Дизайн и анализ на алгоритми

план на упражненията

КН 2.1, летен семестър 2023/2024

Калоян Цветков
kaloyants25@gmail.com

ФМИ, СУ

1.1

Съдържание

Списък с определения, задачи, важни твърдения и допълнения

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 $\mathbb{N}^\star$.
  Можем да мислим за *Input* и *Output* като описания на *екземпляра* 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* $\gamma_0, \gamma_1, \ldots, \gamma_n$, $n \in \mathbb{N}^+$.
- *Основна памет, която се състои от безкрайно много клетки номерирани* $0, 1, 2, \ldots$

Регистри	*Памет*
регистър γ_0	0
регистър γ_1	1
регистър γ_2	2
$\ldots$	3
регистър γ_n	$\ldots$

С $\gamma_k \in \mathbb{Z}, k \in \{0. \ldots, n\}$ *ще означаваме стойността, която е в регистъра* γ_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* α *един от регистрите. Без ограничение на общността избираме* γ_0. *Регистъра* α *ще наричаме акумулатор. В него се акумулира резултатът от аритметичните операции. Останалите регистри* $\gamma_1, \gamma_2, \ldots, \gamma_n$ *ще наричаме индексни регистри. Нека* $i, j, k \in \mathbb{N}$. *По-долу ще използваме:*

- *reg, за да означим произволен регистър от* $\alpha, \gamma_1, \ldots, \gamma_n$.
- *op, за да означим операнд от вида* i, $\rho(i)$ *или reg.*
- *mop, за да означим модифициран операнд от вида* $\rho(i + \gamma_j)$. *Стойността на* $\rho(i + \gamma_j)$ *е в клетката на паметта на позиция* $(i +$ *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):*
   - $\underline{goto}\ k$
   - $\underline{if}\ reg\ \pi\ 0\ \underline{then}\ \underline{goto}\ k$, *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 взимаме 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*	$S(i)$
$\rho(i + \gamma_j)$	*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*	$1 + S(\alpha)$
$\underline{goto}\ k$	*1*	$1 + S(k)$
$\underline{if}\ reg\ \pi\ 0\ \underline{then}\ \underline{goto}\ k$	*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 е 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), $a \mod b$
- $a = b$, $a < b$ (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 γ_j ще използваме подходящи имена (псевдоними).
- Относно if *condition* then *option*1 else *option*2.
- Относто for $i = start$ *to* end *with* $step$ do *body* done.
- Относто 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).
Вход: A,B $\in \mathbb{N}$.
Изход: НОД(A,B).

EuclidGCD(A, B : non-negative integers)

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

Твърдение 1.1. *При вход естествени числа A и B,* EuclidGCD(A, B) *връща тяхното най-малко общо кратно.*

Лема 1.2. *Ако d =НОД(A,B) и A_n и B_n са състоянията съответно на A и B при n-тото достигане на ред @1 на* EuclidGCD, *то 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 EuclidGCDRec$(A \; mod \; B, B)$
@6 end if
@7 return EuclidGCDRec$(A, B \; mod \; A)$

Твърдение 1.3. *При вход естествени числа A и B,* 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* EuclidGCDRec(A, B) *връща най-малкото общо кратно на A и B.*

Рекурсивната версия **EuclidGCDRec** позволява сръчно намиране на числата от тъждеството на Безу:

$$A' = A - Bq, \quad B' = B$$
$$u'A' + v'B' = (A, B)$$
$$u'(A - Bq) + v'B_n = (A, B)$$

$$u'A + (v' - u'q)B = (A, B)$$
$$\boxed{u = u' \quad v = v' - u'q}$$

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* `@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 $\mathbb{N}^+ \to \mathbb{R}$:

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

$$f \preceq g \overset{def}{\longleftrightarrow} \exists N \in \mathbb{N} \exists c \in \mathbb{R}^+ \forall n \, (n > N \to f(n) \leq cg(n))$$

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* Ω)**.** *Класът на асимптотично не по-малките функции от f е множеството*

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

Определение 1.8 (*Big* Θ)**.** *Класът на асимптотично равните функции на* f *е множеството*

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

Под записа $f = X(g)$, където $X \in \{O, \Omega, \Theta\}$, 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 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$, 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 Линейни 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.

Вход: `A[1...n] - масив от рационални числа.`
Изход: $\max\left\{\sum_{k=i}^{j} A[k] \mid 1 \le i \le j \le n\right\}$ *- максимална сума на непразен инфикс на* A.

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

```
@1  maxInfix ← −∞
@2  maxSuffix ← 0
@3  for i = 1 to n do
@4   maxSuffix ← maxSuffix + A[i]
@5   maxInfix ← max{maxSuffix, maxInfix}
@6   maxSuffix ← max{0, 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 **Твърдение** 2.1 ще използваме следната **Инвариант**:

Ако при k-тото достигане на ред `@3` състоянията на $maxSuffix$ и $maxInfix$ са съответно $maxSuffix_k$ и $maxInfix_k$, то

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

$$maxSuffix_k = \max\left\{\sum_{t=i}^{k} A[t] \mid 1 \le i \le k\right\}$$

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:

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

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

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

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:

```
Point = {
  x : rational
  y : 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...n] - масив от Points.`
Изход: $\max\{d_M(P[i],P[j])\mid 1\le i,j\le n\}$.

```
DISTANT( P[1...n] : array of Points)
@1  maxSum ← −∞
@2  minSum ← +∞
@3  maxDiff ← −∞
@4  minDiff ← +∞
@5  for i = 1 to n do
@6   maxSum ← max{P[i].x + P[i].y, maxSum}
@7   minSum ← min{P[i].x + P[i].y, minSum}
@8   maxDiff ← max{P[i].x − P[i].y, maxDiff}
@9   minDiff ← min{P[i].x − P[i].y, minDiff}
@10 end for
@11 return  max{maxSum − minSum, maxDiff − minDiff}
```

Лема 2.3.

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

Твърдение 2.4. *При вход масив $P[1...n]$, 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: $P[1...k]$=$\{p \mid p \leq n \,\&\, p$ `- просто`$\}$.

```
Sieve( n : positive integer)
@1  A[1...n] ← malloc(n): ARRAY OF BOOLEANS

@2  for i = 2 to n do
@3   A[i] ← true
@4  end for
@5  A[1] ← false
@6  for i = 2 to n do
@7   if A[i] = true then
@8    Eliminate(i, A[2...n])
@9   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)

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

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

@1  k ← 0
@2  for i = 1 to n do
@3   if A[i] = true then
@4    k ← k + 1
@5   end if
@6  end for
@7  return  k
```

```
ArgTrue( A[1...n] : array of booleans)
@1  k ← CountTrue(A[2...n])
@2  P[1...k] ← malloc(k): ARRAY OF INTEGERS

@3  k ← 1
@4  for i = 2 to n do
@5   if A[i] = true then
@6    P[k] ← i
@7    k ← k + 1
@8   end if
@9  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 ***Твърдение*** 2.6:

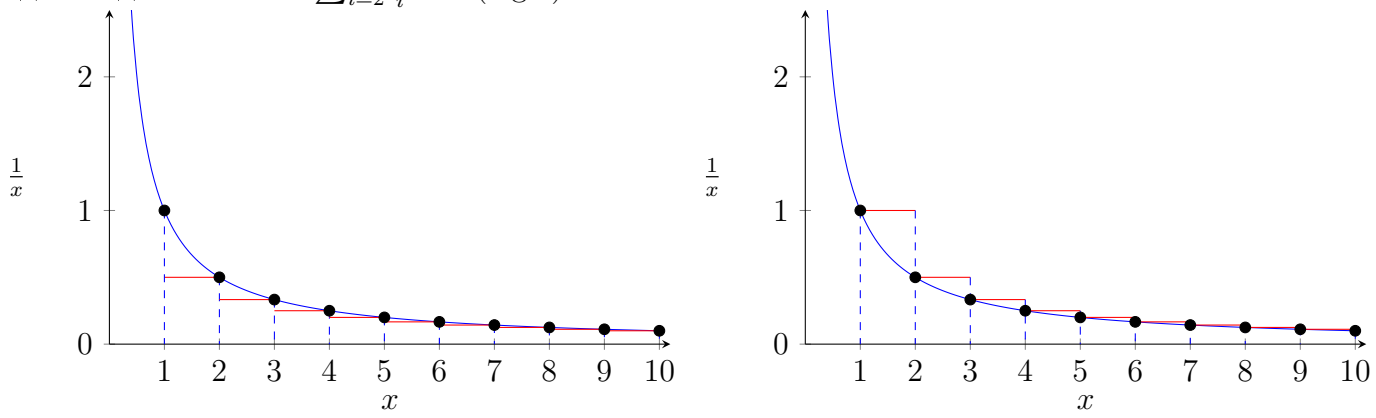
1. *Инвариантa: При всяко достигане на* `@7` *на* Sieve, *за всяко* $1 \leq l \leq i$ *е в сила, че* $A[l] = true$ *т.с.т.к.* l *е просто.*
   Вярността на тази инвариантa съществено ползва ***Твърдение*** 2.5.

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*

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

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* $f : \mathbb{N}^+ \to \mathbb{R}^+$ *монотонна непрекъсната функция и нека*

$$S = \sum_{i=1}^{n} f(i) \qquad I = \int_1^n f(x)dx$$

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)*

Допълнение 2.1 ($(\mathcal{O}(n \log(\log n))$ е по-добра оценка за SIEVE)**.**
Използваме наготово, че $\pi(k) = |\{p \mid p \leq k \ \& \ p \text{ е просто}\}| = \frac{k}{\log(k)} \left(1 + \mathcal{O}\left(\frac{1}{\log(k)}\right)\right)$,

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

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
@5  end while
```

Задача 2.4 (**Longest Unique Subarray**). *Елементите на масив са естествени числа, по-малки от n. Търсим най-дългия подмасив, който не съдържа повтарящи се елементи.*

Вход: $A[1...n]$ - *масив от естествени числа,* $A[i] \leq n$ за $1 \leq i \leq n$
Изход: $\max\left\{j - i + 1 \mid 1 \leq i \leq j \leq n \;\&\; \{A[k]\}_{k=i}^{j}\right.$ *не съдържа повтарящи се елементи* $\left.\right\}$.

Задача 2.5 (Алгоритъм на $Boyer - Moore$).
Изборните резултати са представени с масив от вотове $V[1...n]$, 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...n] - *масив от вотове (положителни естествени числа).*
Изход: i - *победител в изборите, ако има такъв, в противен случай* - 0.

3 Сортиране (упражнения 3 и 4)

Цел: подредба на елементите "по големина" (т.е. по някакво тяхно свойство)
Алгоритми за сортиране:

- $\mathcal{O}(n^2)$ в най-лошия случай
  - SelectionSort
  - InsertionSort
  - QuickSort (при случаен *pivot*, средна сложност $\mathcal{O}(n \log n)$)
- $\mathcal{O}(n \log n)$
  - HeapSort
  - MergeSort
  - QuickSort (при *pivot*, определен като "Median of medians")
- $\mathcal{O}(n + d) \longrightarrow$ CountingSort (използваме, когато имаме горна граница d за елементите на масива)

Задача 3.1 (**2-Sum**).

Вход: $A[1...n]$ - *масив от рационални числа.*
Изход: Има ли $1 \leq i < j \leq n$, *такива, че* $A[i] + A[j] = 0$?

TwoPointerSum($A[l...r]$: sorted array of rationals, $target$: rational number)

```
@1  i ← l
@2  j ← r
@3  while i < j ∧ A[i] + A[j] ≠ target do
@4   if A[i] + A[j] < target then
@5    i ← i + 1
@6   else
@7    j ← j − 1
@8   end if
@9  end while
@10 return  i < j
```

Твърдение 3.1. *При вход сортиран масив* $A[1...n]$ *и рационално число target,* TwoPointerSum *връща истина т.с.т.к. има* $1 \leq i < j \leq n$, *такива, че* $A[i] + A[j] = target$.

Още нещо, TwoPointerSum *работи върху всеки свой вход за време* $\Theta(n)$.

2Sum($A[1...n]$: array of rationals)

```
@1  MergeSort(A[1...n])
@2  return  TwoPointerSum(A[1...n], 0)
```

Коректността на 2Sum е директно следствие от **Твърдение** 3.1, тъй като подаденият на @2 масив е сортираният вход и е в сила, че такива индекси $1 \leq i, j \leq n$ има в A т.с.т.к. има такива индекси $1 \leq i', j' \leq n$ в сортираната пермутация A' на A.

Задача 3.2 (**3-Sum**).

Вход: $A[1...n]$ - *масив от рационални числа.*
Изход: Има ли $1 \leq i < j < k \leq n$, *такива, че* $A[i] + A[j] + A[k] = 0$?

3Sum($A[1...n]$: array of rationals)

```
@1  MergeSort(A[1...n])
@2  i ← 1
@3  while i < n − 1 ∧ ¬TwoPointerSum(A[i + 1...n], −A[i]) do
@4   i ← i + 1
@5  end while
@6  return  i < n − 1
```

Отворен е въпросът дали има $\mathcal{O}(n^{2-\epsilon})$ алгоритъм, решаващ **3-Sum**.

Задача 3.3 (**4-Sum**).

Вход: $A[1...n]$ - *масив от рационални числа.*
Изход: Има ли $1 \leq i < j < k < l \leq n$, *такива, че* $A[i] + A[j] + A[k] + A[l] = 0$*?*

$$\begin{aligned}&\texttt{Sum} = \{\\ &\quad \texttt{left} : \texttt{positive integer}\\ &\quad \texttt{right} : \texttt{positive integer}\\ &\quad \texttt{value} : \texttt{rational}\\ &\}\end{aligned}$$

Нека MergeSortSums е модификация на MergeSort, която сортира масив от Sums по контекст $s1 \prec s2$ т.с.т.к. $s1.value < s2.value \vee (s1.value = s2.value \wedge (s1.left < s2.left \vee (s1.left = s2.left \wedge s1.right < s2.right)))$.

Нека TwoPointerSumSums е модификация на TwoPointerSum, която:

- на вход получава масив от Sums и *target*
- на @3 заменя $A[i] + A[j] \neq target$ с $A[i].value + A[j].value \neq target \vee A[i].left = A[j].left \vee A[i].right = A[j].right$
- на @4 заменя $A[i] + A[j] < target$ с $A[i].value + A[j].value < target$

4Sum($A[1...n]$: array of rationals)

```
@1  B[1...(n 2)] ← malloc((n 2)): ARRAY OF Sums
@2  k ← 1
@3  for i = 1 to n − 1 do
@4   for j = i + 1 to n do
@5    B[k] ← Sum(left ← i, right ← j, value ← A[i] + A[j])
@6    k ← k + 1
@7   end for
@8  end for
@9  MergeSortSums(B[1...(n 2)])
@10 TwoPointerSumSums(B[1...(n 2)], 0)
```

Времева сложност на 4Sum: $\mathcal{O}(\binom{n}{2}) + \mathcal{O}(\binom{n}{2}) + \mathcal{O}(\binom{n}{2} \log \binom{n}{2}) + \mathcal{O}(\binom{n}{2}) = \mathcal{O}(n^2 \log n)$.

Задача 3.4 (**Minimum Absolute Subarray**). *Да се намери инфиксна сума в масив с най-малка абсолютна стойност.*
(Стока в магазин, да няма недостиг, но и да няма излишък; в кой период е постигнат оптимален резултат?)

Вход: $A[1...n]$ - *масив от рационални числа.*
Изход: $\min\left\{\left|\sum_{k=i}^{j} A[k]\right| \mid 1 \leq i \leq j \leq n\right\}$ - *най-близка до* 0 *сума на непразен инфикс на* A.

Лема 3.2. *Ако за* $k < l$ *имаме, че*

$$|S_k - S_l| = \min\{|S_j - S_i| \mid 0 \leq i < j \leq n\}$$

то няма q*, такова, че* $S_k < S_q < S_l$ *или* $S_l < S_q < S_k$.

Сега след **Лема** 3.2 остава да съобразим, че е достатъчно да сортираме префиксните суми и да търсим минимална абсолютна разлика между някои два последователни.

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

```
@1  B[0...n] ← malloc(n+1): ARRAY OF rationals
@2  B[0] ← 0
@3  for i = 1 to n do
@4   B[i] ← B[i−1] + A[i]
@5  end for
@6  MergeSort(B[0...n])
@7  m ← +∞
@8  for i = 1 to n do
@9   m ← min{m, B[i] − B[i−1]}
@10 end for
@11 return m
```

Твърдение 3.3. *При вход* $A[1...n]$ *- масив от рационални числа,* MinAbsSubarray *работи за време* $\mathcal{O}(n \log n)$ *и връща най-малката абсолютна стойност на сума на елементите на непразен подмасив на* A.

Задача 3.5 (**Triangles**).

Вход: $A[1...n]$ - *масив от рационални числа.*
Изход: $|\{\langle i,j,k\rangle \mid 1 \leq i < j < k \leq n \;\&\; A[i], A[j], A[k]$ *са страни на триъгълник*$\}|$

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

```
@1  MergeSort(A[1...n])
@2  count ← 0
@3  for i = 1 to n − 2 do
@4   k ← i + 2
@5   for j = i + 1 to n − 1 do
@6    while k < n ∧ A[i] + A[j] > A[k] do
@7     k ← k + 1
@8    end while
@9    count ← count + (k − j − 1)
@10  end for
@11 end for
@12 return count
```

Коректност на TRIANGLES може да се покаже с инварианта за цикъла на @3, като за нейното доказване е удобно в стъпката да се дефинира друга инварианта за цикъла на @5.

Относно времевата сложност на TRIANGLES, ключово наблюдение е @9 се изпълнява най-много $n-i-2$ пъти за всяко изпълнение на тялото на цикъла на ред @3...

$$Time_{Triangles} = \sum_{i=1}^{n-2} (\mathcal{O}(n-i-2) + \mathcal{O}(n-i-2)) = ... = \mathcal{O}(n^2)$$

Задача 3.6. *Разглеждаме редица от $2n$ рационални числа $\{a_i\}_{i=1}^{2n}$. Групиране на $\{a_i\}_{i=1}^{2n}$ ще наричаме множество от наредени двойки (a_p, a_q), такова, че всеки член a_i на редицата участва точно в една наредена двойка и то веднъж. Тежест на групиране P ще наричаме*

$$w(P) = \max\{a_p + a_q \mid (a_p, a_q) \in P\}$$

. Целим да намерим минималната възможна тежест измежду всички групирания.

Вход: $A[1...2n]$ - *представяне на редица от рационални числа.*
Изход: $m = \min\{w(P) \mid P$ *е групиране за* $A[1...2n]\}$.

Лема 3.4. *Минимална тежест на растяща редица се достига за групирането* $\{(a_i, a_{2n-i+1}) \mid 1 \le i \le n\}$.

Очевидно пермутирането на входа не променя всички възможни групирания, а следователно и това с най-малка тежест не се променя.
След придобитата интуиция за отговора следва само да се възползваме от сортирането:

LIGHTESTGROUP($A[1...2n]$: array of rationals)

```
@1  HeapSort(A[1...2n])
@2  maxSum ← −∞
@3  for i = 1 to n do
@4   maxSum ← max{maxSum, A[i] + A[2n − i + 1]}
@5  end for
@6  return  maxSum
```

Времева сложност: $\mathcal{O}(n \log n)$.

Оказва се, че това групиране решава и следните подобни проблеми:

1. Лекота на групиране P ще наричаме

   $$l(P) = \min\{a_p + a_q \mid (a_p, a_q) \in P\}$$

   Вход: $A[1...2n]$ - представяне на редица от рационални числа.
   Изход: $m = \max\{l(P) \mid P$ е групиране за $A[1...2n]\}$.

2. Потенциал на групиране P ще наричаме

   $$\pi(P) = w(P) - l(P)$$

   Вход: $A[1...2n]$ - представяне на редица от рационални числа.
   Изход: $m = \min\{\pi(P) \mid P$ е групиране за $A[1...2n]\}$.

Задача 3.7 (Задача 1, писмен изпит 2023г.).

Представяне на редица от n интервала I_1,...,I_n ще наричаме двойка от масиви $L[1..n]$ и $R[1..n]$, за които:

$$I_i = [L[i], R[i]] \text{ за всяко } i \in \{1, 2, ..., n\}$$

1. *Разглеждаме следния проблем* **IntersectingPairs**:

   Вход: $L[1..n]$, $R[1..n]$ *представяне на редица от* n *интервала* $I_1, I_2, ..., I_n$
   Изход: $N = |\{(i,j) \mid i < j \text{ и } I_i \cap I_j \neq \emptyset\}|$

   Да се предложи алгоритъм със сложност $\mathcal{O}(n \log n)$, който решава проблема **IntersectingPairs**.
   Да се докаже коректността и времевата сложност на предложения алгоритъм.

2. *За редица от интервали $I = (I_1, I_2, ..., I_n)$ с $m(I)$ означаваме най-големия брой интервали от редицата, които в съвкупност имат непразно сечение.*
   Разглеждаме следния проблем **MaxIntersection**:

   Вход: $L[1..n]$, $R[1..n]$ *представяне на редица от* n *интервала* $I_1, I_2, ..., I_n$
   Изход: $m(I)$

   Да се предложи алгоритъм със сложност $\mathcal{O}(n \log n)$, който решава проблема **MaxIntersection**.
   Да се докаже коректността и времевата сложност на предложения алгоритъм.

Удобно е използването на запис

```
Endpoint = {
  x : rational
  closing : boolean
}
```

Забелязваме, че сортиране на тези записи лексикографски ($false$ считаме за "по-малко" от $true$, за да "засечем" едноточкови пресичания) би помогнало за решаване и на двата проблема за линейно време след това. За целта нека MergeSortEndpoints е модификация на Mergesort, получаваща на вход масив от Endpoints и връщаща негова сортирана пермутация по контекст $e1 \preceq e2$ т.с.т.к. $e1.x < e2.x \vee (e1.x = e2.x \wedge e1.closing = false)$.

1. IntersectingPairs:

```
IntersectingPairs( L[1...n] : left ends,  R[1...n] : right ends)
@1  A[1...2*n] ← malloc(2*n): ARRAY of Endpoints
@2  for i = 1 to n do
@3   A[2*i − 1] ← Endpoint(x ← L[i], closing ← false)
@4   A[2*i] ← Endpoint(x ← R[i], closing ← true)
@5  end for
@6  MergeSortEndpoints(A[1...2n])
@7  active ← 0
@8  intersections ← 0
@9  for i = 1 to 2*n do
@10  if A[i].closing = true then
@11   active ← active − 1
@12  else
@13   intersections ← intersections + active
@14   active ← active + 1
@15  end if
```

```
@16  end for
@17  return  intersections
```

Твърдение 3.5. *При вход представяне на интервали $I_1, ..., I_n$ с $L[1...n], R[1...n]$,* IntersectingPairs *връща броя на двойките различни интервали, които се пресичат.*

Аргументация:
Нека полетата `x` в сортирания на `@6` $A'[1...2n]$ са

$$\underbrace{x_{1,1}, x_{1,2}, ..., x_{1,c_1}}_{c_1}, \underbrace{x_{2,1}, x_{2,2}, ..., x_{2,c_2}}_{c_2}, ..., \underbrace{x_{t,1}, x_{t,2}, ..., x_{t,c_t}}_{c_t}$$

където $x_{i,p} = x_{j,q}$ т.с.т.к. $i = j$.
Тогава, ако $active_k$ и $intersections_k$ са състоянията на $active$ и $intersections$ непосредствено преди обработването на $x_{k,1}$ (при $1 + c_1 + c_2 + ... + c_{k-1}$-вото достигане на `@9`), то

$$active_k = |\{I_i \mid L[i] < x_{k,1} \ \& \ R[i] \geq x_{k,1}\}|$$

$$intersections_k = |\{(i,j) \mid L[i] < x_{k,1} \ \& \ L[j] < x_{k,1} \ \& \ I_i \cap I_j \neq \emptyset\}|$$

Относно времевата сложност:

$$Time_{IntersectingPairs} = O(2n) + O(n) + O(2n\log(2n)) + O(2n) = O(n\log n)$$

2. MaxIntersection:

MaxIntersection($L[1...n]$: left ends, $R[1...n]$: right ends)

```
@1   A[1...2*n] ← malloc(2*n): ARRAY of Endpoints
@2   for i = 1 to n do
@3    A[2*i − 1] ← Endpoint(x ← L[i], closing ← false)
@4    A[2*i] ← Endpoint(x ← R[i], closing ← true)
@5   end for
@6   MergeSortEndpoints(A[1...2n])
@7   active ← 0
@8   maxIntersecting ← −∞
@9   for i = 1 to 2*n do
@10   if A[i].closing = true then
@11    active ← active − 1
@12   else
@13    active ← active + 1
@14    maxIntersecting ← max{maxIntersecting, active}
@15   end if
@16  end for
@17  return  maxIntersecting
```

Твърдение 3.6. *При вход представяне на интервали $I_1, ..., I_n$ с $L[1...n], R[1...n]$,* MaxIntersection *връща максималния брой интервали с непразно сечение в съвкупност.*

Аргументация:
Нека полетата `x` в сортирания на `@6` $A'[1...2n]$ са

$$\underbrace{x_{1,1}, x_{1,2}, ..., x_{1,c_1}}_{c_1}, \underbrace{x_{2,1}, x_{2,2}, ..., x_{2,c_2}}_{c_2}, ..., \underbrace{x_{t,1}, x_{t,2}, ..., x_{t,c_t}}_{c_t}$$

където $x_{i,p} = x_{j,q}$ т.с.т.к. $i = j$.
Тогава, ако $active_k$ и $intersections_k$ са състоянията на *active* и *intersections* непосредствено преди обработването на $x_{k,1}$ (при $1 + c_1 + c_2 + ... + c_{k-1}$-вото достигане на `@9`), то

$$active_k = |\{I_i \mid L[i] < x_{k,1} \ \& \ R[i] \geq x_{k,1}\}|$$

$$maxIntersecting_k = m(\{I_i \mid L[i] < x_{k,1}\})$$

Относно времевата сложност:

$$Time_{MaxIntersection} = O(2n) + O(n) + O(2n \log(2n)) + O(2n) = O(n \log n)$$

Определение 3.1 (Редукция на изчислителни задачи)**.**
Казваме, че **PR1** *се свежда до* **PR2** *и пишем*

$$\mathbf{PR1} \lll_{f(n)} \mathbf{PR2}$$

ако има алгоритъм, разрешаващ **PR1**, *който константен брой пъти използва алгоритъм, решаващ* **PR2**, *както и извършва* $\mathcal{O}(f(n))$ *допълнителна работа.*

Задача 3.8 (**3-Sum-Target**)**.**

Вход: $A[1...n], target$ *- масив от рационални числа и търсена сума.*
Изход: Има ли $1 \leq i < j < k \leq n$*, такива, че* $A[i] + A[j] + A[k] = target$*?*

Твърдение 3.7 (връзка между проблемите **3-SUM** и **3-SUM-Target**)**.**
Проблемите **3-SUM** *и* **3-SUM-Target** *лесно се свеждат един към друг със следните алгоритми:*

- **3-SUM-Target** $\lll_n$ **3-SUM**
  - *За време* $\mathcal{O}(n)$ *заменяме* $A[i] \rightsquigarrow A[i] - target/3$.
  - *Връщаме резултата от извикването на* **3-SUM** *при вход модифицирания масив* A.
- **3-SUM** $\lll_1$ **3-SUM-Target**
  - *Връщаме резултата от извикването на* **3-SUM-Target** *при вход масива* A *и целева сума* 0.

Задача 3.9 (**3-Collinear**)**.**

Вход: $P_1, ..., P_n$ *- представяния на точки в равнината.*
Изход: Има ли $1 \leq i < j < k \leq n$*, такива, че* P_i, P_j, P_k *са колинеарни?*

Твърдение 3.8.

$$\textbf{3-Sum} \lll_{n \log n} \textbf{3-Collinear}$$

Алгоритъм за **3-Sum***:*

1. *За време* $\mathcal{O}(n \log n)$ *може да се провери дали има тройка индекси, за които някои два елемента са равни и сумата на трите е* 0 *(как?)*

2. Премахваме повтарящите се елементи

3. За всеки елемент на A конструираме $A[i] \rightsquigarrow (A[i], A[i]^3)$

4. Връщаме резултата от **3-Collinear** *при вход съпоставените точки.*

Използваме фактът, че $A[i] + A[j] + A[k] = 0$ т.с.т.к. $\langle A[i], A[i]^3\rangle, \langle A[j], A[j]^3\rangle, \langle A[k], A[k]^3\rangle$ са колинеарни.

(Второто е еквивалентно на нулева детерминанта $\begin{vmatrix} A[i] & A[i]^3 & 1 \\ A[j] & A[j]^3 & 1 \\ A[k] & A[k]^3 & 1 \end{vmatrix}$ *)*

Задача 3.10 (Concurrent Lines).

Вход: $l_1, ..., l_n$ *- представяния на прави в равнината.*
Изход: Има ли $1 \leq i < j < k \leq n$*, такива, че* l_i, l_j, l_k *се пресичат в една точка?*

Допълнение 3.1 (Concurrent Lines).

3-Collinear $\lll_{n \log n}$ Concurrent Lines

Алгоритъм за **3-Collinear***:*

1. *Сортираме по x точките за $\mathcal{O}(n \log n)$*
2. *Проверяваме дали има някои с еднаква x координата за $\mathcal{O}(n)$*
3. *Заменяме всяка точка $(x_i, y_i) \rightsquigarrow l_i : x_i x - y - y_i = 0$*
4. *Три точки с различни x координати са колинеарни т.с.т.к. съпоставените им прави се пресичат в 1 точка, т.е. връщаме резултата от* **Concurrent Lines** *при вход съпоставените прави.*

Използваме, че $(x_i, y_i), (x_j, y_j)$ с $x_i \neq x_j$ лежат на права $l : ax - y - b = 0$ т.с.т.к (a, b) е пресечната точка на l_i и l_j.

Неразгледани (до момента) задачи:

Задача 3.11 (**Max Cluster**).

Вход: $A[1...n]$ - *масив от рационални числа,* $d \in \mathbb{Q}^+$.
Изход: $\max\{|S| \mid S \subseteq A \,\&\, \forall x, y \in S\,(|x - y| \leq d)\}$.

Задача 3.12 (**IntersectingSegments**).
Разглеждаме правите в равнината $y = 0$ *и* $y = 1$. *Върху* $y = 0$ *са разположени* n *различни точки*

$$(a_1, 0), (a_2, 0), ..., (a_n, 0)$$

Върху $y = 1$ *са разположени* n *различни точки*

$$(b_1, 1), (b_2, 1), ..., (b_n, 1)$$

С s_i *бележим отсечката с краища* $(a_i, 0)$ *и* $(b_i, 1)$.

Вход: $A[1...n], B[1...n]$ - *масив от абсцисите на точките.*
Изход: $|\{(s_i, s_j) \,\&\, 1 \leq i < j \leq n \,\&\, s_i \cap s_j \neq \emptyset\}|$.

4 Двоично търсене (Разделяй и владей)

Задача 4.1 (**Flip**). *$A[1...n]$, за който $A[1] > A[n]$; да се намери позицията на елемент $A[i] > A[i+1]$.*

Вход: $A[1...n]$ *- масив от рационални числа, за който* $A[1] > A[n]$.
Изход: $i \in \mathbb{N}$, *такова, че* $1 \leq i < n$ *и* $A[i] > A[i+1]$.

Лема 4.1. *Ако за редица от рационални числа $\{a_i\}_{i=1}^{n}$ е изпълнено, че $a_1 > a_n$, то има $1 \leq i < n$, такова, че $a_i > a_{i+1}$.*

Тогава такъв индекс в масива, подаден на вход има. Удобно е да го потърсим двоично:

```
Flip( A[1...n] : array of rationals, such that A[1] > A[n])
@1  left ← 1
@2  right ← n
@3  while right − left > 1 do
@4   m ← ⌊(left + right)/2⌋
@5   if A[left] > A[m] then
@6    right ← m
@7   else
@8    left ← m
@9   end if
@10 end while
@11 return left
```

Твърдение 4.2. *При вход $A[1...n]$, за който $A[1] > A[n]$,* Flip *връща индекс i, такъв, че $1 \leq i < n$ и $A[i] > A[i+1]$. Също така* Flip *има времева сложност $\mathcal{O}(\log n)$.*

По-удобно за анализ е следното рекурсивно решение на същия проблем:

```
FlipRec( A[l...r] : array of rationals, such that A[1] > A[n])
@1  if r − l <= 1 then return l
@2  m ← ⌊(l + r)/2⌋
@3  if A[l] > A[m] then return FlipRec(A[l...m])
@4  return FlipRec(A[m...r])
```

FlipRec поражда следното уравнение за времевата си сложност:

$$T(2) \leq 2$$
$$T(n) \leq T\left(\frac{n}{2}\right) + \mathcal{O}(1)$$

...чието решение е $T(n) = \log(n)$.

Задача 4.2 (Домашно 1, писмен изпит 2023г.)**.**
За матрица $A = (a_{i,j})_{i,j=1}^{n,n}$ от нули и единици ще казваме, че е 4-интересна, ако сумата на четирите ѝ ъглови елемента не се дели на 4, т.е., ако:

$$a_{1,1} + a_{1,n} + a_{n,1} + a_{n,n} \not\equiv 0 \pmod 4$$

1. *Да се докаже, че ако A е 4-интересна, то A съдържа подматрица 2×2, определена от два съседни реда и два съседни стълба, която е 4-интересна.*

2. *Да се предложи алгоритъм с времева сложност* $O(\log n)$*, който решава следния проблем:*

   Вход: $A[1...n][1...n]$ - 4-*интересна матрица*
   Изход: (i,j)*, за които* $A[i...i+1][j...j+1]$ *е* 4-*интересна подматрица.*

Ще използваме FLIP, заедно със следните негови модификации:

- FLIPCOL - получава на вход матрица $A[1...n][1...n]$, както и индекс на неин стълб j, за който $A[1][j] > A[n][j]$ и връща индекс на ред i, такъв, че $A[i][j] > A[i+1][j]$.
  Единствено заменяме проверката на @5 с $A[left][j] > A[m][j]$
- FLIPINV - получава на вход масив $A[1...n]$, за който $A[n] > A[1]$ и връща индекс $1 < i \leq n$, такъв, че $A[i] > A[i-1]$.
  Единствено заменяме проверката на @5 с $A[left] < A[mid]$.
- FLIPCOLINV - получава на вход матрица $A[1...n][1...n]$, както и индекс на неин стълб j, за който $A[1][j] < A[n][j]$ и връща индекс на ред i, такъв, че $A[i][j] > A[i-1][j]$.
  Единствено заменя проверката на @5 с $A[left][j] < A[m][j]$

4INTERESTING($A[1...n][1...n]$: 0/1 matrix)

```
@1  if A[1][1] > A[1][n] then
@2   pos ←FLIP(A[1][1...n])
@3   return  A[1...2][pos...pos+1]
@4  else if A[1][n] > A[n][n] then
@5   pos ←FLIPCOL(A[1...n][1...n],n)
@6   return  A[pos...pos+1][n-1...n]
@7  else if A[n][n] > A[n][1] then
@8   pos ←FLIPINV(A[n][1...n])
@9   return  A[n-1...n][pos-1...pos]
@10 end if
@11 pos ←FLIPCOLINV(A[1...n][1...n],1)
@12 return  A[pos-1...pos][1...2]
```

Твърдение 4.3.
При вход 4-*интересна матрица* $A[1...n][1...n]$, 4INTERESTING *връща за време* $\mathcal{O}(\log n)$ 4-*интересна* 2×2 *подматрица на* A.

Най-съществената част от доказателството на **Твърдение** 4.3 е фактът, че някое от условията $A[1][1] > A[1][n]$, $A[1][n] > A[n][n]$, $A[n][n] > A[n][1]$ и $A[n][1] > A[1][1]$ е истина за всяка 4-интересна матрица.

Неразгледани (до момента) задачи:

Задача 4.3 (Домашно 1, Контролно 1, 2023г.).

1. *На окръжност по часовниковата стрелка, са написани n реални числа, $a_0, ..., a_{n-1}$, със сума 0. Да се докаже, че има $i \leq n$, за което:*

$$\sum_{k=0}^{j} a_{(i+k) \ (mod\ n)} \geq 0 \text{ за всяко } j \geq 0. \qquad (\star)$$

2. *Да е предложи алгоритъм със сложност $\mathcal{O}(n)$, който решава следния проблем:*

   Вход: $A[0...n-1]$ - *масив от цели числа със сума* 0.
   Изход: i, *за което е изпълнено* $(\star)$.

3. *Да се докаже, че за всяко непразно множество $M = \{x_1, ..., x_n\}$ от точки в равнината и всяка права l има точка $x_i \in M$, такава, че правата през x_i, успоредна на l, разделя равнината на две полуравнини, като една от двете не съдържа точки от M*

4. *Да се предложи алгоритъм със сложност $\mathcal{O}(n)$, който решава следния проблем:*

   Вход: $M[1...n][2]$ - *масив от целочислени точки в ранината,* $v[2]$ - *ненулев целочислен вектор*
   Изход: i - *индекс на точка* $M[i]$, *такава, че* $\langle \overrightarrow{M[i]M[j]}, v \rangle \geq 0$ *за всяко* $j \leq n$.

Задача 4.4 (Локален минимум в масив).
Ще казваме, че индексът $1 < i < n$ е локален минимум за масива $A[1...n]$, когато

$$A[i-1] > A[i] < A[i+1]$$

Предложете алгоритъм с времева сложност $\mathcal{O}(\log n)$, решаващ следния проблем:

Вход: $A[1...n]$ - *масив от рационални числа, в който има индекс на локален минимум*
Изход: i - *индекс на локален минимум.*

Следва продължение...