

# Дизайн и анализ на алгоритми

## план на упражненията

КН 2.1, летен семестър 2023/2024

Калоян Цветков  
kaloyants25@gmail.com

ФМИ, СУ

1.0

## Съдържание

## Списък с определения, задачи и важни твърдения

1.1	Определение . . . . .	4
1.2	Определение – Машина с произволен достъп или RAM . . . . .	4
1.3	Определение – Елементарна операция в RAM . . . . .	4
1.4	Определение – Цена на елементарна операция в RAM . . . . .	5
1.1	Задача – <b>GCD</b> . . . . .	7
1.5	Определение – Асимптотично сравнение - $\preceq$ . . . . .	8
1.6	Определение – <i>Big O</i> . . . . .	8
1.7	Определение – <i>Big <math>\Omega</math></i> . . . . .	9
1.8	Определение – <i>Big <math>\Theta</math></i> . . . . .	9
1.2	Задача – <b>HOLE</b> . . . . .	9
2.1	Задача – <b>MAXIMUM SUBARRAY</b> . . . . .	10
2.2	Задача – <b>DISTANT PAIR</b> . . . . .	11
2.3	Задача – <b>SIEVE</b> . . . . .	12
2.7	Твърдение – техника за работа със сума от монотонна непрекъсната функция . .	13
2.4	Задача – <b>MAX BOUNDED SUBARRAY</b> . . . . .	14
2.5	Задача – Алгоритъм на <i>Boyer – Moore</i> . . . . .	14

# 1 Въведение

## 1.1 Алгоритъм

Също както за *множество*, така и за *алгоритъм* няма общоприета формална дефиниция. В този курс *алгоритъм* ще интерпретираме така:

- Алгоритъм е крайна редица от операции, която решава дадена задача.  
Разглеждаме го като реализация на тотална функция  $A : Input \mapsto Output$ , където *Input* и *Output* са крайни редици от числа и масиви.

Потенциални въпроси:

- "Защо редицата от операциите е крайна?"  
Можем да отслабим изискването за крайност и ще получим т.н. *изчислителен метод*.  
В рамките на курса ще разглеждаме единствено редици с краен брой операции.
- "Какво представлява една *операция*?"  
Зависи от *изчислителния модел*. В курса предимно ще се използва RAM моделът.  
В съответната секция ще бъдат описани позволените в RAM модела *операции*.
- "Какво представлява една *задача* и как тя се решава?"  
Става въпрос за *изчислителна задача* - понятие, което има дефиниция.  
Характеризира се със своите *екземпляри*, на всеки от които съответстват неотрицателен брой *решения*.  
Формално, за изчислителна задача може да се счита всяка релация над  $\mathbb{N}^*$ .  
Можем да мислим за *Input* и *Output* като описания на *екземпляри* и *решението* съответно (или едно от всички възможни *решения*).
- "Защо *Input* и *Output* са крайни редици и защо в тях участват само числа и масиви?"  
Тук отново може да бъде отслабено изискването за крайност, но в рамките на курса няма да разглеждаме задачи с неограничено големи *Input* и *Output*.  
Използваме числа и масиви за описание на *Input* и *Output*, понеже с тях могат да се представят математическите обекти, за които ще решаваме задачи.
- "Какво разбираме под 'числа и масиви'?"  
Числата могат да са от  $\mathbb{N}, \mathbb{Z}$  и  $\mathbb{Q}$ . При  $\mathbb{R}$  възниква въпросът за представянето, чрез апроксимация с крайна точност (тази тема може да бъде засегната по-късно).  
Масивите са крайни редици от числа или от други масиви. Множество на масивите -  $\mathbb{M}$ .  
За  $\mathbf{I} = \{I_i\}_{i=1}^n, n \in \mathbb{N}^+$  - редица от числа,  $\mathbf{I} \in \mathbb{M}$ .  
За  $\mathbf{M} = \{M_i\}_{i=1}^n, n \in \mathbb{N}^+$ , за която  $M_i \in \mathbb{M}$ ,  $\mathbf{M} \in \mathbb{M}$ .

Разговорно ще наричаме *Input* вход на алгоритъма и *Output* изход на алгоритъма. Както входът, така и изходът притежават характеристика *размер*  $\in \mathbb{N}^+$ , определен от съдържанието на този вход.

*Размерът* на число  $n \in \mathbb{N}$  се дава с  $S_{\mathbb{N}} : \mathbb{N} \mapsto \mathbb{N}^+$ :

$$S_{\mathbb{N}}(n) = \begin{cases} 1 & \text{ако } n = 0 \\ \lfloor \log_2(n) \rfloor + 1 & \text{иначе} \end{cases}$$

Естествено може да бъде продължена дефиницията за  $\mathbb{Z}, \mathbb{Q}$ .

*Размерът* на масив  $\mathbf{M} = \{M_i\}_{i=1}^m, m \in \mathbb{N}^+$  се дава с  $S_{\mathbb{M}} : \mathbb{M} \mapsto \mathbb{N}^+$  и  $S : \mathbb{N} \cup \mathbb{M} \mapsto \mathbb{N}^+$ :

$$S_{\mathbb{M}}(\mathbf{M}) = \sum_{i=1}^m S(M_i)$$

$$S(a) = \begin{cases} S_{\mathbb{N}}(a) & \text{ако } a \in \mathbb{N} \\ S_{\mathbb{M}}(a) & \text{иначе} \end{cases}$$

Размерът на редица от числа и масиви  $\{E_i\}_{i=1}^n, n \in \mathbb{N}^+$  е сумата от размерите на елементите ѝ.

**Определение 1.1** (Размер на входа). *Размер на входа  $Input$  наричаме размерът на крайната редица от числа и масиви, които  $Input$  представлява.*

## 1.2 Изчислителен модел

### 1.2.1 Въвеждане на RAM модела

Машина с произволен достъп или RAM (от английски: Random Access Machine). Това е еквивалентен модел на машините на Тюринг като изразителна способност, но е по-близък до общата представа за модерен компютър.

**Определение 1.2** (Машина с произволен достъп или RAM). *Машините с произволен достъп спадат към клас машини с непоследователен достъп до паметта (Random Access Memory или RAM памет). Всяка машина с произволен достъп се състои от памет и програма.*

Паметта на машината е разделена на две части:

- Краен брой регистри  $\gamma_0, \gamma_1, \dots, \gamma_n, n \in \mathbb{N}^+$ .
- Основна памет, която се състои от безкрайно много клетки номерирани  $0, 1, 2, \dots$

Регистри	Памет
регистър $\gamma_0$	0 <input type="text"/>
регистър $\gamma_1$	1 <input type="text"/>
регистър $\gamma_2$	2 <input type="text"/>
...	3 <input type="text"/>
регистър $\gamma_n$	...

С  $\gamma_k \in \mathbb{Z}, k \in \{0, \dots, n\}$  ще означаваме стойността, която е в регистъра  $\gamma_k$ .

С  $\rho(i) \in \mathbb{Z}, i \in \mathbb{N}$  ще означаваме стойността, която е в  $i$ -тата клетка на паметта.

Стойностите в регистрите и в паметта могат да имат произволно голям размер (тип данни *integer*).

Програма на машина с произволен достъп е крайна редица от номерирани елементарни инструкции.

**Определение 1.3** (Елементарна операция в RAM). За определеност нека означим с  $\alpha$  един от регистрите. Без ограничение на общността избираме  $\gamma_0$ . Регистъра  $\alpha$  ще наричаме акумулатор. В него се акумулира резултатът от аритметичните операции. Останалите регистри  $\gamma_1, \gamma_2, \dots, \gamma_n$  ще наричаме индексни регистри. Нека  $i, j, k \in \mathbb{N}$ . По-долу ще използваме:

- *reg*, за да означим произволен регистър от  $\alpha, \gamma_1, \dots, \gamma_n$ .
- *op*, за да означим операнд от вида  $i, \rho(i)$  или *reg*.
- *top*, за да означим модифициран операнд от вида  $\rho(i + \gamma_j)$ . Стойността на  $\rho(i + \gamma_j)$  е в клетката на паметта на позиция  $(i + \text{стойността на } \gamma_j)$ .

Елементарните операции разделяме на следните категории:

1. Операции за достъп до паметта (четене и писане):

- $reg \leftarrow op$
- $\alpha \leftarrow mop$
- $op \leftarrow reg$
- $mop \leftarrow \alpha$

2. Операции за преход (*jump*):

- $\text{goto } k$
- $\text{if } reg \pi 0 \text{ then } \text{goto } k$ , за  $\pi \in \{=, \neq, <, \leq, >, \geq\}$

3. Аритметични операции:

- $\alpha \leftarrow \alpha \pi mop$ , за  $\pi \in \{+, -, \times, div, mod\}$

4. Операции с индексните регистри:

- $\gamma_j \leftarrow \gamma_j \pm i$ ,  $1 \leq j \leq n, i \in \mathbb{N}$

Горната дефиниция на RAM позволява да се съхраняват и обработват произволно големи числа, но това не е реалистично предположение. Поради това внимателно ще дефинираме *цената* за изпълнение на елементарните операции.

*Цената* за изпълнение на елементарна операция се състои от **достъпа до паметта** и **същинската цена за изпълнение**.

Има два основни подхода за определяне на *цената*: UCM (Unit Cost Measure) и LCM (Logarithmic Cost Measure). При UCM се абстрахираме от *размера* на операндите. Това е подходящ подход, при условие че *размерът* на числата, които са в паметта по време на изпълнение на програмата е ограничен отгоре. При LCM вземаме под внимание *размера* на операндите. Използваме дефинираната нагоре функция за *размера*  $S$ .

**Определение 1.4** (Цена на елементарна операция в RAM). *Определя се според:*

- Цена за достъп до паметта според операнд:

Операнд	UCM	LCM
$i$	0	0
$reg$	0	0
$\rho(i)$	1	$S(i)$
$\rho O(i + \gamma_j)$	1	$S(i) + S(\gamma_j)$

- Същинска цена за изпълнение:

Операция	UCM	LCM
$reg \leftarrow op$	1	$1 + S(op)$
$\alpha \leftarrow mop$	1	$1 + S(mop)$
$op \leftarrow reg$	1	$1 + S(reg)$
$mop \leftarrow \alpha$	1	$1 + S(\alpha)$
$\text{goto } k$	1	$1 + S(k)$
$\text{if } reg \pi 0 \text{ then } \text{goto } k$	1	$1 + S(k)$
$\alpha \leftarrow \alpha \pi mop$	1	$1 + S(\alpha) + S(mop)$
$\gamma_j \leftarrow \gamma_j \pm i$	1	$1 + S(\gamma_j) + S(i)$

### 1.2.2 Представяне на данни в паметта

Представянето на числа в RAM модела става чрез запис в определена  $k$ -ична бройна система. Числото  $k$  е със семантиката на броя различни единици информация, които машината различава. (при машините на Тюринг  $k$  е размерът на азбуката, чиито символи пишем върху лентата).

Важна характеристика в RAM модела е т.н. размер на машинната дума: броя единици информация, които една клетка от паметта съдържа. На този етап се вижда значението на числото  $k$ . Нека размерът на машинната дума бележим с  $d$ . Тогава:

- При  $k = 1$  най-голямото число, което може да се запише в една клетка, е  $d$ . Записване на числото  $n$  в паметта изисква  $\lceil n/d \rceil$  клетки.
- При  $k > 1$  най-голямото число, което може да се запише в една клетка, е  $k^d$ . Записване на числото  $n$  в паметта изисква  $\lceil n/k^d \rceil$  клетки.

Както се вижда, разликата е експоненциална. Ние ще работим с машини, в които представянето на числата е в  $k > 1$ -ична бройна система.

### 1.2.3 Операции за константно време в RAM модела

Следните операции върху числа могат да се изпълнят за константно време в RAM модела, при условие, че операндите се побират в една машинна дума:

- $a + b, -a, a * b, a/b$  ( $b \neq 0$ )
- $a^b, \lfloor \log_b a \rfloor, a!$  (в случаите, когато  $a!$  се побира в машинната дума)
- $a \div b$  (целочислено),  $a \bmod b$
- $a = b, a < b$  (може да считаме, че това са операции, чиито резултат е 0 или 1)
- $a \vee b, \neg a, a >> b, a << b$

Разбира се, това не са всички операции. Има доста такива, които могат да се изразят като суперпозиция на изложените (например  $a \leq b = a < b \vee a = b$ ).

### 1.2.4 Псевдокод

Използването на RAM програми за описване на алгоритмите в курса ще бъде доста тромаво. Затова ще представим, чрез RAM модела някои познати програмни конструкции като if then else, for, while, псевдонимите за *променливи*. Кодът, който ще пишем и анализираме ще използва тези конструкции (и други, дефинирани, когато е уместно).

Ще пишем *псевдокод* - улеснен запис на програма в RAM модела.

- Относно псевдонимите за променливи. В известните от досегашните курсове програмни езици като C++ или Java е въведено понятието за *променлива*. В RAM модела *променливите* представляват клетките от паметта и регистрите. Вместо да ги достъпваме с  $\rho(i)$  или  $\gamma_j$  ще използваме подходящи имена (псевдоними).
- Относно if condition then option1 else option2.
- Относно for  $i = start$  to  $end$  with step do body done.
- Относно while condition do body done.

### 1.3 Първи пример

Ще разгледаме добре известния алгоритъм на *Евклид* за намиране на най-голям общ делител на две неотрицателни естествени числа.

**Задача 1.1 (GCD).**

GCD

Вход:  $A, B \in \mathbb{N}$ .

Изход:  $\text{НОД}(A, B)$ .

EUCLIDGCD(  $A, B$  : non-negative integers)

```

@1  while  $A > 0 \wedge B > 0$  do
@2    if  $A > B$  then
@3       $A \leftarrow A \bmod B$ 
@4    else
@5       $B \leftarrow B \bmod A$ 
@6    end if
@7  end while
@8  return  $\max\{A, B\}$ 

```

**Твърдение 1.1.** При вход естествени числа  $A$  и  $B$ ,  $\text{EUCLIDGCD}(A, B)$  връща тяхното най-малко общо кратно.

**Лема 1.2.** Ако  $d = \text{НОД}(A, B)$  и  $A_n$  и  $B_n$  са състоянията съответно на  $A$  и  $B$  при  $n$ -тото достигане на ред @1 на  $\text{EUCLIDGCD}$ , то  $\text{НОД}(A_n, B_n) = d$ .

Колко са операциите, които алгоритъмът извършва върху следните входове:

- $A = 0, B = 3$
- $A = 64, B = 32$
- $A = 13, B = 21$
- $A = f_n, B = f_{n+1}$  ( $f_n$  е  $n$ -тото число на Фибоначи)

EUCLIDGCDREC(  $A, B$  : non-negative integers)

```

@1  if  $A = 0 \vee B = 0$  then
@2    return  $\max\{A, B\}$ 
@3  end if
@4  if  $A > B$  then
@5    return  $\text{EuclidGCDRec}(A \bmod B, B)$ 
@6  end if
@7  return  $\text{EuclidGCDRec}(A, B \bmod A)$ 

```

**Твърдение 1.3.** При вход естествени числа  $A$  и  $B$ ,  $\text{EUCLIDGCDREC}(A, B)$  връща тяхното най-малко общо кратно.

**Лема 1.4.** За произволно естествено  $n$ , при вход  $A, B$ , такива, че  $A + B = n$  е изпълнено, че  $\text{EUCLIDGCDREC}(A, B)$  връща най-малкото общо кратно на  $A$  и  $B$ .

Рекурсивната версия **EUCLIDGCDREC** позволява съгласно намиране на числата от тъждеството на Безу:

$$A' = A - Bq, \quad B' = B$$



$$\begin{aligned}
u'A' + v'B' &= (A, B) \\
u'(A - Bq) + v'B_n &= (A, B) \\
u'A + (v' - u'q)B &= (A, B) \\
\boxed{u = u' \quad v = v' - u'q}
\end{aligned}$$

## 1.4 Коректност на алгоритъм

Искаме алгоритмите, които пишем да гарантират, че върнатата стойност за съответния *екземпляр* на изчислителната задача, да е сред множеството от възможни *решения*.

За целта доказваме свойството *коректност* за алгоритмите си.

### 1.4.1 Итеративен подход

Основава се на *изобретяването* на инварианти: твърдения за състоянието на променливите и масивите, което остава вярно през целия ход на алгоритъма.

### 1.4.2 Рекурсивен подход

При доказването на *коректност* на алгоритми, които използват рекурсия се използва метода на математическата индукция по *свойство* на входа.

## 1.5 Времева сложност

Времева сложност на алгоритъм  $\mathcal{A}$  при вход  $Input$  наричаме броя елементарни операции, които  $\mathcal{A}$  извършва върху  $Input$ , за да завърши.

Времева сложност в най-лошия случай на  $\mathcal{A}$  е функция  $Time_{\mathcal{A}}(n)$ , приемаща големина на входа  $n$  и връщаща максималния брой операции, които  $\mathcal{A}$  може да извърши върху вход с големина  $n$ .

Ще покажем, че действително входът, при който  $EuclidGCD$  извършва най-много операции, е пряко обвързан с числата на Фибоначи.

**Лема 1.5.** Ако при вход  $A$  и  $B$   $EuclidGCD$  достига ред  $\textcircled{1}$  точно  $k$  пъти, то  $\min\{A, B\} \geq f_{k-1}$ .

**Твърдение 1.6.** Времевата сложност на  $EuclidGCD$  при вход  $(A > 0, B > 0)$  е не повече от  $6 \log_{\varphi}(\min\{A, B\}) + 14$ .

(разликата спрямо упражнението е, че там е пропусната операцията присвояване на стойност)

## 1.6 Асимптотика

Имаме горна граница за времевата сложност на  $EuclidGCD$  в най-лошия случай. Такъв аргумент обаче би бил тромав за следене при по-сложни алгоритми. За целта въвеждаме следната класификация на функциите  $\mathbb{N}^+ \rightarrow \mathbb{R}$ :

**Определение 1.5** (Асимптотично сравнение -  $\preceq$ ).

$$f \preceq g \stackrel{def}{\iff} \exists N \in \mathbb{N} \exists c \in \mathbb{R}^+ \forall n (n > N \rightarrow f(n) \leq cg(n))$$

За нас стремеж ще бъде алгоритмите, които пишем да имат "възможно най-малка" относно  $\preceq$  сложност в най-лошия случай.

**Определение 1.6** (*Big O*). Класът на асимптотично не по-големите функции от  $f$  е множеството

$$\mathcal{O}(f) = \{g \mid g \preceq f\}$$

**Определение 1.7** (*Big  $\Omega$* ). Класът на асимптотично не по-малките функции от  $f$  е множеството

$$\Omega(f) = \{g \mid f \preceq g\}$$

**Определение 1.8** (*Big  $\Theta$* ). Класът на асимптотично равните функции на  $f$  е множеството

$$\Theta(f) = \mathcal{O}(f) \cap \Omega(f)$$

Под записа  $f = X(g)$ , където  $X \in \{\mathcal{O}, \Omega, \Theta\}$ , ще имаме предвид  $f \in X(g)$ .

### 1.6.1 Основни свойства

1. За всяко  $c \in \mathbb{R}^+$  е в сила, че  $cf = \Theta(f)$
2. Ако  $f = \mathcal{O}(g)$  и  $g = \mathcal{O}(h)$ , то  $f = \mathcal{O}(h)$
3. Ако  $f_1 = \mathcal{O}(g_1)$  и  $f_2 = \mathcal{O}(g_2)$ , то  $f_1 + f_2 = \mathcal{O}(\max\{g_1, g_2\})$
4. Ако  $f_1 = \mathcal{O}(g_1)$  и  $f_2 = \mathcal{O}(g_2)$ , то  $f_1 f_2 = \mathcal{O}(g_1 g_2)$
5. Ако съществува границата  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = L$ , то:
  - (а)  $L < \infty$  т.с.т.к.  $f \in \mathcal{O}(g)$
  - (б)  $0 < L < \infty$  т.с.т.к.  $f \in \Theta(g)$
  - (в) ако  $L = \infty$ , то  $f \in \Omega(g)$  (Обратното не винаги е вярно!)
6. За произволни  $a > 1$  и  $k \geq 0$  е в сила, че  $n^k = \mathcal{O}(a^n)$
7. За произволни  $a > 1$  и  $k > 0$  е в сила, че  $\log_a n = \mathcal{O}(n^k)$
8. Ако  $f = \Theta(g)$ , то  $\log_a f = \Theta(\log_a g)$

Релацията  $\preceq$  е преднаредба; не всеки две функции са сравними (пример са  $\sin(n)$  и  $\cos(n)$ ).

**Твърдение 1.7.** Времевата сложност в най-лошия случай на `EuclidGCD` при вход  $(A, B)$  е  $\mathcal{O}(\log(\min\{A, B\}))$ .

**Задача 1.2 (Hole).** Даден е масив с  $n$  различни числа от 1 до  $n + 1$ . Да се намери първото положително липсващо число в масива.

Вход:  $A[1..n]$ ,  $A[i] \leq n + 1$ ,  $1 \leq i \leq n$ . - масив от пол. естествени числа.

Изход:  $\min\{i \mid i \in \mathbb{N} \ \& \ i \notin A\}$ .

## 2 Линејни обхождания на масив

### 2.1 Сложност по памет

Сума на размерите на заделените променливи и масиви по време на работа на алгоритъма  $\mathcal{A}$  върху даден вход.

Заделянето на масив с размер  $n$  в псевдокод ще означаваме с:

SOMEALG(Input)

```

@1  ...
@2  A[1...n] ← malloc(n): ARRAY OF <TYPE>
@3  ...

```

Такова заделяне на памет извършва  $\Theta(S(A[1...n]))$  елементарни операции.

#### Задача 2.1 (MAXIMUM SUBARRAY).

Търсим инфикс на масив с максимална сума. За целта разглеждаме т.н. алгоритъм на Kadane.

Maximum Subarray

Вход:  $A[1...n]$  - масив от рационални числа.

Изход:  $\max \left\{ \sum_{k=i}^j A[k] \mid 1 \leq i \leq j \leq n \right\}$  - максимална сума на непразен инфикс на  $A$ .

KADANE(  $A[1...n]$  : array of rationals)

```

@1  maxInfix ←  $-\infty$ 
@2  maxSuffix ← 0
@3  for  $i = 1$  to  $n$  do
@4    maxSuffix ← maxSuffix +  $A[i]$ 
@5    maxInfix ←  $\max \{ \text{maxSuffix}, \text{maxInfix} \}$ 
@6    maxSuffix ←  $\max \{ 0, \text{maxSuffix} \}$ 
@7  end for
@8  return maxInfix

```

**Твърдение 2.1.** При подаден на вход масив от рационални числа  $A[1...n]$ , Kadane връща максималната сума на непразен последователен подмасив на  $A$ .

В доказателството на ?ТНМ? ?? ще използваме следната **Инварианта**:

Ако при  $k$ -тото достигане на ред @3 състоянията на  $\text{maxSuffix}$  и  $\text{maxInfix}$  са съответно  $\text{maxSuffix}_k$  и  $\text{maxInfix}_k$ , то

$$\text{maxInfix}_k = \max \left\{ \sum_{t=i}^j A[t] \mid 1 \leq i \leq j \leq k-1 \right\} \text{ и}$$

$$\text{maxSuffix}_k = \max \left\{ \sum_{t=i}^k A[t] \mid 1 \leq i \leq k \right\}$$

**Твърдение 2.2.** Времевата сложност на KADANE в най-лошия случай е  $\Theta(n)$ .

Друго възможно линейно решение използва т.н. *префиксни суми* и се опира на следните наблюдения:

1. Всяка инфиксна сума е разлика на 2 префиксни:

$$\sum_{k=i}^j A[k] = \sum_{k=1}^j A[k] - \sum_{k=1}^{i-1} A[k]$$

2. Тогава най-голямата инфиксна сума се намира лесно чрез:

$$\max_{1 \leq i \leq j \leq n} \left\{ \sum_{t=i}^j A[t] \right\} = \max_{1 \leq i \leq j \leq n} \left\{ \sum_{t=1}^j A[t] - \sum_{t=1}^{i-1} A[t] \right\} = \max_{1 \leq j \leq n} \left\{ \sum_{t=1}^j A[t] - \min_{1 \leq i \leq j} \sum_{t=1}^{i-1} A[t] \right\}$$

### Задача 2.2 (Distant Pair).

Дадени са  $n$  точки в равнината. Да се намери най-голямото манхатънско разстояние между 2 точки.

Точка ще представяме чрез записа:

$$\text{Point} = \{ \\ \quad x : \text{rational} \\ \quad y : \text{rational} \\ \}$$

Манхатънското разстояние между записи от тип *Point*  $P1$  и  $P2$  ще бележим с  $d_M(P1, P2) = |P1.x - P2.x| + |P1.y - P2.y|$

Вход:  $P[1 \dots n]$  - масив от *Points*.

Изход:  $\max \{d_M(P[i], P[j]) \mid 1 \leq i, j \leq n\}$ .

DISTANT( $P[1 \dots n]$  : array of Points)

```

01  maxSum ← -∞
02  minSum ← +∞
03  maxDiff ← -∞
04  minDiff ← +∞
05  for i = 1 to n do
06    maxSum ← max {P[i].x + P[i].y, maxSum}
07    minSum ← min {P[i].x + P[i].y, minSum}
08    maxDiff ← max {P[i].x - P[i].y, maxDiff}
09    minDiff ← min {P[i].x - P[i].y, minDiff}
10  end for
11  return max {maxSum - minSum, maxDiff - minDiff}

```

### Лема 2.3.

$$\max_{1 \leq i, j \leq n} \{d_M(P[i], P[j])\} = \max \left\{ \max_{1 \leq i, j \leq n} \{(P[i].x + P[i].y) - (P[j].x + P[j].y)\}, \right. \\ \left. \max_{1 \leq i, j \leq n} \{(P[i].x - P[i].y) - (P[j].x - P[j].y)\} \right\}$$

**Твърдение 2.4.** При вход масив  $P[1 \dots n]$ , масив от записи *Point*, DISTANT връща най-голямото манхатънско разстояние между някои две от точките за време  $\Theta(n)$ .

**Задача 2.3 (SIEVE).** Да се намерят простите числа  $\leq n$  (решето на Ератостен)

Вход:  $n$  - положително естествено число

Изход:  $P[1...k] = \{p \mid p \leq n \text{ \& } p \text{ - просто}\}$ .

SIEVE( $n$  : positive integer)

```

01  A[1...n] ← malloc(n): ARRAY OF BOOLEANS
02  for i = 2 to n do
03    A[i] ← true
04  end for
05  A[1] ← false
06  for i = 2 to n do
07    if A[i] = true then
08      Eliminate(i, A[2...n])
09    end if
10  end for
11  P[1...k] ← ArgTrue(A[1...n])
12  return P[1...k]

```

ELIMINATE( $p$  : prime, A[1...n] : booleans)

```

01  j ← 2 * p
02  while j ≤ n do
03    A[j] ← false
04    j ← j + p
05  end while

```

COUNTTRUE(A[1...n] : array of booleans)

```

01  k ← 0
02  for i = 1 to n do
03    if A[i] = true then
04      k ← k + 1
05    end if
06  end for
07  return k

```

ARGTRUE(A[1...n] : array of booleans)

```

01  k ← CountTrue(A[2...n])
02  P[1...k] ← malloc(k): ARRAY OF INTEGERS
03  k ← 1
04  for i = 2 to n do
05    if A[i] = true then
06      P[k] ← i
07      k ← k + 1
08    end if
09  end for
10  return P[1...k]

```

**Твърдение 2.5.** При вход просто число  $p$  и масив от булеви стойности  $A[1...n]$ , ELIMINATE модифицира  $A[1...n]$  до  $A'[1...n]$ , където за  $1 \leq i \leq n$ :

$$A'[i] = \begin{cases} false & , \text{ ако } p|i \\ A[i] & , \text{ иначе} \end{cases}$$

Още нещо, ELIMINATE завършва за време  $\Theta\left(\frac{n}{p}\right)$ .

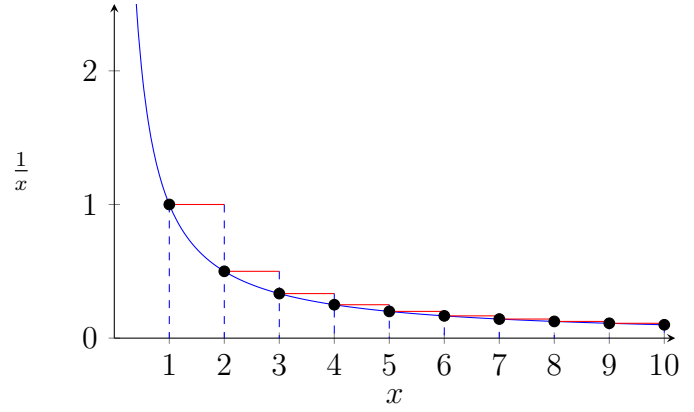
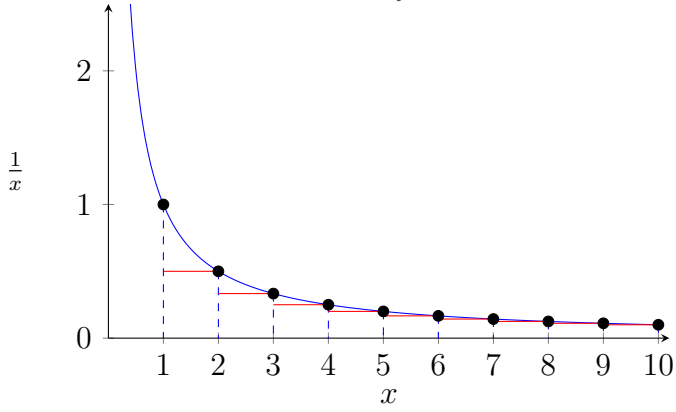
**Твърдение 2.6.** При вход положително цяло число  $n$ , SIEVE връща масив, съдържащ простите числа  $p \leq n$ . Още нещо, SIEVE завършва за  $\mathcal{O}(n \log n)$  време.

За доказателството на ?THM? ??:

1. Инварианта: При всяко достигане на 07 на SIEVE, за всяко  $1 \leq l \leq i$  е в сила, че  $A[l] = true$  т.с.т.к.  $l$  е просто.  
Вярността на тази инварианта съществено ползва ?THM? ??.
2. COUNTTRUE и ARGTRUE работят за време  $\Theta(n)$  и връщат съответно броя и масив от простите числа  $p \leq n$
3. SIEVE работи в най-лошия случай за време

$$\mathcal{O}(n) + \sum_{i=2, i \text{ - просто}}^n Time_{Eliminate}(p, A[1...n]) + \Theta(n) + \Theta(n) = \dots = \mathcal{O}(n \log n)$$

Идея за доказване на  $\sum_{i=2}^n \frac{1}{i} = \Theta(\log n)$ :



**Твърдение 2.7** (техника за работа със сума от монотонна непрекъсната функция). Нека  $f : \mathbb{N}^+ \rightarrow \mathbb{R}^+$  монотонна непрекъсната функция и нека

$$S = \sum_{i=1}^n f(i)$$

$$I = \int_1^n f(x) dx$$

Тогава:

- ако  $f$  е растяща, то  $I + f(1) \leq S \leq I + f(n)$ .
- ако  $f$  е намаляваща, то  $I + f(n) \leq S \leq I + f(1)$ .

( $ne^{n^2}$  е пример за неподходяща употреба)

( $\mathcal{O}(n \log(\log n))$ ) е по-добра оценка)

Използваме наготово, че  $\pi(k) = |\{p \mid p \leq k \text{ \& } p \text{ е просто}\}| = \frac{k}{\log(k)} \left(1 + \mathcal{O}\left(\frac{1}{\log(k)}\right)\right)$ ,

$$\begin{aligned} \sum_{p \leq n} \frac{1}{p} &= \sum_{k=1}^n \frac{\pi(k) - \pi(k-1)}{k} = \sum_{k=1}^n \frac{\pi(k)}{k} - \sum_{k=0}^{n-1} \frac{\pi(k)}{k+1} \\ &= \frac{\pi(n)}{n} + \sum_{k=1}^{n-1} \frac{\pi(k)}{k(k+1)} = \mathcal{O}(1) + \sum_{k=1}^{n-1} \frac{\pi(k)}{k^2} - \sum_{k=1}^{n-1} \frac{\pi(k)}{k^2(k+1)} \\ &= \sum_{k=3}^{n-1} \frac{\pi(k)}{k^2} + \mathcal{O}(1) = \sum_{k=3}^{n-1} \left[ \frac{1}{k \log(k)} + \mathcal{O}\left(\frac{1}{k \log(k)^2}\right) \right] + \mathcal{O}(1) \\ &= \log(\log(n)) + \mathcal{O}(1) \end{aligned}$$

Инициализацията на @7 задава ненужно малка стойност за  $j$ . Оптимално е тя да бъде  $i * i$ , защото съставните числа по-малки от  $i * i$  имат прост делител  $< i$ , тоест вече са отпаднали като кандидати за прости числа ( $A[k] = false$ ). Това е мотивация за "подобрен" алгоритъм (асимптотично нещата не се променят):

ELIMINATE2( $p$  : prime,  $A[1..n]$  : booleans)

```
@1  j ← p * p
@2  while j ≤ n do
@3    A[j] ← false
@4  j ← j + p
```

05 end while

**Задача 2.4 (MAX BOUNDED SUBARRAY).**

По даден масив  $A$  и рационално число  $S$ , търсим максималната сума на подмасив, което не надвишава  $S$ .

Да се предложи алгоритъм с времева сложност  $\Theta(n)$ , който решава следния проблем:

Вход:  $A[1 \dots n]$  - масив от рационални числа,  $S$  - рационално число.

Изход:  $\max \left\{ \sum_{k=i}^j A[k] \mid 1 \leq i, j \leq n \ \& \ \sum_{k=i}^j A[k] \leq S \right\}$ .

**Задача 2.5 (Алгоритъм на Boyer – Moore).**

Изборните резултати са представени с масив от вотове  $V[1 \dots n]$ , като  $V[i]$  е подаден вот за кандидат  $i$  (броят на кандидатите не ни е известен). Изборите се печелят от кандидат, когато гласовете за него са над 50%, т.е. поне  $\lfloor \frac{n}{2} \rfloor + 1$ .

Да се предложи алгоритъм с времева сложност  $\Theta(n)$ , който решава следния проблем:

Вход:  $V[1 \dots n]$  - масив от вотове (положителни естествени числа).

Изход:  $i$  - победител в изборите, ако има такъв, в противен случай - 0.

Следва продължение...