Politechnika Wrocławska

Wstęp do programowania

INP001213Wcl

rok akademicki 2018/19

semestr zimowy

Wykład 13

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

A-1 p. 411B

Plan prezentacji (1)

- Złożoność algorytmów
  - czy to istotne, skoro komputery są coraz szybsze?
  - optymalizacja algorytmów
  - notacja „duże-O”
  - rekurencja, a złożoność
  - ograniczenia górne i dolne
  - luka algorytmiczna

Na podstawie:

- D. Harel, *Rzecz o istocie informatyki. Algorytmika*

Plan prezentacji (2)

- Struktura w C - przykład
- Użycie funkcji w celu ukrycia układu struktury

- Coś więcej niż C?
- Funkcje jako pola struktury
- Klasa – rozszerzenie struktury

Na podstawie:

- Alex Allain, *C++ : przewodnik dla początkujących*

Przykład algorytmu

KAKAOWE CIASTO BEZ PIECZENIA

POPULARNE CIASTA | SZYBKIE CIASTO | DESERY BEZ PIECZENIA | CIASTA | CIASTA CZEKOLADOWE | CIASTA KOSTKI | CZEKOLADA

Rewelacyjne ciasto bez pieczenia! Kakaowe hebatniki przełożone kakaową masą budyniową i dżemem porzeczkowym (można połączyć pół na pół z powidłami śliwkowymi).

https://www.kwestiasmaku.com/przepis/kakaowe-ciasto-bez-pieczenia

Przykład algorytmu

SKŁADNIKI

KREM CZEKOLADOWY

- 1 litr mleka
- 3 łyżki mąki pszennej
- 3 łyżki mąki ziemniaczanej
- 1 szklanka cukru
- 3 łyżki kakao
- 2 żółtka

ORAZ

- ok. 600 g kakaowych herbatników "petit beurre"
- 200 g masła
- 1 słoiczek dżemu porzeczkowego lub powideł śliwkowych
- kakao

PRZYGOTOWANIE

KREM CZEKOLADOWY

- Odlać 1 i 1/2 szklanki mleka i dokładnie wymieszać je (np. rózgą) z mąką pszenną i ziemniaczaną, cukrem, żółtkami oraz likierami jeśli ich używamy.
- Resztę mleka zagotować (dokładnie, aż zacznie kipieć), następnie wlewać do niego mieszankę mleka, mąki i żółtek, jednocześnie energicznie mieszając rózgą. Zagotować co chwilę mieszając.
- Po zagotowaniu gotowy budyń odstawić z ognia, przelać do czystej miski i całkowicie ostudzić (na wierzch można położyć folię spożywczą aby nie zrobił się kożuch).
- Miękkie masło ubijać przez ok. 3 minuty aż się napuszy, następnie stopniowo, w krótkich odstępach czasu, dodawać budyń ciągle ubijając.

https://www.kwestiasmaku.com/przepis/kakaowe-ciasto-bez-pieczenia

Politechnika Wrocławska

Przykład algorytmu

PRZEŁOŻENIE

- Formę o wymiarach ok. **20 x 30 cm** (może być większa) wysmarować masłem i wyłożyć papierem do pieczenia. Układać warstwami na przemian herbatniki i krem budyniowy, otrzymując 4 lub 5 takich warstw, z tym, że druga warstwa od dołu ma mieć zamiast kremu - dżem. Na wierzchu też ma być cienka warstwa kremu.
- Ciasto oprószyć kakao i wstawić do lodówki na kilka godzin lub na całą noc. Ciasto można przygotować dzień wcześniej.

WSKAZÓWKI

Opcjonalnie do kremu można dodać 4 łyżki likieru pomarańczowego lub amaretto lub orzechowego lub 2 łyżki mocnego alkoholu.

https://www.kwestiasmaku.com/przepis/kakaowe-ciasto-bez-pieczenia

Przykład algorytmu 2

SKŁADNIKI

- 250 g mąki
- 1 łyżeczka soli
- 250 g chłodnego masła
- 150 ml wody

PRZYGOTOWANIE

DODAJ NOTATKĘ

- Mąkę przesiać do miski, dodać sól. Wymieszać. Masło pokroić na małe kawałki i kilka większych. Małe kawałki masła włożyć do miski z mąką i rozcierać palcami, aż uformują się grudki. Dodać większe kawałki masła i wymieszać, wówczas ciasto będzie bardziej kruche. Stopniowo dodając wodę, wyrabiać ciasto tylko do czasu, aż składniki się połączą.
- Z ciasta uformować kulę, zawinąć w folię spożywczą i wstawić do lodówki i trzy razy wałkować w jednym kierunku na lekko posypanej mąką stolnicy oraz składać na 3 części formując prostokąt, po każdym złożeniu wstawić do lodówki na 3 godziny.
- Rozwałkować i układać np. w rogaliki jak w przepisie powyżej lub formować dowolne kształty i wypełniać dowolnym nadzieniem słodkim lub słonym. Ciasto piec w temperaturze 200°C

https://www.kwestiasmaku.com/przepis/proste-ciasto-francuskie

Złożoność algorytmów

- Algorytmy powinny być poprawne
- Algorytmy powinny działać efektywnie

- Miary złożoności:
  - złożoność pamięciowa,
  - złożoność czasowa.

Złożoność algorytmów

- Koszt wykonania algorytmu zależy zwykle od danych wejściowych
- Złożoność pamięciową i czasową określamy jako funkcje rozmiaru danych wejściowych

Politechnika Wrocławska

Złożoność algorytmów

Problem komiwojażera

- Czy coraz szybsze komputery nie rozwiązują problemu?

- Problem komiwojażera:
  dane jest n miast, które komiwojażer ma odwiedzić, oraz koszt podróży pomiędzy każdą parą miast; celem jest znalezienie najtańszej drogi łączącej wszystkie miasta.

Złożoność algorytmów

Problem komiwojażera

- Sprawdzamy wszystkie możliwe ustawienia miast – jest ich $(n-1)!$
- Załóżmy, że superkomputer jest w stanie przejrzeć 100 miliardów (10^{11}) kombinacji na sekundę

Złożoność algorytmów

Problem komiwojażera

miasta w Polsce		
liczba mieszkańców	liczba miast	czas obliczeń
≥ 500 000	5	$2,4\times10^{-10}$ s

dane GUS z 24 lipca 2014 za https://pl.wikipedia.org/wiki/Miasta_w_Polsce

Złożoność algorytmów

Problem komiwojażera

miasta w Polsce		
liczba mieszkańców	liczba miast	czas obliczeń
≥ 500 000	5	$2{,}4 \times 10^{-10}$ s
≥ 200 000	16	13 s

dane GUS z 24 lipca 2014 za https://pl.wikipedia.org/wiki/Miasta_w_Polsce

Złożoność algorytmów

Problem komiwojażera

miasta w Polsce		
liczba mieszkańców	liczba miast	czas obliczeń
≥ 500 000	5	$2{,}4 \times 10^{-10}$ s
≥ 200 000	16	13 s
≥ 100 000	39	$1{,}7 \times 10^{26}$ lat

dane GUS z 24 lipca 2014 za https://pl.wikipedia.org/wiki/Miasta_w_Polsce

Złożoność algorytmów

Problem komiwojażera

miasta w Polsce		
liczba mieszkańców	liczba miast	czas obliczeń
≥ 500 000	5	$2{,}4\times10^{-10}$ s
≥ 200 000	16	13 s
≥ 100 000	39	$1{,}7\times10^{26}$ lat
≥ 40 000	111	$5{,}0\times10^{159}$ lat
≥ 20 000	222	
≥ 10 000	409	
≥ 5 000	587	
≥ 2 500	806	
wszystkich	923	

dane GUS z 24 lipca 2014 za https://pl.wikipedia.org/wiki/Miasta_w_Polsce

Optymalizacja algorytmu

- Normalizowanie wyników zaliczenia do 100 punktów:
  1. oblicz najwyższą ocenę w zmiennej **`max`**;
  2. dla każdej oceny **`y`** podstaw **`y*100/max`**;
- W obu krokach przebiegamy tablicę elementów, w pierwszym wyszukujemy maksimum, w drugim aktualizujemy wartości

```
2.   dla i od 1 do n wykonaj:
   2.1.  y[i] = y[i]*100/max;
```

Optymalizacja algorytmu

- W obu krokach przebiegamy tablicę elementów, w pierwszym wyszukujemy maksimum, w drugim aktualizujemy wartości

```
2.   dla i od 1 do n wykonaj:
  2.1.  y[i] = y[i]*100/max;
```

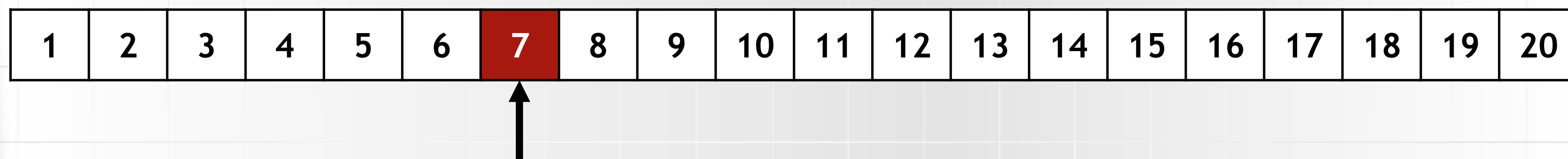
- Zmodyfikowany algorytm:

```
2.   czynnik = 100/max;
3.   dla i od 1 do n wykonaj:
  3.1.  y[i] = y[i]*czynnik;
```

- Zmodyfikowany algorytm wykonuje dwukrotnie mniej operacji arytmetycznych

Optymalizacja algorytmu (2)

Przeszukiwanie liniowe



- Złożoność czasowa proporcjonalna do długości danych: $O(N)$
- Notacja „duże-O" służy do wyrażania rzędu zależności
- Nie jest ważne, czy algorytm wykonuje się w N jednostkach czasu, czy w $3N$, $100N$, lub $N/5$
- Istnieje taka stała K, że w najgorszym przypadku algorytm wykona się w czasie krótszym, niż $K{\cdot}N$

Politechnika Wrocławska

Optymalizacja algorytmu (2)

Przeszukiwanie binarne

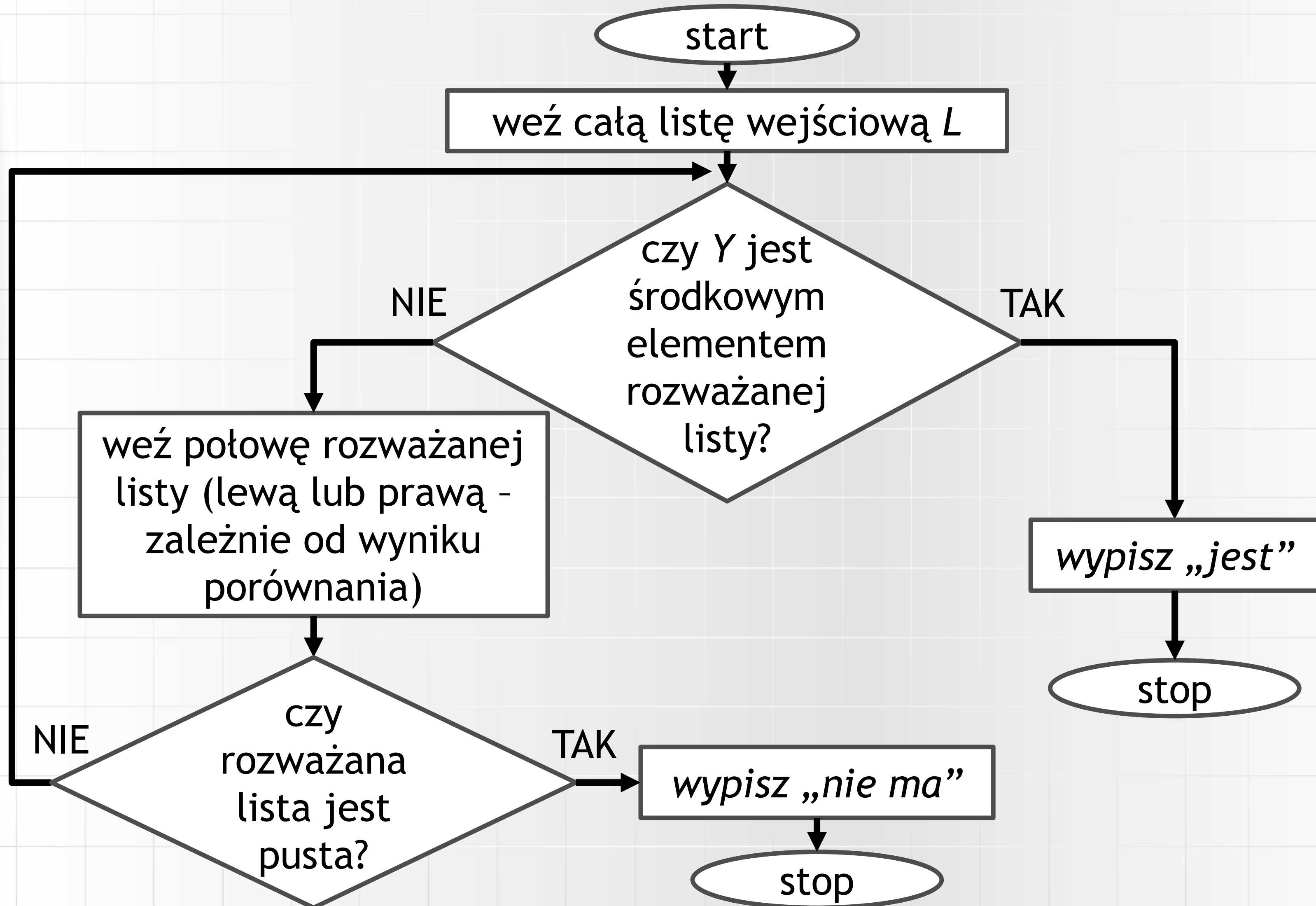
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Optymalizacja algorytmu (2)

Przeszukiwanie binarne



Politechnika Wrocławska

Optymalizacja algorytmu (2)

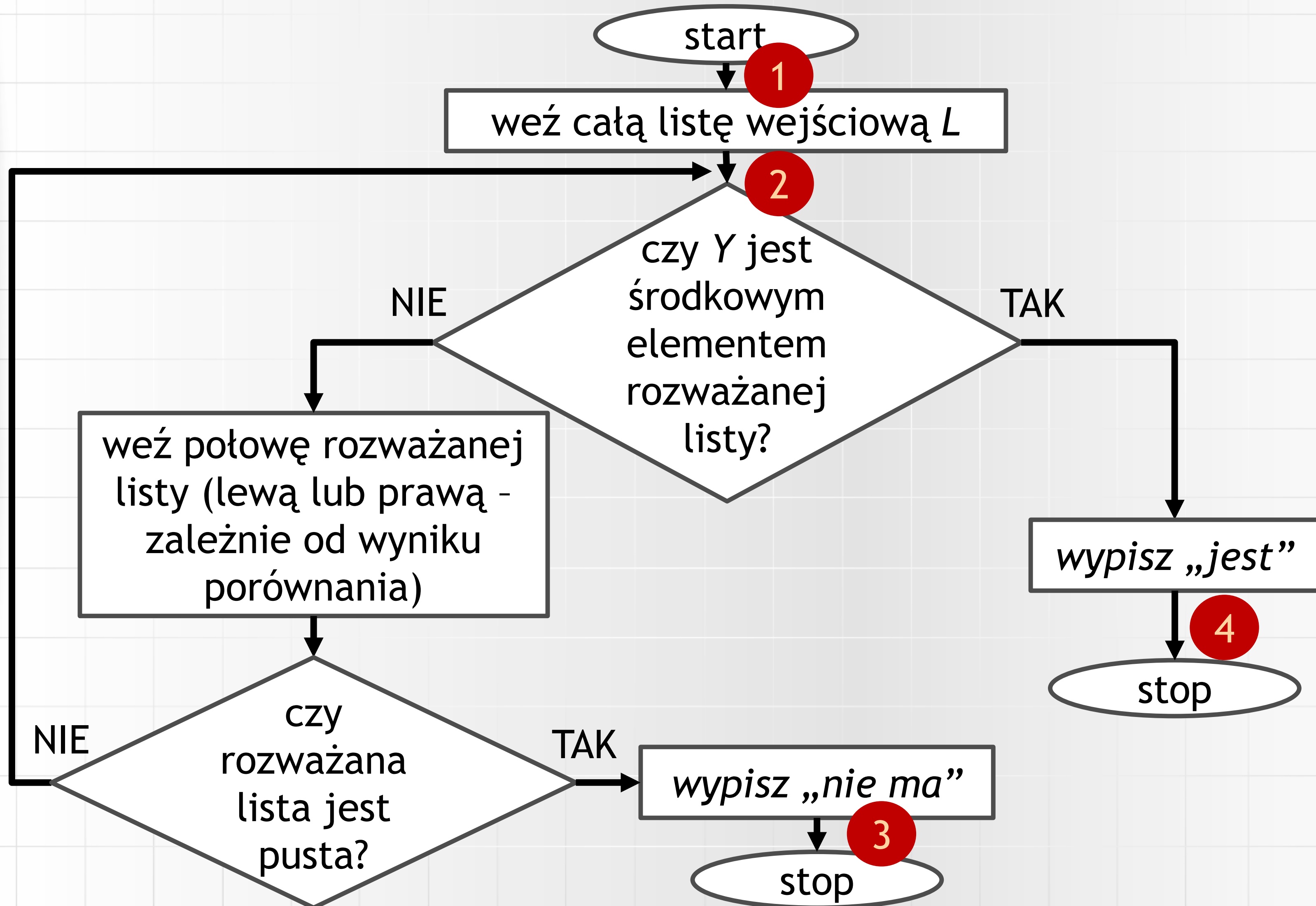
Przeszukiwanie binarne

- Złożoność czasowa przeszukiwania binarnego wynosi $O(\log_2 N)$

N	$1+\log_2 N$
10	4
100	7
1000	10
10^6	20
10^9	30
10^{18}	60

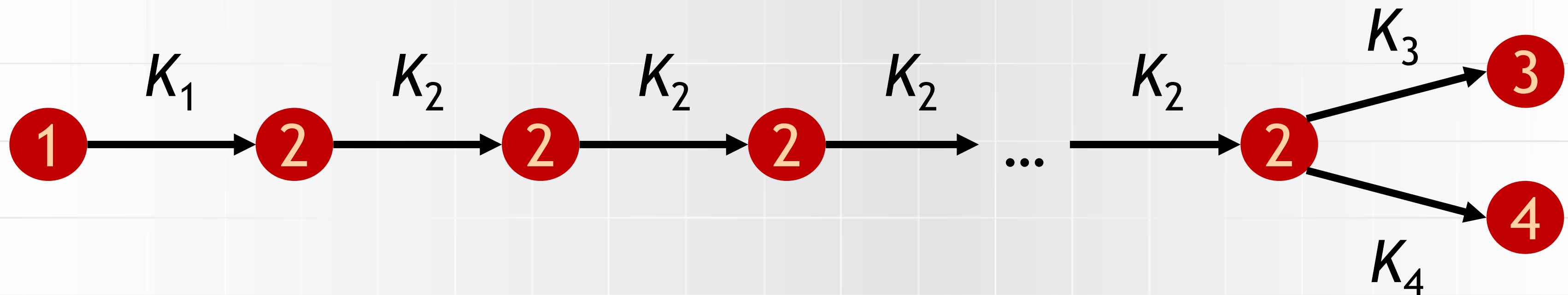
Optymalizacja algorytmu (2)

Przeszukiwanie binarne



Optymalizacja algorytmu (2)

Przeszukiwanie binarne



- $K = \max(K_1+K_3,\ K_1+K_4)$
- całkowita liczba instrukcji
  $K + (K_2 \cdot \log_2 N)$
- złożoność obliczeniowa $O(\log_2 N)$

Złożoność czasowa

- Złożoność czasowa ma sens w połączeniu ze zbiorem instrukcji elementarnych
- Liczby $\log_2 N$ oraz $\log_K N$ różnią się o stały czynnik, zatem $O(\log_2 N) = O(\log N)$

Złożoność czasowa

- Porównajmy dwa algorytmy rozwiązujące ten sam problem:
  - algorytm A ma złożoność O(logN),
  - algorytm B ma złożoność O(N).
- Który algorytm jest lepszy?

Złożoność czasowa

- Dodajmy, że czasy wykonywania algorytmów można ograniczyć następująco:
  - algorytm A $1000 \cdot \log_2 N$,
  - algorytm B $10 \cdot N$.
- Który algorytm jest lepszy?

Złożoność czasowa

N	$10 \cdot N$	$1000 \cdot \log_2 N$
10	100	3 321
100	1 000	6 643
1 000	10 000	9 965
10 000	100 000	13 288
1 000 000	10 000 000	19 932

Złożoność czasowa

Zagnieżdżone pętle

- Sortowanie bąbelkowe 1

```
(1) wykonaj n-1 razy:
      …
      (1.2) wykonaj n-1 razy:
                  …
```

$(n-1) \times (n-1) = n^2 - 2n + 1 = O(n^2)$

- Sortowanie bąbelkowe 2

$(n-1) + (n-2) + \ldots + 1 = (n^2 - n)/2 = O(n^2)$

Złożoność czasowa

Rekurencja

- Jednoczesne wyznaczanie minimum i maksimum

$h(1) = 0$

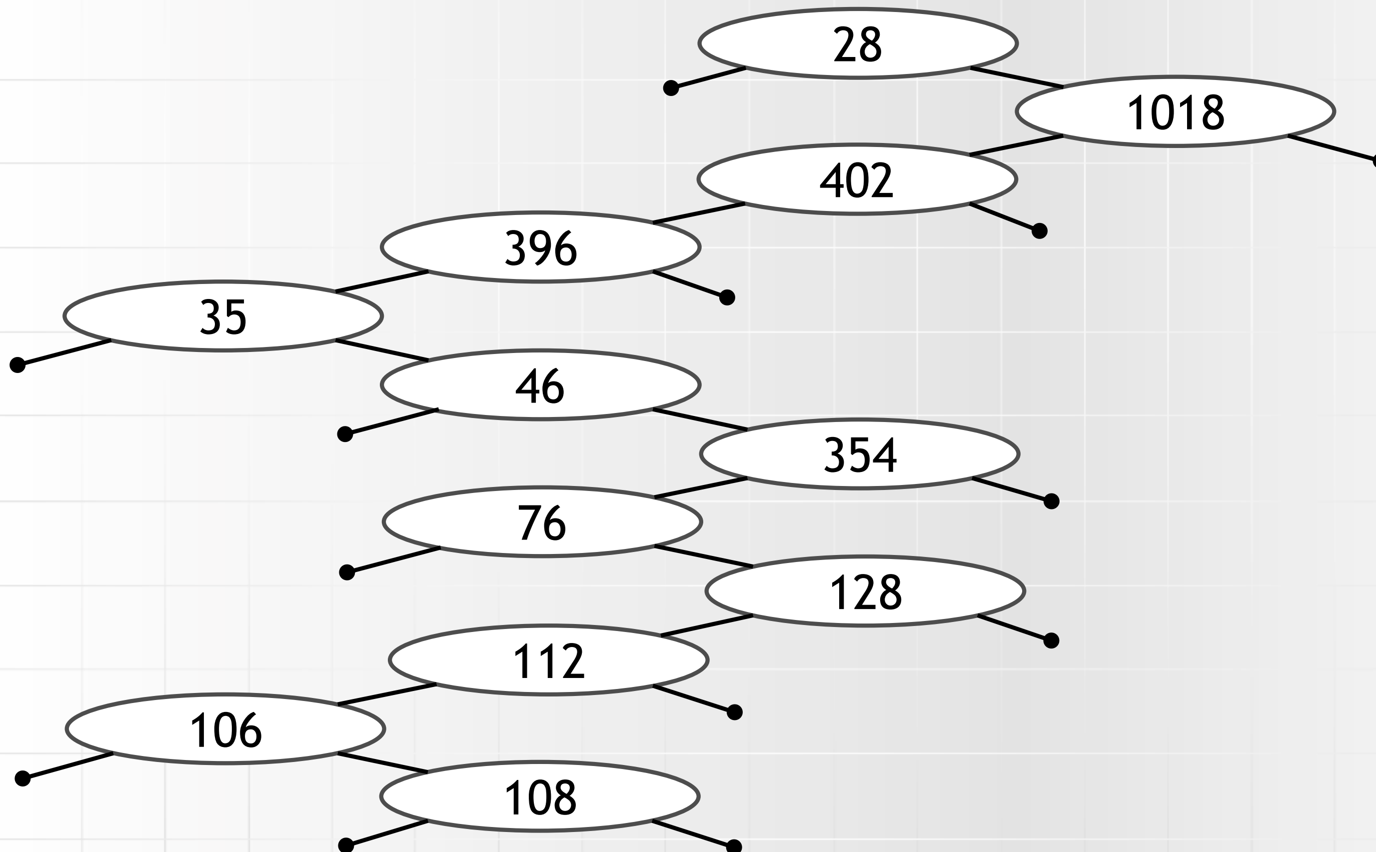
$h(2) = 1$

$h(n) = 2h(n/2) + 2$

$h(n) = 3n/2 - 2$

Złożoność czasowa

28	1018	402	396	35	46	354	76	128	112	106	108



Złożoność czasowa

- Złożoność pesymistyczna
- Złożoność średnia

- Przykładowo dla sortowania szybkiego:
  - pesymistycznie O(n^2)
  - średnio $O(n \cdot \log n)$ – $1{,}4 n \cdot \log_2 n$

Politechnika Wrocławska

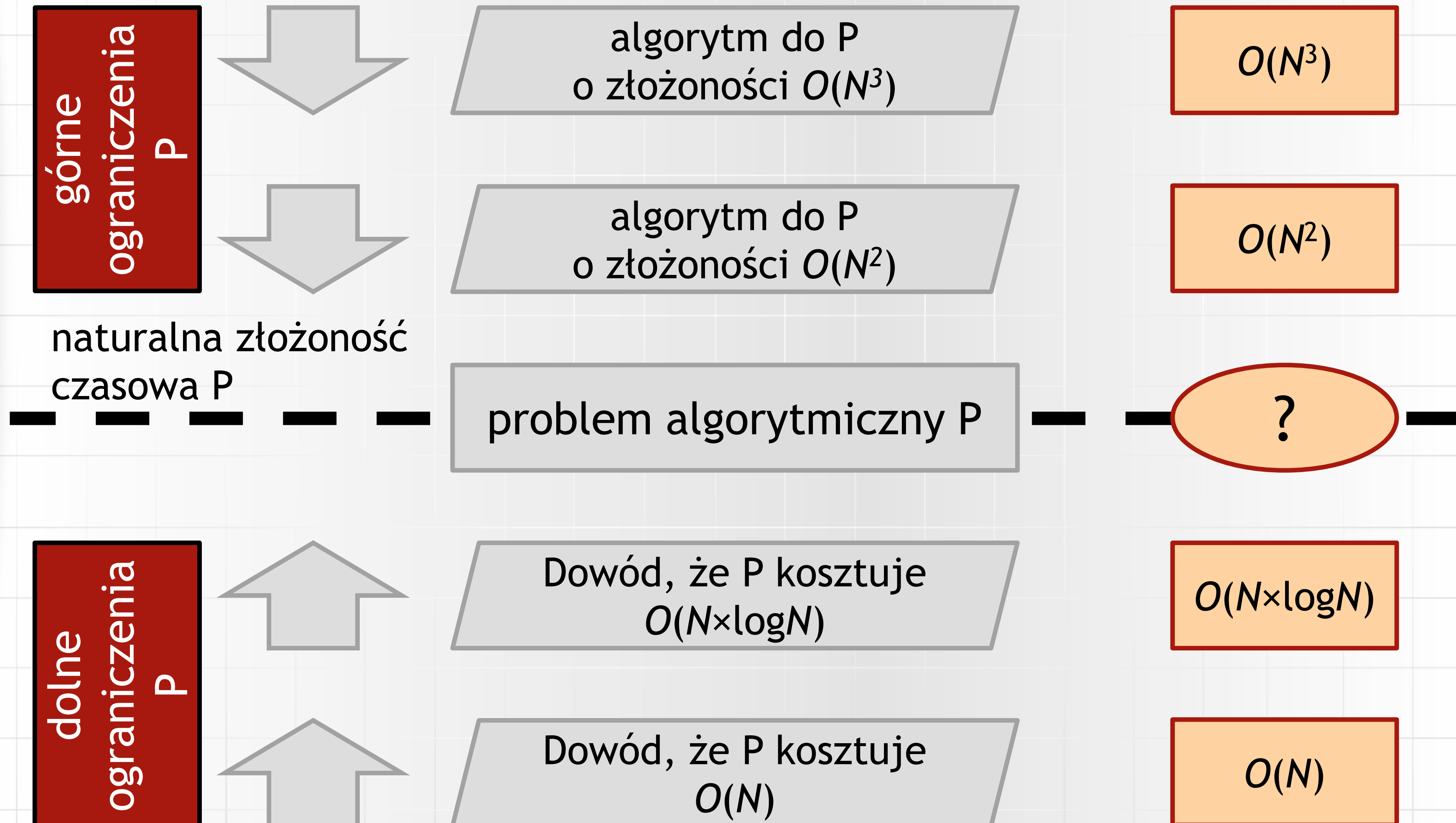
Złożoność czasowa

Górne i dolne ograniczenia

- Przeszukiwanie listy uporządkowanej:
  - przeszukiwanie liniowe $O(N)$,
  - przeszukiwanie binarne $O(\log N)$,
  - czy da się lepiej?
- Istnienie algorytmu o pewnej złożoności ustala **górne ograniczenie** problemu.
- Dowód, że danego problemu nie da się rozwiązać algorytmem o złożoności mniejszej niż podana ustanawia **dolne organicznie** problemu.

Złożoność czasowa

Górne i dolne ograniczenia



Złożoność czasowa

Górne i dolne ograniczenia

- Rozważając drzewo porównań dla pewnego algorytmu A możemy pokazać, że minimalna głębokość tego drzewa jest $\log_2 N$, zatem dolnym ograniczeniem dla przeszukiwania listy uporządkowanej jest $O(\log N)$.

Politechnika Wrocławska

Złożoność czasowa

Problemy zamknięte i luki algorytmiczna

- Problemy algorytmiczne, w których górne i dolne ograniczenia się spotykają nazywamy problemami zamkniętymi
- Problemy zamknięte:
  - przeszukiwanie listy uporządkowanej $O(\log N)$,
  - sortowanie $O(N\log N)$
- Problem minimalnego drzewa rozpinającego
  - ograniczenie dolne $O(N)$
  - ograniczenie górne $O(f(N)\cdot N)$

Podsumowanie (1)

- Złożoność algorytmów
  - czy to istotne, skoro komputery są coraz szybsze?
  - optymalizacja algorytmów
  - notacja „duże-O”
  - rekurencja, a złożoność
  - ograniczenia górne i dolne
  - luka algorytmiczna

Plan prezentacji (2)

- Struktura w C - przykład
- Użycie funkcji w celu ukrycia układu struktury

- Coś więcej niż C?
- Funkcje jako pola struktury
- Klasa – rozszerzenie struktury

Na podstawie:

- Alex Allain, *C++ : przewodnik dla początkujących*

Struktury - przykład

main.c

```c
/* point_01
Program definujący strukturę point
i funkcje obsługujące tę strukturę.
Program demonstruje możliwości
jej wykorzystania.
*/

#include <stdio.h>
#include <math.h>

struct point{
    double x,y;
};

typedef struct point point;

int    printPoint(point*);
double modulus(point*);
point  rotated(point*, double);

```

Struktury - przykład

main.c

```c
21  int main()
22  {
23      point a = {3,4}, b;
24      printf("a = ");printPoint(&a);printf("\n");
25      printf("a_x = %lf\n",a.x);
26      printf("a_y = %lf\n",a.y);
27      printf("|a| = %lf\n",modulus(&a));
28      b = rotated(&a,M_PI_2);
29      printf("b = ");printPoint(&b);printf("\n");
30      return 0;
31  }
32
```

Struktury - przykład

main.c

```
33  int printPoint(point*a){
34      return printf("[%lf,%lf]",a->x,a->y);
35  }
36
37  double modulus(point*a){
38      return sqrt(a->x*a->x + a->y*a->y);
39  }
40
41  point rotated(point*a, double alpha){
42      point b;
43      b.x = a->x*cos(alpha) - a->y*sin(alpha);
44      b.y = a->x*sin(alpha) + a->y*cos(alpha);
45      return b;
46  }
```

Użycie funkcji w celu ukrycia układu struktury

- Ważniejsze jest to, co można zrobić z danymi, niż to jak są przechowywane.
- W celu ukrycia układu struktury można wykorzystać funkcje.

Użycie funkcji w celu ukrycia układu struktury

main.c

```c
/* point_02
Program definujący strukturę point
i funkcje obsługujące tę strukturę.
Funkcje getX(), getY() pozwalają
ukryć układ struktury.
*/


#include <stdio.h>
#include <math.h>

struct point{
    double x,y;
};

typedef struct point point;

int    printPoint(point*);
double modulus(point*);
point  rotated(point*, double);
//funkcje getX() i getY() służą do odczytywania
//danych ze struktury
double getX(point*);
double getY(point*);
```

Użycie funkcji w celu ukrycia układu struktury

main.c

```
25  int main()
26  {
27      point a = {3,4}, b;
28      printf("a = ");printPoint(&a);printf("\n");
29      printf("a_x = %lf\n",getX(&a));
30      printf("a_y = %lf\n",getY(&a));
31      printf("|a| = %lf\n",modulus(&a));
32      b = rotated(&a,M_PI_2);
33      printf("b = ");printPoint(&b);printf("\n");
34      return 0;
35  }
36
```

Użycie funkcji w celu ukrycia układu struktury

main.c

```
52 double getX(point* a){
53     return a->x;
54 }
55
56 double getY(point* a){
57     return a->y;
58 }
59
```

Funkcje jako pola struktury

- Funkcje (metody) zostały zadeklarowane i zdefiniowane wewnątrz struktury – powinny być uważane za nieodłączną część tej struktury.
- Wywołanie funkcji skojarzonej ze strukturą wygląda niemal jak odczytanie pola struktury.
- Metody w strukturze mają dostęp do wszystkich pól struktury.

Funkcje jako pola struktury

main.cpp

```
1   /* point_03
2   Program w C++ definiujący strukturę point
3   i metody należące do tej struktury.
4   */
5
6   #include <iostream>   //obsługa strumieni wejścia/wyjścia
7   #undef __STRICT_ANSI__
8   #include <cmath>      //biblioteka matematyczna
9
10  using namespace std;
11
12  struct point{
13      double x,y;
14
15      //definicje metod obsługujących strukturę point
16      void print(){
17          //wypisanie współrzędnych punktu do strumienia cout
18          cout << "[" << x << "," << y << "]";
19      }
```

Funkcje jako pola struktury

main.cpp

```
21      double modulus(){
22          return sqrt(x*x + y*y);
23      }
24
25      point rotated(double alpha){
26          point b;
27          b.x = x*cos(alpha) - y*sin(alpha);
28          b.y = x*sin(alpha) + y*cos(alpha);
29          return b;
30      }
31
32      double getX(){
33          return x;
34      }
35
36      double getY(){
37          return y;
38      }
39  };
```

Funkcje jako pola struktury

main.cpp

```
41  int main()
42  {
43      //dostęp do metod obsługujących strukturę
44      //uzyskuje się analogicznie jak dostęp do
45      //pól struktury - operatorem .
46      point a = {3,4}, b;
47      cout << "a = "; a.print(); cout << endl;
48      cout << "a_x = " << a.getX() << endl;
49      cout << "a_y = " << a.getY() << endl;
50      cout << "|a| = " << a.modulus() << endl;
51      b = a.rotated(M_PI_2);
52      cout << "b = "; b.print(); cout << endl;
53      return 0;
54  }
```

Definicje funkcji poza strukturą

- Istnieje możliwość rozbicia metod na deklaracje, które znajdą się wewnątrz struktury oraz definicje umieszczone poza nią.
- Definicje metod muszą być powiązane ze swoją strukturą.
- Nazwę metody zapisuje się następująco **`nazwaStruktury::nazwaMetody()`**
- Można wydzielić kod struktury z kodu programu, utworzyć nagłówek i plik źródłowy.

Definicje funkcji poza strukturą

main.cpp

```
/* point_04
Program w C++ definiujący strukturę point
i metody obsługujące tę strukturę.
Definicje metod znajdują się poza
deklaracją struktury.
*/

#include <iostream>  //obsługa strumieni wejścia/wyjścia
#undef __STRICT_ANSI__
#include <cmath>      //biblioteka matematyczna

using namespace std;

struct point{
    double x,y;

    void print();
    double modulus();
    point rotated(double alpha);
    double getX();
    double getY();
};
```

Definicje funkcji poza strukturą

main.cpp

```
35    //definicje metod obsługujących strukturę point
36    void point::print(){
37        cout << "[" << x << "," << y << "]";
38    }
39
40    double point::modulus(){
41        return sqrt(x*x + y*y);
42    }
43
44    point point::rotated(double alpha){
45        point b;
46        b.x = x*cos(alpha) - y*sin(alpha);
47        b.y = x*sin(alpha) + y*cos(alpha);
48        return b;
49    }
50
51    double point::getX(){
52        return x;
53    }
54
55    double point::getY(){
56        return y;
57    }
```

Klasa – rozszerzenie struktury

- Klasa przypomina strukturę – jest wzbogacona o możliwość określenia, które metody i dane należą do wewnętrznej implementacji klasy, a które z metod są przeznaczone dla jej użytkowników.

Klasa - rozszerzenie struktury

Definicja

```
#ifndef POINT_H
#define POINT_H

class point
{
    public:
        void print();
        double modulus();
        point rotated(double alpha);
        double getX();
        double getY();

    private:
        double _x;
        double _y;
};

#endif // POINT_H
```

Klasa - rozszerzenie struktury

Hermetyzacja - metody get i set

```
#ifndef POINT_H
#define POINT_H

class point
{
    public:
        void print();
        double modulus();
        point rotated(double alpha);
        double getX();
        double getY();

    private:
        double _x;
        double _y;
};

#endif // POINT_H
```

Klasa - rozszerzenie struktury

Istnieją trzy podstawowe operacje, które najpewniej będzie realizować każda klasa:

1. Własna inicjalizacja.
2. Czyszczenie pamięci.
3. Kopiowanie samej siebie.

Klasa - rozszerzenie struktury

Konstruktor

```
#ifndef POINT_H
#define POINT_H

class point
{
    public:
        point(); //brak typu wartości zwracanej
        void print();
        double modulus();
        point rotated(double alpha);
        double getX();
        double getY();

    private:
        double _x;
        double _y;
};

#endif // POINT_H
```

Klasa - rozszerzenie struktury

Konstruktor

```
point::point(){
    _x = 0;
    _y = 0;
}
```

Klasa - rozszerzenie struktury

Konstruktor z parametrami

```
#ifndef POINT_H
#define POINT_H

class point
{
    public:
        point(); //brak typu wartości zwracanej
        point(double,double);
        void print();
        double modulus();
        point rotated(double alpha);
        double getX();
        double getY();

    private:
        double _x;
        double _y;
};

#endif // POINT_H
```

Klasa - rozszerzenie struktury

Konstruktor z parametrami

```
point::point(double x, double y){
    _x = x;
    _y = y;
}
```

Klasa - rozszerzenie struktury

Konstruktor z parametrami

```
int main()
{
    point a(3,4);
    point b;
    cout << "a = "; a.print(); cout << endl;
    cout << "a_x = " << a.getX() << endl;
    cout << "a_y = " << a.getY() << endl;
    cout << "|a| = " << a.modulus() << endl;
    b = a.rotated(M_PI_2);
    cout << "b = "; b.print(); cout << endl;
    return 0;
}
```

Klasa - rozszerzenie struktury

Przykład

main.cpp

main.cpp × point.h × point.cpp ×

```
1  /* point_06
2  Program w C++ demonstrujący możliwości klasy point.
3  */
4
5  #include <iostream>
6  #undef __STRICT_ANSI__
7  #include <cmath>
8  #include "point.h"
9
10 using namespace std;
11
12 int main()
13 {
14     point a(3,4);
15     point b;
16     cout << "a = "; a.print(); cout << endl;
17     cout << "a_x = " << a.getX() << endl;
18     cout << "a_y = " << a.getY() << endl;
19     cout << "|a| = " << a.modulus() << endl;
20     b = a.rotated(M_PI_2);
21     cout << "b = "; b.print(); cout << endl;
22     return 0;
23 }
```

Klasa - rozszerzenie struktury

Przykład

point.h

main.cpp | point.h | point.cpp

```
#ifndef POINT_H
#define POINT_H

class point
{
    public:
        point();
        point(double,double);
        void print();
        double modulus();
        point rotated(double alpha);
        double getX();
        double getY();

    private:
        double _x;
        double _y;
};

#endif // POINT_H
```

Klasa - rozszerzenie struktury

Przykład

point.cpp

main.cpp × point.h × point.cpp ×

```
1  #include "point.h"
2  #include <iostream>
3  #undef __STRICT_ANSI__
4  #include <cmath>
5
6  using namespace std;
7
8  point::point(){
9      _x = 0;
10     _y = 0;
11 }
12
13 point::point(double x, double y){
14     _x = x;
15     _y = y;
16 }
17
18 void point::print(){
19     cout << "[" << _x << "," << _y << "]";
20 }
```

Klasa - rozszerzenie struktury

Przykład

point.cpp

main.cpp | point.h | point.cpp

```
22  double point::modulus(){
23      return sqrt(_x*_x + _y*_y);
24  }
25
26  point point::rotated(double alpha){
27      point b;
28      b._x = _x*cos(alpha) - _y*sin(alpha);
29      b._y = _x*sin(alpha) + _y*cos(alpha);
30      return b;
31  }
32
33  double point::getX(){
34      return _x;
35  }
36
37  double point::getY(){
38      return _y;
39  }
```

Podsumowanie (2)

- Koncepcja **klasy** rozszerza strukturę o metody.
- Pojedyncza zmienną klasy nazywamy **obiektem**.
- Język C++ umożliwia **programowanie obiektowe**.

Tematy wykładów (1)

- Algorytmy i programy

- Proste typy danych
- Rozgałęzienia i iteracje
- Funkcje
- Tablice i wskaźniki
- Złożone typy danych

Tematy wykładów (2)

- Rekurencja
- Złożoność czasowa algorytmów
- Zasada dziel i zwyciężaj
- Algorytmy porządkowania
- Przeszukiwanie z nawrotami
- Poprawność i skończoność algorytmów
- Złożoność i efektywność algorytmów
- Kolokwium zaliczeniowe
- Kolokwium poprawkowe