PAMSI 4,5

Wygenerowano przez Doxygen 1.8.6

Pn, 7 kwi 2014 04:00:36

Spis treści

1 Strona główna

Laboratorium 4 i 5.

Implementacje algorytmow sortowania: quicksort, mergesort, heapsort. Porownanie algorytmu quicksort dla zwyklego i najgorszego przypadku (sortowanie tablicy posortowanej) z wykorzystaniem quicksorta, ktory wybiera pierwszy lub losowy element do dzielenia problemu.

Autor

Jakub Chmiel 200314

2 Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

4 Dokumentacja klas

4.1 Dokumentacja klasy SortowanieKopcowanie

Klasa implementuje algorytm sortowania przez kopcowanie (heapsort).

`#include <SortowanieKopcowanie.h>`

Statyczne metody publiczne

- static void sortuj (StrukturaDanych &do_sortowania)
  Sortowanie dla calej struktury.

4.1.1 Opis szczegółowy

Klasa implementuje algorytm sortowania przez kopcowanie (heapsort).

4.1.2 Dokumentacja funkcji składowych

4.1.2.1 void SortowanieKopcowanie::sortuj (StrukturaDanych & *do_sortowania*) [static]

Sortowanie dla calej struktury.

Parametry

&do_sortowania	Struktura ktora chcemy sortowac.

Dokumentacja dla tej klasy została wygenerowana z plików:

- SortowanieKopcowanie.h
- SortowanieKopcowanie.cpp

4.2 Dokumentacja klasy SortowanieScalanie

Klasa implementuje algorytm sortowania przez scalanie (mergesort).

`#include <SortowanieScalanie.h>`

Statyczne metody publiczne

- static void sortuj (StrukturaDanych &do_sortowania)
  Sortowanie dla calej struktury.

4.2.1 Opis szczegółowy

Klasa implementuje algorytm sortowania przez scalanie (mergesort).

4.2.2 Dokumentacja funkcji składowych

4.2.2.1 void SortowanieScalanie::sortuj (StrukturaDanych & *do_sortowania*) [static]

Sortowanie dla calej struktury.

Parametry

&do_sortowania	Struktura ktora chcemy sortowac.

Dokumentacja dla tej klasy została wygenerowana z plików:

- SortowanieScalanie.h
- SortowanieScalanie.cpp

4.3 Dokumentacja klasy SortowanieSzybkie

Klasa implementuje algorytm sortowania szybkiego (quicksort) z wyborem pierwszego elementu jako dzielacego.

`#include <SortowanieSzybkie.h>`

Statyczne metody publiczne

- static void sortuj (StrukturaDanych &do_sortowania)
  Sortowanie dla calej struktury.

4.3.1 Opis szczegółowy

Klasa implementuje algorytm sortowania szybkiego (quicksort) z wyborem pierwszego elementu jako dzielacego.

4.3.2 Dokumentacja funkcji składowych

4.3.2.1 void SortowanieSzybkie::sortuj (StrukturaDanych & *do_sortowania*) [static]

Sortowanie dla calej struktury.

Parametry

&do_sortowania	Struktura ktora chcemy sortowac.

Dokumentacja dla tej klasy została wygenerowana z plików:

- SortowanieSzybkie.h
- SortowanieSzybkie.cpp

4.4 Dokumentacja klasy SortowanieSzybkie2

Klasa implementuje algorytm sortowania szybkiego (quicksort) z losowym wyborem elementu dzielacego.

`#include <SortowanieSzybkie2.h>`

Statyczne metody publiczne

- static void sortuj (StrukturaDanych &do_sortowania)

  Sortowanie dla calej struktury.

4.4.1 Opis szczegółowy

Klasa implementuje algorytm sortowania szybkiego (quicksort) z losowym wyborem elementu dzielacego.

4.4.2 Dokumentacja funkcji składowych

4.4.2.1 void SortowanieSzybkie2::sortuj (StrukturaDanych & *do_sortowania*) [static]

Sortowanie dla calej struktury.

Parametry

&do_sortowania	Struktura ktora chcemy sortowac.

Dokumentacja dla tej klasy została wygenerowana z plików:

- SortowanieSzybkie2.h
- SortowanieSzybkie2.cpp

4.5 Dokumentacja klasy StrukturaDanych

Struktura danych o funkcjonalnosci tablicy.

`#include <StrukturaDanych.h>`

Metody publiczne

- StrukturaDanych (StrukturaDanych &inna)

  Konstruktor kopiujacy.
- const int ilosc_elementow ()

  zwraca ilosc elementow w strukturze.
- const TYP element_na (int i)

  zwraca wartosc elementu na danym indeksie
- StrukturaDanych & operator= (StrukturaDanych &T)

  Operator przypisania.

- StrukturaDanych & operator+= (StrukturaDanych &T)

  Operator dodawania z przypisaniem.

Statyczne metody publiczne

- static bool zmien_element (StrukturaDanych &T, int index, TYP e)

  Nadpisuje element na danym indeksie.
- static bool kopiuj_wycinek (StrukturaDanych &zrodlo, StrukturaDanych &cel, int i_od, int i_do)

  Dodaje elementy ze struktury zrodlowej do docelowej z indeksow od i_od do i_do.
- static bool zamien_elementy (StrukturaDanych &T, int i, int j)

  Zamienia kolejnosc dwoch dowolnych elementow.
- static bool odwroc_kolejnosc (StrukturaDanych &T)

  Odwraca kolejnosc wszystkich elementow struktury.
- static bool dodaj_element (StrukturaDanych &T, TYP e)

  Dodaje element na koniec struktury.
- static StrukturaDanych dodaj_elementy (StrukturaDanych &T1, StrukturaDanych &T2)

  Laczy 2 struktury ze soba.
- static void wypisz_wszystko (StrukturaDanych &T)

  Wypisuje dane ze struktury na standardowe wyjscie.

Przyjaciele

- StrukturaDanych operator+ (StrukturaDanych T1, StrukturaDanych &T2)

  Laczy 2 struktury ze soba.
- bool operator== (StrukturaDanych &T1, StrukturaDanych &T2)

  Operator porownania.

4.5.1 Opis szczegółowy

Struktura danych o funkcjonalnosci tablicy.

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 StrukturaDanych::StrukturaDanych (StrukturaDanych & *inna*)

Konstruktor kopiujacy.

Parametry

inna	kopiowana StrukturaDanych

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 bool StrukturaDanych::dodaj_element (StrukturaDanych & *T*, TYP *e*) [static]

Dodaje element na koniec struktury.

Parametry

T	docelowa StrukturaDanych
e	element do dodania

Zwraca

- true sukces
- false porazka

4.5.3.2 StrukturaDanych StrukturaDanych::dodaj_elementy (StrukturaDanych & *T1,* StrukturaDanych & *T2*) [static]

Laczy 2 struktury ze soba.

Parametry

T1	pierwsza StrukturaDanych
T2	druga StrukturaDanych

Zwraca

Struktura bedaca polaczeniem dwoch wejsciowych struktur.

4.5.3.3 const TYP StrukturaDanych::element_na (int *i*)

zwraca wartosc elementu na danym indeksie

Parametry

i	indeks

Zwraca

wartosc na i-tym indeksie

4.5.3.4 const int StrukturaDanych::ilosc_elementow ( )

zwraca ilosc elementow w strukturze.

Zwraca

ilosc elementow.

4.5.3.5 bool StrukturaDanych::kopiuj_wycinek (StrukturaDanych & *zrodlo,* StrukturaDanych & *cel,* int *i_od,* int *i_do*) [static]

Dodaje elementy ze struktury zrodlowej do docelowej z indeksow od i_od do i_do.

Parametry

&zrodlo	Struktura zrodlowa.
&cel	Struktura docelowa
i_od	Indeks pierwszego elementu ktory chcemy kopiowac.
i_do	Indeks ostatniego elementu ktory chcemy kopiowac.

Zwraca

Stan powodzenia funkcji.

4.5.3.6 bool StrukturaDanych::odwroc_kolejnosc (StrukturaDanych & *T*) [static]

Odwraca kolejnosc wszystkich elementow struktury.

Parametry

T	docelowa StrukturaDanych

Zwraca

- true sukces
- false porazka

4.5.3.7 StrukturaDanych & StrukturaDanych::operator+= (StrukturaDanych & *T*)

Operator dodawania z przypisaniem.

Parametry

T	StrukturaDanych

Zwraca

Struktura z dodanymi na koniec elementami struktury T

4.5.3.8 StrukturaDanych & StrukturaDanych::operator= (StrukturaDanych & *T*)

Operator przypisania.

Parametry

T	StrukturaDanych

Zwraca

taka sama struktura jak parametr T

4.5.3.9 void StrukturaDanych::wypisz_wszystko (StrukturaDanych & *T*) [static]

Wypisuje dane ze struktury na standardowe wyjscie.

Parametry

T	docelowa StrukturaDanych

4.5.3.10 bool StrukturaDanych::zamien_elementy (StrukturaDanych & *T*, int *i*, int *j*) [static]

Zamienia kolejnosc dwoch dowolnych elementow.

Parametry

T	docelowa StrukturaDanych
i	indeks 1
j	indeks 2

Zwraca

- true sukces
- false porazka

4.5.3.11 bool StrukturaDanych::zmien_element (StrukturaDanych & *T*, int *index*, TYP *e*) [static]

Nadpisuje element na danym indeksie.

Parametry

&T	Docelowa struktura.
index	Indeks elementu do nadpisu.
e	nowy element.

Zwraca

Stan powodzenia funkcji.

4.5.4 Dokumentacja przyjaciół i funkcji związanych

4.5.4.1 StrukturaDanych operator+ (StrukturaDanych *T1,* StrukturaDanych & *T2*) [friend]

Laczy 2 struktury ze soba.

Parametry

T1	pierwsza StrukturaDanych
T2	druga StrukturaDanych

Zwraca

Struktura bedaca polaczeniem dwoch wejsciowych struktur.

4.5.4.2 bool operator== (StrukturaDanych & *T1,* StrukturaDanych & *T2*) [friend]

Operator porownania.

Parametry

T1	pierwsza StrukturaDanych
T2	druga StrukturaDanych

Zwraca

- true struktury maja identyczne elementy
- false elementy nie sa identyczne

Dokumentacja dla tej klasy została wygenerowana z plików:

- StrukturaDanych.h
- StrukturaDanych.cpp

5 Dokumentacja plików

5.1 Dokumentacja pliku definicje.h

Plik zawiera ogolne instrukcje preprocesora wspolne dla wszystkich plikow zrodlowych.

Definicje

- #define **TYP** int

5.1.1 Opis szczegółowy

Plik zawiera ogolne instrukcje preprocesora wspolne dla wszystkich plikow zrodlowych.

5.2 Dokumentacja pliku SortowanieKopcowanie.cpp

```
#include "SortowanieKopcowanie.h"
```

5.3 Dokumentacja pliku SortowanieKopcowanie.h

```
#include "StrukturaDanych.h"
```

Komponenty

- class SortowanieKopcowanie

  Klasa implementuje algorytm sortowania przez kopcowanie (heapsort).

5.4 Dokumentacja pliku SortowanieScalanie.cpp

```
#include "SortowanieScalanie.h"
```

5.5 Dokumentacja pliku SortowanieScalanie.h

```
#include "StrukturaDanych.h"
```

Komponenty

- class SortowanieScalanie

  Klasa implementuje algorytm sortowania przez scalanie (mergesort).

5.6 Dokumentacja pliku SortowanieSzybkie.cpp

```
#include "SortowanieSzybkie.h"
```

5.7 Dokumentacja pliku SortowanieSzybkie.h

```
#include "StrukturaDanych.h"
```

Komponenty

- class SortowanieSzybkie

  Klasa implementuje algorytm sortowania szybkiego (quicksort) z wyborem pierwszego elementu jako dzielacego.

5.8 Dokumentacja pliku SortowanieSzybkie2.cpp

```
#include "SortowanieSzybkie2.h"
```

5.9 Dokumentacja pliku SortowanieSzybkie2.h

```
#include "StrukturaDanych.h"
#include <time.h>
```

Komponenty

- class SortowanieSzybkie2

  Klasa implementuje algorytm sortowania szybkiego (quicksort) z losowym wyborem elementu dzielacego.

5.10 Dokumentacja pliku StrukturaDanych.cpp

```
#include "StrukturaDanych.h"
```

Funkcje

- StrukturaDanych operator+ (StrukturaDanych T1, StrukturaDanych &T2)

  Laczy 2 struktury ze soba.
- bool operator== (StrukturaDanych &T1, StrukturaDanych &T2)

  Operator porownania.

5.10.1 Dokumentacja funkcji

5.10.1.1 StrukturaDanych operator+ (StrukturaDanych *T1*, StrukturaDanych & *T2*)

Laczy 2 struktury ze soba.

Parametry

T1	pierwsza StrukturaDanych
T2	druga StrukturaDanych

Zwraca

Struktura bedaca polaczeniem dwoch wejsciowych struktur.

5.10.1.2 bool operator== (StrukturaDanych & *T1*, StrukturaDanych & *T2*)

Operator porownania.

Parametry

T1	pierwsza StrukturaDanych
T2	druga StrukturaDanych

Zwraca

- true struktury maja identyczne elementy
- false elementy nie sa identyczne

5.11 Dokumentacja pliku StrukturaDanych.h

```
#include <iostream>
#include "definicje.h"
```

Komponenty

- class StrukturaDanych

  Struktura danych o funkcjonalnosci tablicy.

5.12 Dokumentacja pliku TestSortowania.cpp

```
#include "TestSortowania.h"
```

Funkcje

- bool wczytaj_dane (const char ∗nazwa_pliku, StrukturaDanych &tablica)

  wczytuje dane z pliku
- bool zapisz_dane (const char ∗nazwa_pliku, int ∗col_rozmiar_problemu, double ∗col_czas, int rozmiar)

  zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu
- void generuj_dane (char ∗nazwa_pliku)

  Generuje nowe dane do pliku.
- void testuj_algorytm (void(∗algorytm)(StrukturaDanych &dane), StrukturaDanych &dane, const char ∗plik_wyjsciowy)

  wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku
- StrukturaDanych sprawdz_poprawnosc (void(∗algorytm)(StrukturaDanych &dane), StrukturaDanych &dane, int ile_liczb)

  funkcja sprawdza poprawnosc dzialania struktur danych.
- int main ()

  funkcja main

5.12.1 Dokumentacja funkcji

5.12.1.1 StrukturaDanych sprawdz_poprawnosc (void(∗)(StrukturaDanych &dane) *algorytm,* StrukturaDanych & *dane,* int *ile_liczb*)

funkcja sprawdza poprawnosc dzialania struktur danych.

Parametry

void(∗algorytm)(StrukturaDanych&	dane) Wskaznik na wybrany algorytm.
dane	Dane do obrobki, niezmieniane.
ile_liczb	Wielkosc problemu do sprawdzenia, niewielka aby dalo sie zobaczyc w konsoli.

Zwraca

Zwraca Strukture po wykonaniu algorytmu.

5.12.1.2 void testuj_algorytm (void(∗)(StrukturaDanych &dane) *algorytm,* StrukturaDanych & *dane,* const char ∗ *plik_wyjsciowy*)

wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku

Parametry

double(∗algorytm)	funkcja z algorytmem do testowania
∗tablica	dane dla algorytmu
rozmiar	rozmiar tablicy
∗plik_wyjsciowy	nazwa pliku do zapisu zmierzonych czasow

5.12.1.3 bool wczytaj_dane (const char ∗ *nazwa_pliku,* StrukturaDanych & *tablica*)

wczytuje dane z pliku

Format:

liczba_danych

dana1

dana2

.

Parametry

∗nazwa_pliku	nazwa pliku z danymi
∗&tablica	tablica docelowa (usuwana w przypadku !=NULL)
&rozmiar	rozmiar tablicy docelowej

5.12.1.4 bool zapisz_dane (const char ∗ *nazwa_pliku,* int ∗ *col_rozmiar_problemu,* double ∗ *col_czas,* int *rozmiar*)

zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu

Parametry

∗nazwa_pliku	nazwa pliku do zapisu
∗col_rozmiar_problemu	tablica z 1 kolumna
∗col_czas	druga kolumna
rozmiar	rozmiar obu tablic

Zwraca

- true sukces
- false blad

5.13 Dokumentacja pliku TestSortowania.h

```
#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>
#include "definicje.h"
#include "StrukturaDanych.h"
#include "SortowanieSzybkie.h"
#include "SortowanieSzybkie2.h"
#include "SortowanieScalanie.h"
#include "SortowanieKopcowanie.h"
```

Definicje

- #define PLIK_DANYCH "dane4.txt"

  nazwa pliku z danymi wejsciowymi, takze dla wyjscia generowanych liczb.
- #define LICZBA_POWTORZEN 10

  ilosc powtorzen pomiaru czasu dla kazdego rozmiaru problemu.
- #define LICZBA_WIELKOSCI 8

  ilosc roznych rozmiarow problemu.
- #define WIELKOSCI_PROBLEMU {100, 1000, 5000, 10000, 20000, 40000, 60000, 100000}

  tablica zawierajaca wszystkie mierzone rozmiary problemu.
- #define WIELKOSC_GENEROWANYCH_DANYCH 1000000

  ilosc danych do wygenerowania.

Funkcje

- bool wczytaj_dane (const char ∗nazwa_pliku, StrukturaDanych &tablica)

  wczytuje dane z pliku
- bool zapisz_dane (const char ∗nazwa_pliku, int ∗col_rozmiar_problemu, double ∗col_czas, int rozmiar)

  zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu
- void generuj_dane (char ∗nazwa_pliku)

  Generuje nowe dane do pliku.
- void testuj_algorytm (void(∗algorytm)(StrukturaDanych &dane), StrukturaDanych &dane, const char ∗plik_wyjsciowy)

  wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku
- StrukturaDanych sprawdz_poprawnosc (void(∗algorytm)(StrukturaDanych &dane), StrukturaDanych &dane, int ile_liczb)

  funkcja sprawdza poprawnosc dzialania struktur danych.
- int main ()

  funkcja main

5.13.1 Dokumentacja funkcji

5.13.1.1 StrukturaDanych sprawdz_poprawnosc (void(∗)(StrukturaDanych &dane) *algorytm,* StrukturaDanych & *dane,* int *ile_liczb*)

funkcja sprawdza poprawnosc dzialania struktur danych.

Parametry

void(∗algorytm)(StrukturaDanych&	dane) Wskaznik na wybrany algorytm.
dane	Dane do obrobki, niezmieniane.
ile_liczb	Wielkosc problemu do sprawdzenia, niewielka aby dalo sie zobaczyc w konsoli.

Zwraca

Zwraca Strukture po wykonaniu algorytmu.

5.13.1.2 void testuj_algorytm (void(∗)(StrukturaDanych &dane) *algorytm,* StrukturaDanych & *dane,* const char ∗ *plik_wyjsciowy*)

wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku

Parametry

double(∗algorytm)	funkcja z algorytmem do testowania
∗tablica	dane dla algorytmu
rozmiar	rozmiar tablicy
∗plik_wyjsciowy	nazwa pliku do zapisu zmierzonych czasow

5.13.1.3 bool wczytaj_dane (const char ∗ *nazwa_pliku,* StrukturaDanych & *tablica*)

wczytuje dane z pliku

Format:

liczba_danych

dana1

dana2

.

Parametry

∗nazwa_pliku	nazwa pliku z danymi
∗&tablica	tablica docelowa (usuwana w przypadku !=NULL)
&rozmiar	rozmiar tablicy docelowej

5.13.1.4 bool zapisz_dane (const char ∗ *nazwa_pliku,* int ∗ *col_rozmiar_problemu,* double ∗ *col_czas,* int *rozmiar*)

zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu

Parametry

∗nazwa_pliku	nazwa pliku do zapisu
∗col_rozmiar_problemu	tablica z 1 kolumna
∗col_czas	druga kolumna
rozmiar	rozmiar obu tablic

Zwraca

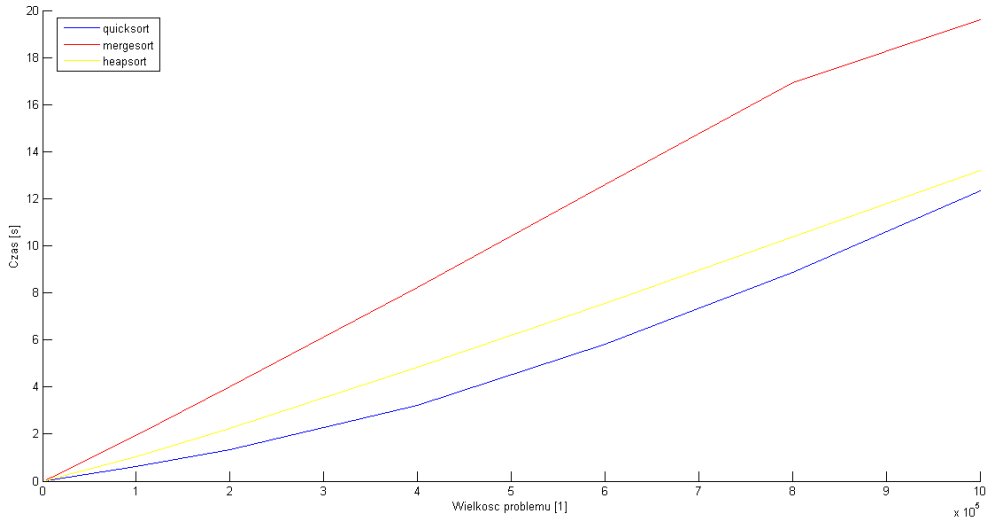
- true sukces
- false blad

6 Sprawozdanie z wykonania programu labolatorium 4.

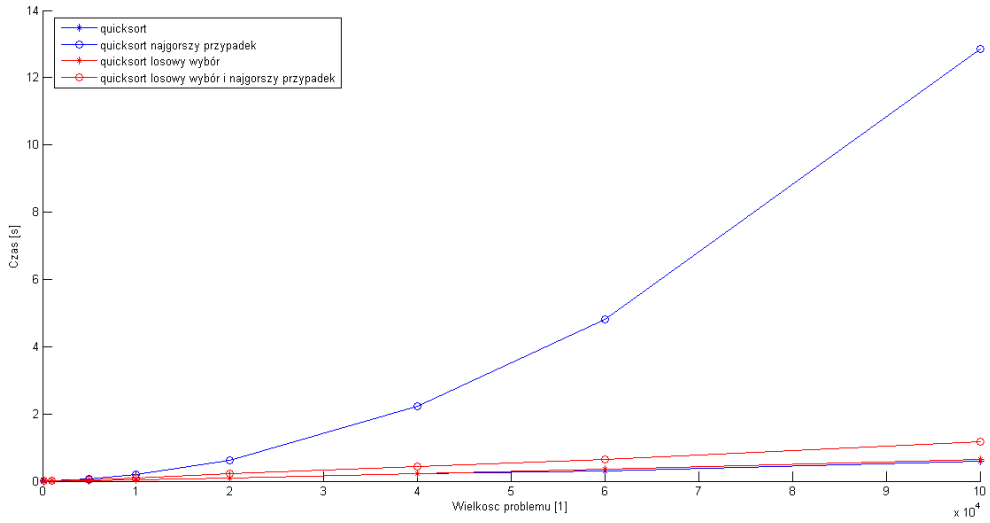
Pomiar czasu dla algorytmów quicksort, mergesort i heapsort przedstawiono na wykresie 1. Z wykresu wynika, że każdy algorytm ma podobną złożoność obliczeniową (teoretycznie liniowo logarytmiczną). W kolejności od najszybszego do najwolniejszego wypadły: quicksort, heapsort i mergesort.

7 Sprawozdanie z wykonania programu labolatorium 5.

Pomiar czasu dla algorytmu quicksort w normalnym i najgorszym przypadku w dwóch wersjach. Wersja pierwsza wybiera pierwszy element do dzielenia problemu (na wykresie kolor niebieski). Wersja druga wybiera losowy element do dzielenia problemu (na wykresie kolor czerwony). 1. W przypadku zwykłym obie wersje algorytmu spisują się podobnie. W przypadku najgorszym, wersja z wyborem pierwszego elementu ma złożoność kwadratową, na-



Rysunek 1: Wykres pomiaru czasu algorytmów sortowania.



Rysunek 2: Wykres pomiaru czasu algorytmu quicksort.

tomiast wersja z losowym wyborem ma podobną złożoność jak w normalnym przypadku, jednak zajmuje około dwukrotnie więcej czasu.

Skorowidz