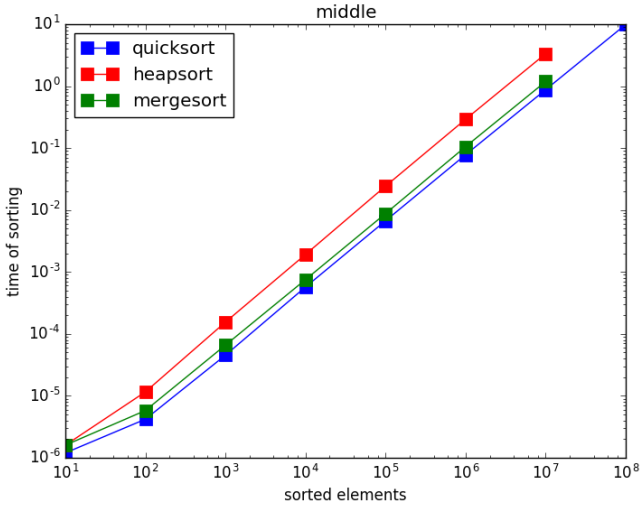
Rysunek 5: Przypadki dla algorytmu sortowania Heapsort

3.2 Wybrane przypadki dla wszystkich algorytmów sortowania



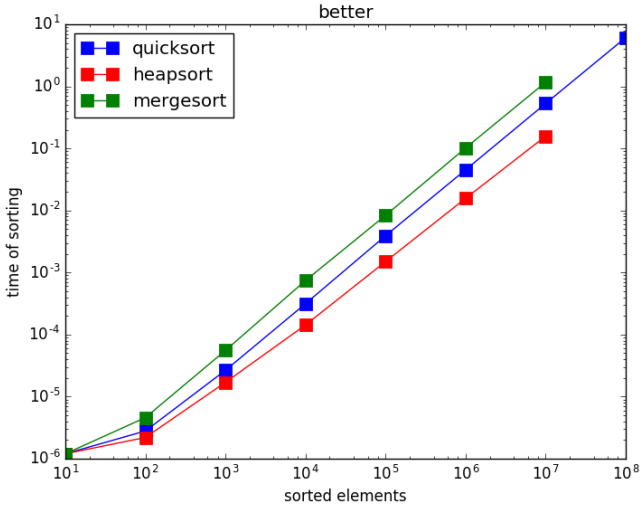
(a) worst-case

Rysunek 6: Przypadki pesymistyczne dla wszystkich algorytmów sortowania



(a) normal-case

Rysunek 7: Przypadki normalne dla wszystkich algorytmów sortowania



(a) the best-case

Rysunek 8: Przypadki optymistyczne dla wszystkich algorytmów sortowania

4 Tabelki

4.1 Quicksort

4.1.1 Przypadek optymistyczny

ilość elementów	czas[s]
10	1.2e-06
100	2.2e-06
1000	1.7e-05
10000	0.000143
100000	0.0015008
1000000	0.0156854
10000000	0.157468

4.1.2 Przypadek normalny

ilość elementów	czas[s]
10	1.2e-06
100	4.2e-06
1000	4.54e-05
10000	0.0005718
100000	0.0067092
1000000	0.0780638
10000000	0.876317
100000000	9.88534

4.1.3 Przypadek pesymistyczny

ilość elementów	czas[s]
10	1.4e-06
100	1.36e-05
1000	0.0010008
10000	0.100182
100000	10.0234

4.2 Mergesort

4.2.1 Przypadek optymistyczny

ilość elementów	czas[s]
10	1.2e-06
100	4.6e-06
1000	5.56e-05
10000	0.0007434
100000	0.0083426
1000000	0.101612
10000000	1.16551

4.2.2 Przypadek normalny

ilość elementów	czas[s]
10	1.6e-06
100	5.8e-06
1000	6.6e-05
10000	0.000753
100000	0.0087794
1000000	0.105519
10000000	1.18686

4.2.3 Przypadek pesymistyczny

ilość elementów	czas[s]
10	1.4e-06
100	6.2e-06
1000	6.68e-05
10000	0.0008674
100000	0.0103934
1000000	0.123629
10000000	1.41516

4.3 Heapsort

4.3.1 Przypadek optymistyczny

ilość elementów	czas[s]
10	1.2e-06
100	2.2e-06
1000	1.7e-05
10000	0.000143
100000	0.0015008
1000000	0.0156854
10000000	0.157468

4.3.2 Przypadek normalny

ilość elementów	czas[s]
10	1.6e-06
100	1.16e-05
1000	0.0001552
10000	0.0019166
100000	0.024317
1000000	0.292083
10000000	3.34007

4.3.3 Przypadek pesymistyczny

ilość elementów	czas[s]
10	2e-06
100	1.46e-05
1000	0.000202
10000	0.0023866
100000	0.0296416
1000000	0.36103
10000000	4.55591

5 Wnioski

Sortowanie Szybkie(QuickSort) charakteryzuje się oczekiwaną złożonością obliczeniową O(n lg n). W przypadku optymistycznym ilość porównań wykonywanych we wszystkich operacjach wynosi około n(lgn+1), co oznacza że złóżoność obliczeniowa wynosi O(n lg n). W przypadku średnim, gdy liczby są ułożone losowo złożoność obliczeniowa podobna jest do O(n lg n). W przypadku pesymistycznym z kolei wynosic O(n^2). Quicksort najbardziej sprawdza się do sortowania dużych ilości elementów.

Sortowanie przez scalanie(MergeSort) w przypadku optymistycznym i średnim charakteryzuje się złożonością obliczeniową O(n lg n). W przypadku pesymistycznym złożoność ta również wynosi O(n lg n).Jednak algorytm sortowania przez scalanie nie stosuje się w przypadku dużych ilości danych, ze zwględu na wykorzystanie dodatkowego miejsca w pamięci podczas scalania tablic. Sprawdza się najlepiej w przypadkach pesymistycznych.

Sortowanie przez kopcowanie(HeapSort) charakteryzuje sie oczekiwaną złożonością obliczeniową O(n lg n). W przypadku optymistycznym, średnim i pesymistycznym wynosi ona O(n lg n).Heapsort jest gorszy od QuickSorta, jednak w przypadkach pesymistycznych sprawia się lepiej. Wolniejszy od MergeSorta. Najlepiej sprawdza się w optymalnych przypadkach.

Wyniki Naszych badań związanych z obliczaniem złożoności czasowej poszczególnych algorytmów w różnych przypadkach są podobne do tych w książkach i w internecie, co sugeruje poprawną implementacje i sposób działania algorytmów.

6 Wykorzystane biblioteki

Python:

1. numpy
2. matplotlib.pyplot
3. os
4. pathlib