Европейски съюз

ОП "Развитие на човешките ресурси"
Договор BG051PO001-4.3.04-0018
Разработване на програми за електронни форми на дистанционно обучение във Факултета по математика и информатика

Европейски социален фонд

Шеста лекция по ДАА на втори поток КН
Долни граници върху сложност на задачи.

07 април 2014

Абстракт

Въвеждаме понятието долна граница върху сложност на задача. Доказваме долни граници върху броя на сравненията в две занимателни задачи. Въвеждаме понятието дърво на вземане на решение. Използвайки този модел, доказваме долна граница $\Omega(n \lg n)$ върху сложностите на Сортиране и Уникалност на елементи, при определени допускания.

Съдържание

1 Долни граници върху сложността на изчислителни задачи

Долна граница върху сложността на изчислителна задача Π (на английски, *lower bound on a computational problem* Π) е функция $\phi(n)$, такава че **всеки алгоритъм**, който решава Π, работи във време $\Omega(\phi(n))$. Значението на това понятие е голямо: доказателство за долна граница на задача обезсмисля опитите да бъдат конструирани по-бързи алгоритми. Долната граница е фундаментално ограничение, нещо като природна даденост, с която сме длъжни да се съобразяваме, независимо дали ни харесва или не, също както не можем да игнорираме, примерно, закона за запазване на енергията.

Доказателствата на нетривиални долни граници върху сложността на задачи са изключително трудни. Какво е тривиална долна граница, ще обясним с пример. За Сортиране, тривиална долна граница е $\Omega(n)$, защото няма как да сортираме масив, без да сме "прегледали" всеки елемент[†]. Доказателството, че $\Omega(n \lg n)$ е долна граница за Сортиране е значително по-трудно и затова казваме, че не е тривиално.

Забележете една принципна разлика между горната и долната граница. От това, че Insertion Sort работи във време $O(n^2)$ и факта, че Insertion Sort решава задачата Сортиране, следва, че $O(n^2)$ е (асимптотична) горна граница за Сортиране. Но от това, че Mergesort работи във време $\Omega(n \lg n)$ и факта, че Mergesort решава задачата Сортиране, **не следва**, че $\Omega(n \lg n)$ е (асимптотична) долна граница за Сортиране. С други думи, горната граница на един конкретен алгоритъм е горна граница за задачата, но долната граница на един конкретен алгоритъм не е долна граница за задачата.

И още една забележка. Това, че $\Omega(n)$ е долна граница за Сортиране и това, че $\Omega(n \lg n)$ е долна граница за Сортиране (което все още не сме доказали), не са в противоречие. Просто втората долна граница е по-висока,

[†]Аналогично, както ще видим в следваща лекция, тривиална долна граница за обхождане на граф, представен със списъци на съседства, е $\Omega(n + m)$, където n е броят на върховете и m е броят на ребрата, поради това, че размерът на списъците на съседства е $\Theta(n + m)$, и няма как да обходим графа, без да сме "прегледали" всеки елемент от тези списъци.

следователно е и по-добра (и по-интересна). Когато става дума за долни граници, интересуваме се от колкото е възможно по-високи такива. Аналогично, при горните граници искаме колкото е възможно по-ниски такива: $O(n^9)$ е валидна горна граница за Сортиране, но тя е безинтересна предвид факта, че $O(n^2)$ също е горна граница, а на свой ред тя е безинтересна предвид това, че $O(n \lg n)$ е горна граница.

2 Задачите the Balance Puzzle и the Twelve-Coin Puzzle

Сега ще разгледаме две занимателни задачи, решението на които ще ни даде необходимата интуиция за ключовото понятие "дърво на вземане на решения".

2.1 The balance puzzle

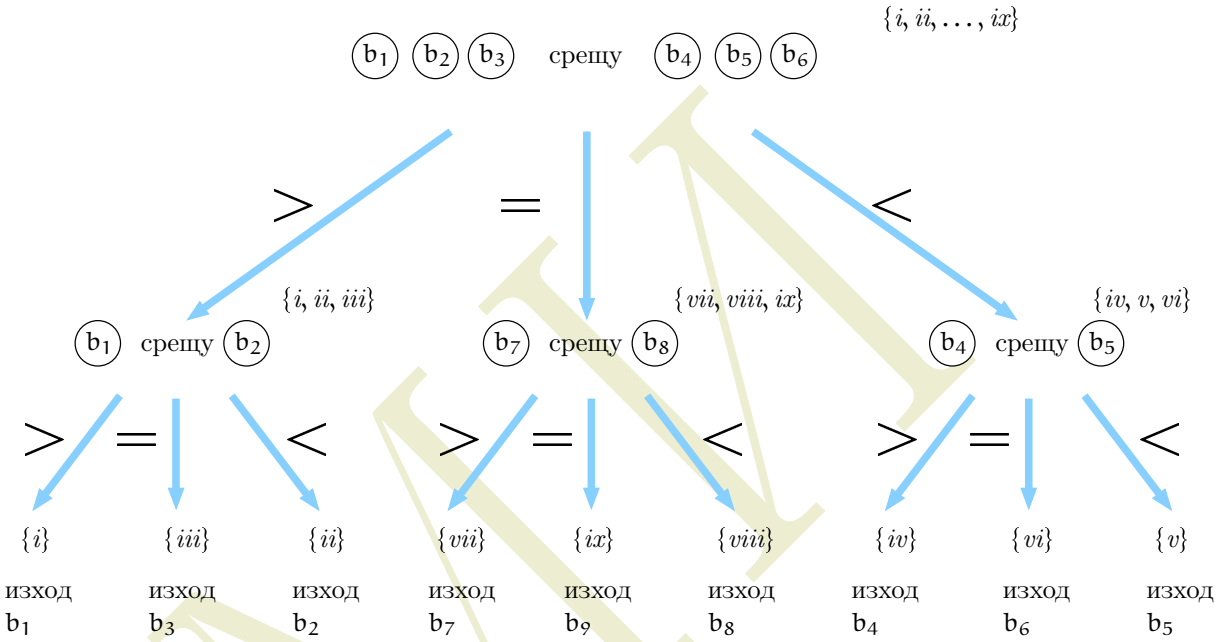
Задача 1 (The balance puzzle)**.** *Дадени са 9 номерирани предмета, да речем топки. Осем от тях имат едно и също тегло, а една топка е по-тежка (от коя да е от осемте). Нашата цел е да идентифицираме тежката топка, като използваме везни. Везните нямат стандартни теглилки, така че можем правим единствено измервания от вида: някои топки на едното блюдо срещу други топки на другото блюдо, и да наблюдаваме резултата. Има точно три възможности за резултата: везната да се наклони наляво или везната да се наклони надясно или или везната да остане балансирана. Трябва да намерим тежката топка с колкото е възможно по-малко измервания.*

Предложете схема за измервания, която ползва не повече от две измервания. Докажете, че две е долна граница за броя на измерванията.

Решение:
Нека деветте топки са $b_1, \ldots, b_9$. С едно използване на везната мерим b_1, b_2 и b_3 срещу b_4, b_5 и b_6. Има точно три възможни изхода.

- Ако b_1, b_2 и b_3 са по-тежки от b_4, b_5 и b_6, използваме везната втори път с b_1 срещу b_2. Отново има точно три възможни изхода.
  - Ако b_1 е по-тежка от b_2, връщаме "b_1 *е тежката топка*".
  - Ако b_2 е по-тежка от b_1, връщаме "b_2 *е тежката топка*".
  - Ако b_1 и b_2 са еднакво тежки, връщаме "b_3 *е тежката топка*".
- Ако b_4, b_5 и b_6 са по-тежки от b_1, b_2 и b_3, използваме везната втори път с b_4 срещу b_5. Отново има точно три възможни изхода.
  - Ако b_4 е по-тежка от b_5, връщаме "b_4 *е тежката топка*".
  - Ако b_5 е по-тежка от b_4, връщаме "b_5 *е тежката топка*".
  - Ако b_4 и b_5 са еднакво тежки, връщаме "b_6 *е тежката топка*".
- Ако b_1, b_2 и b_3, от една страна, и b_4, b_5 и b_6, от друга страна, са еднакво тежки, използваме везната втори път с b_7 срещу b_8. Отново има точно три възможни изхода.
  - Ако b_7 е по-тежка от b_8, връщаме "b_7 *е тежката топка*".
  - Ако b_8 е по-тежка от b_7, връщаме "b_8 *е тежката топка*".
  - Ако b_7 и b_8 са еднакво тежки, връщаме "b_9 *е тежката топка*".

Фигура 1 илюстрира тази схема. Много полезно е да мислим в термините на възможности, или възможни състояния. Поначало има точно девет възможности: или b_1 е тежката топка, или b_2 е тежката топка, и така нататък, или b_9 е тежката топка. Нека запишем тези девет състояния със, съответно, *i*, *ii*, ..., *ix*. В началото всяко множеството от възможностите е $\{i, ii, \ldots, ix\}$. След всяко измерване, множеството от възможностите намалява. Задачата бива решена когато множеството от възможностите стане едноелементно. На Фигура 1 до всеки връх е записано множеството от възможностите, които са консистентни с резултатите от измерванията досега, изключвайки текущото. Тоест, консистентни с резултатите от всички върхове от корена до родителя на настоящия връх.



Фигура 1: Схема за измерване с не повече от два въпроса за the balance puzzle. Възможностите са записани с римски цифри. Да решим задачата е същото като да отговорим, коя от деветте възможности се е случила.

Доказателството за долна граница е тривиално. Схемата трябва да е такава, че да различи една възможност от общо девет, използвайки тернарно[†] дърво на вземане на решение. Тернарно дърво с височина 1 има само три листа, така че може да различава само три възможности, което не върши работа за тази задача. Очевидно дървото трябва да има височина поне 2, за да има поне девет листа, с които да различава девет възможности. Забележете, че аргументацията за долната граница няма **нищо общо** с конкретната схема, която изложихме горе. Това, че изведохме долна граница две за броя на измерванията съвсем не означава автоматично, че тази долна граница е достижима. В случая, тя **е** достижима, но това следва от показаната конкретна схема. Аргументът за долна граница казва само, че с по-малко измервания няма да стане. □

2.2 The twelve-coin puzzle

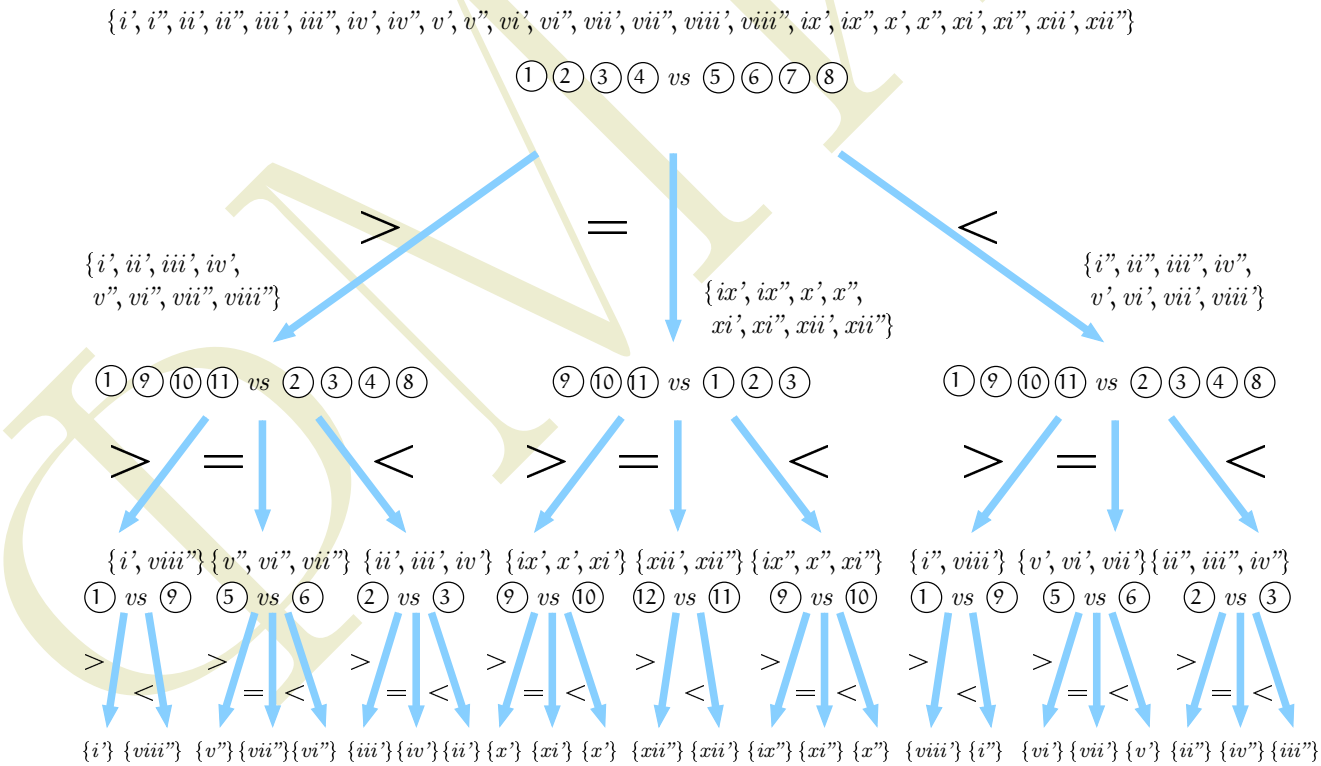
Задача 2 (The twelve-coin puzzle)**.** *Дадени са* 12 *номерирани монети. Измежду тях,* 11 *имат едно и също тегло, а другата—да я наречем* странната монета*—е или по-лека, или по-тежка. Нашата задача е да идентифицираме странната монета, използвайки везни като тези в Задача 1. Монетите са различими на външен вид, примерно са номерирани* $c_1, \ldots, c_{12}$*. Предложете схема за измервания, с която идентифицираме странната монета с не повече от три измервания. Докажете, че три е долна граница за броя на измерванията.*

Решение:
Тази задача е по-сложна от предишната. Броят на възможностите сега е 24: два пъти броят на монетите. Нека *i'* означава "c_1 е тежка", *i"* означава "c_1 е лека", *ii'* означава "c_2 е тежка", by *ii"* означава "c_2 е лека", и така нататък, *xii'* означава "c_{12} е тежка" и *xii"* означава "c_{12} е лека". Фигура 2 показва схема от измервания с най-много три използвания на везните, която определя коя от 24-те възможности е случаят. Дотук доказахме, че три измервания са достатъчни.

Сега ще докажем, че три измервания са необходими. Тъй като искаме да различим една възможност от общо 24 възможности и всяко измерване има най-много три възможни изхода, долна граница за броя на измерванията е $\lceil \log_3 24 \rceil = 3$. Защо? Защото тернарно дърво с $\geqslant 24$ листа има височина поне три.

Забележете, че долната граница, която следва от такива съображения (височината на дървото като логаритъм от броя на листата) не винаги е достижима. В случая с twelve-coin puzzle, долната граница наистина е

[†]"Тернарно" кореново дърво е такова, в което всеки връх има най-много три деца.



Фигура 2: Схема от измервания за the twelve-coin puzzle. Забележете, че понякога само два от трите изхода са възможни. Всеки връх в дървото е асоцииран с множеството от възможностите, които са консистентни с измерванията до този момент.

достижима, както виждаме от схемата на Фигура 2. Но същата задача с 13 монети **не може** да бъде решена с най-много три измервания в най-лошия случай. Ето защо:

- Няма смисъл да мерим едно срещу друго две множества от монети с различен брой монети, защото, ако блюдото с повечето монети се наклони надолу, ние не придобиваме **никаква нова информация**, в смисъл, че възможностите не намаляват, а остават същите.
- Имайки предвид предното съображение, за всеки избор на две подмножества с една и съща кардиналност (от множеството от 13-те монети), или тези множества имат кардиналност $\geqslant 5$, или множеството от останалите монети има кардиналност $\geqslant 5$.
- Множество с кардиналност $\geqslant 5$ има $\geqslant 10$ възможности за странната монета.
- Няма начин да определим една възможност измежду $\geqslant 10$ възможности, използвайки $\leqslant 2$ измервания, защото при не повече от три изхода от измерване, две последователни измервания могат да различат най-много 9 възможности.

Показахме долна граница четири за броя на измерванията. От друга страна, $\lceil \log_3 26 \rceil = 3$. Няма противоречие между факта, че $\lceil \log_3 26 \rceil = 3$, и факта, че the thirteen-coin puzzle изисква поне четири измервания в най-лошия случай. Това са различни аргументации за долна граница, като едната аргументация дава долна граница три, а другата, четири. Три си остава валидна долна граница, просто за задачата the thirteen-coin puzzle долната граница три не е достижима; иначе казано, долната граница три не е точна. □

3 Долна граница $\Omega(n \lg n)$ за Сортиране

Преди да докажем каквато и да е долна граница за сложността на някаква изчислителна задача, трябва да е напълно ясно какъв е изчислителният модел, който ползваме, и какви са допусканията, които правим. Очевидно е, че ако сортираме нули и единици, можем да извършим сортирането в линейно време, просто преброявайки елементите от всеки вид. Долната граница $\Omega(n \lg n)$, която ще докажем, е в сила не за всяко сортиране, а за *сортиране, базирано на директни сравнения*, на английски *comparison-based sorting*. Сортирането, базирано на директни сравнения, прави достъп (access) до елементите **единствено чрез операция директно сравнение на два елемента** $a_i \overset{?}{<} a_j$. Сортиране, което прави какъвто и да е друг вид достъп до елементите, примерно брои по колко елемента от дадена големина има, или разглежда остатъците по модул на елементите, или сравнява елементи с число, което не е част от входа, не е базирано на директни сравения. Образно казано, ако елементите са номерирани топки с положителни тегла, сортирането, базирано на директни сравнения, използва везна, подобна на везните от предната секция, само че с два, а не три възможни изхода; единствените разрешени действия с топките са мерене на две топки една срещу друга.

Има тънка разлика между сортиране, базирано на директни сравнения, и сортиращ алгоритъм, базиран на директни сравнения. В чистия абстрактен вид, сортирането, разирано на директни сравнения, **не размества елементи**, а само задава въпроси от вида $a_i \overset{?}{<} a_j$ и след поредица от такива въпроси казва—въз основа на получените отгори— коя е сортираната последователност от дадените елементи, примерно a_4, a_2, a_5, a_1, a_3. Сортиращите алгоритми, както вече видяхме, **разменят местата** на елементи в масива, така че в края на работата сортираната последователност пак бива наричана $A[1, \ldots, n]$, само че $A[1]$ в края не е непременно същият като $A[1]$ в началото и т.н. Сортиращите алгоритми, базирани на директни сравнения, правят само два вида действия с елементите: сравнения от вида $a_i \overset{?}{<} a_j$ и копирания (примерно, key $\leftarrow A[i]$ или swap$(A[i], A[j])$). Всички разгледани досега сортиращи алгоритми са базирани на директни сравнения.

Сега ще се убедим с пример, че на всеки сортиращ алгоритъм, базиран на директни сравнения (който мести елементи), съответства схема от въпроси, която нарекохме сортиране, базирано на директни сравнения (която не мести елементи). Да си припомним Insertion Sort.

```
Insertion Sort(A[1, ..., n])
1  for i ← 2 to n
2      key ← A[i]
3      j ← i − 1
4      while j > 0 and key < A[j] do
```

```
5            A[j + 1] ← A[j]
6            j ← j − 1
7        A[j + 1] ← key
```

Да разгледаме работата на Insertion Sort върху вход с големина три. Нека входът е

$$a_1, a_2, a_3$$

С малки букви, примерно "a_1", означаваме елементите от входа. С главни букви, примерно "$A[1]$", означаваме елементите от текещия масив $A[]$. В началото е вярно, че $A[] = [a_1, a_2, a_3]$, но след няколко стъпки от изпълнението може да имаме $A[] = [a_2, a_3, a_1]$. Да допуснем, че елементите от входа са уникални. Точно едно от следните е в сила:

$$a_1 < a_2 < a_3 \tag{1}$$
$$a_1 < a_3 < a_2 \tag{2}$$
$$a_2 < a_1 < a_3 \tag{3}$$
$$a_2 < a_3 < a_1 \tag{4}$$
$$a_3 < a_1 < a_2 \tag{5}$$
$$a_3 < a_2 < a_1 \tag{6}$$

Ще наричаме тези, *пермутациите*. Да сортираме входа е същото като да определим кое от (1), ..., (6) е вярно. Insertion Sort е алгоритъм, базиран на директни сравнения. Сравненията стават само на ред 4 в $\text{key} < A[j]$. Първото сравнение е $a_2 < a_1$.

Случай I Ако $a_2 < a_1$ е True, то точно тези три пермутации (от началните шест) са възможни:

$$a_2 < a_1 < a_3$$
$$a_2 < a_3 < a_1$$
$$a_3 < a_2 < a_1$$

В този случай алгоритъмът променя $A[]$ по време на текущото изпълнение на **for**-цикъла, така че $A[]$ става $[a_2, a_1, a_3]$. Други сравнения не се правят по време на текущото изпълнение на **for**-цикъла и започва ново негово изпълнение. Следващото сравнение е $a_3 < a_1$. Но първо да разгледаме алтернативния случай на **Случай I**.

Случай II Ако $a_2 < a_1$ е False, то точно тези три пермутации са възможни:

$$a_1 < a_2 < a_3$$
$$a_1 < a_3 < a_2$$
$$a_3 < a_1 < a_2$$

В този случай алгоритъмът не променя $A[]$ по време на текущото изпълнение на **for**-цикъла, така че $A[]$ остава $[a_1, a_2, a_3]$. Други сравнения не се правят по време на текущото изпълнение на **for**-цикъла и започва ново негово изпълнение.

Случай I.1 Връщаме се към **Случай I**. Ако $a_3 < a_1$ е True, точно тези две пермутации остават възможни:

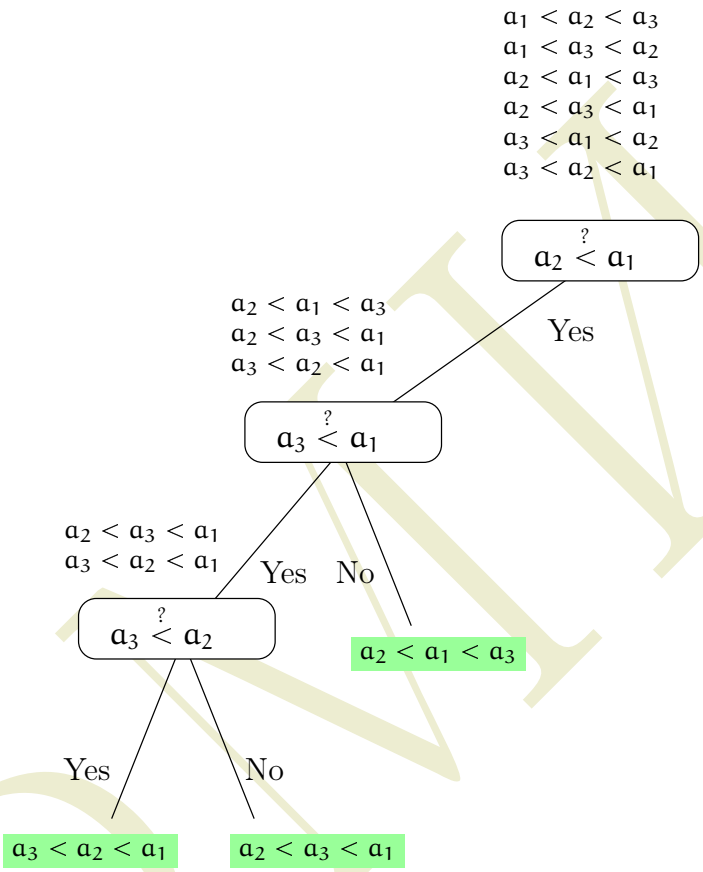
$$a_2 < a_3 < a_1$$
$$a_3 < a_2 < a_1$$

$A[]$ става $[a_2, a_1, a_1]$, като стойността a_3 се съхранява в променливата key. Следващото сравнение е $a_3 < a_2$.

Случай I.1.a Ако $a_3 < a_2$ е True, остава една единствена възможна пермутация:

$$a_3 < a_2 < a_1$$

while-цикълът се изпълнява още веднъж и $A[]$ става $[a_2, a_2, a_1]$. Тогава a_3 се записва на първа позиция в $A[]$ (line 7) и $A[]$ става $[a_3, a_2, a_1]$. После алгоритъмът приключва работата си.



Фигура 3: Подслучаите, в които $a_2 < a_1$ е TRUE.

I.1.b Ако $a_3 < a_2$ е FALSE, остава една единствена възможна пермутация:

$$a_2 < a_3 < a_1$$

while-цикълът не се изпълнява повече. a_3 се записва на втора позиция в $A[\,]$ (line 7)) и $A[\,]$ става $[a_2, a_3, a_1]$.

Случай I.2 Ако $a_3 < a_1$ е FALSE, остава една единствена възможна пермутация:

$$a_2 < a_1 < a_3$$

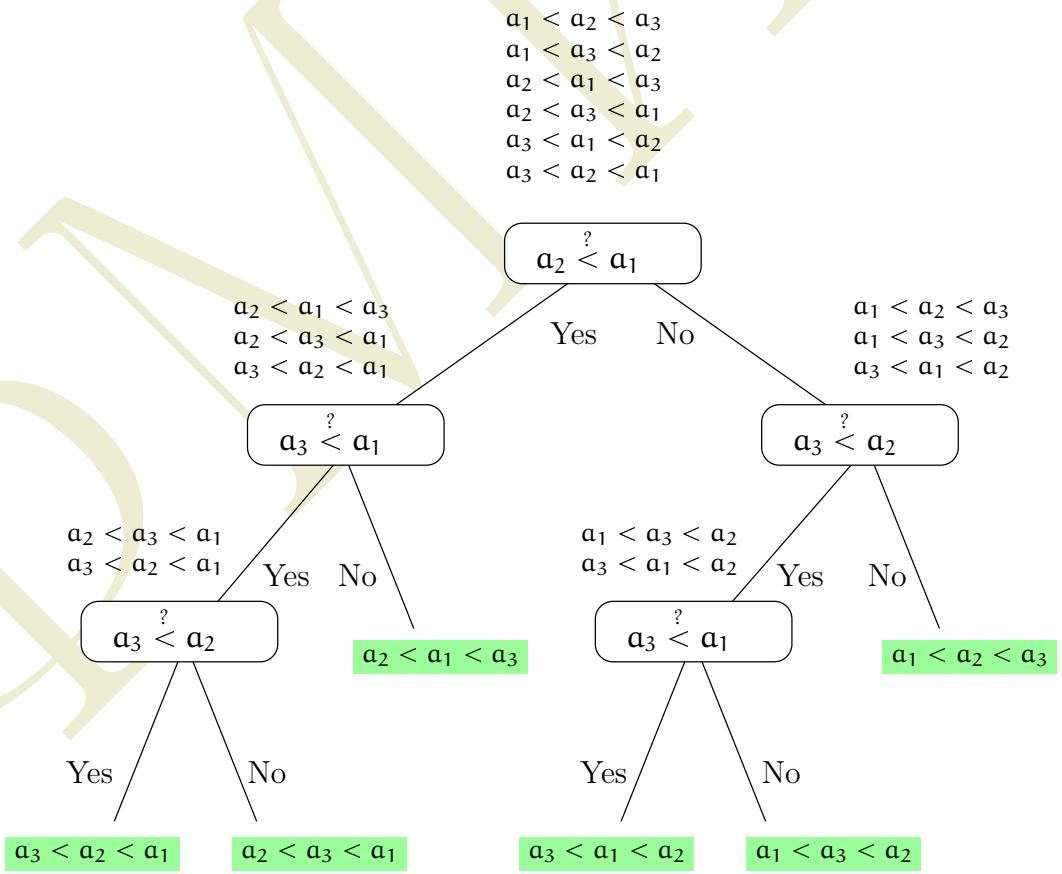
Алгоритъмът приключва работата си. $A[\,]$ остава $[a_2, a_1, a_3]$.

Четирите подслучай на **Case I** може да бъдат представени чрез дървоподобната структура на Фигура 3. Има два типа върхове—върхове-сравнения, съответстващи на въпроси от вида $a_i \overset{?}{<} a_j$, и лисна (в зелено). Над всеки връх-сравнение записваме множеството от пермутациите, които са консистентни с отговорите до този момент (преди това сравнение). Естествено, над корена са записани всички шест пермутации, които са възможни поначало. Листата отговарят на възможните изходи от алгоритъма.

Ако направим аналогичен анализ на подслучаите на **Случай II** ще получим структурата, показана на Фигура 4. Дървото T е двоично, понеже има два възможни отговора при всяко сравнение.

Изведохме дървоподобната структура чрез подробно симулиране на работата на INSERTION SORT върху всички възможни[†] входове с размер три. Такава дървоподобна схема от въпроси и отговори се нарича *дърво на вземане на решения*, на английски *decision tree*. Определението в [CLRS09] е:

[†]Като относителни големини.



Фигура 4: Това е цялото дърво T от сравненията. То представлява схема $\mathcal{S}$ от въпроси и отговори, които определят коя е сортираната пермутация на трите дадени елемента.

> *A decision tree* is a full binary tree[†] that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size. Control, data movement, and all other aspects of the algorithm are ignored.

Дърветата на вземане на решения няма нужда да са двоични и не са ограничени до задачата Сортиране. Както видяхме в миналата секция, известните задачи "the balance puzzle" и "the twelve-coin puzzle" (виж Задача 1 на стр. 2 and Задача 2 на стр. 3) са податливи на анализ чрез дървета на вземане на решение, само че тернарни.

При всяка от тези задачи:

- the balance puzzle,
- the twelve-coin puzzle,
- Insertion Sort върху вход с размер три

дървото на вземане на решения е едно единствено, защото става дума за една единствена големина на входа. Очевидно е, че Insertion Sort по принцип има не едно такова дърво, а безкрайно множество от дървета, по едно за всеки размер на входа. И the balance puzzle, и the twelve-coin puzzle имат очевидни обобщения за вход с произволна големина. Тези обобщения също имат безкрайни множества от дървета на вземане на решение, по едно за всяка големина на входа.

> **Подчертаваме, че ако A е произволен сортиращ алгоритъм, базиран на директни сравнения, A има не едно единствено дърво на вземане на решения, а безкрайно множество такива дървета, по едно за всеки размер на входа. Ако кажем просто "дървото на вземане на решения", имаме предвид общия елемент на това множество.**

При разсъжденията за Insertion Sort ние изведохме дървото. Но е важно е да разберем, че може да мислим за дървото като първичен обект. Дървото, в някакъв смисъл, сортира, само че не чрез местене на елементи, а определяйки коя е сортираната пермутация. И така, може да мислим, че дървото е първичният обект, а алгоритъмът е вторичен, като алгоритъмът мести елементи по такъв начин, че в правилният момент "на везната"[‡] да бъдат поставени правилните елементи.

Една странична забележка. Дървото на вземане на решение е изчислителен модел, също както крайният автомат или машината на Тюринг са изчислителни модели. Казваме, че дървото на вземане на решение е неуниформен изчислителен модел, *на английски* nonuniform computational model, *защото за всяка големина на входа изчисляващото дърво е различно. Докато автоматът и машината на Тюринг са униформени изчислителни модели, защото за всяка големина на входа, автоматът (машината) е един и същи.*

Сега ще покажем, че дърветата на вземане на решение ни дават механизъм за доказване на долни граници на задачи. Първото ще разгледаме конкретното дърво T, съответстващо на Insertion Sort с размер на входа три (Фигура 4). Забелязваме, че има листа с дълбочина две и листа с дълбочина три. Това означава, че върху някои входове определяме сортираната пермутация с два въпроса, а върху други, с три. Височината на дървото е максималният брой въпроси, които са ни необходими **в най-лошия случай**, по отношение на конкретната схема. Естествено, има много други възможни схеми, отговарящи на други сортиращи алгоритми. Нека читателят сам си направи дървото-схема, отговарящо на Selection Sort с размер на входа три, и ще се убеди, че то е различно от дървото-схема на Insertion Sort.

Но, каквото и да е дървото-схема, за вход с размер три, то трябва да има височина поне три. Защо? Допуснете, че има дърво T' на вземане на решения за сортиране на три елемента, което има височина най-много две. Тоест, схема от въпроси $\mathcal{S}'$, която с най-много два въпроса определя коя е сортираната пермутация. Но T' също е двоично дърво, защото всеки въпрос има два възможни отговора. Тривиално наблюдение е, че T' не може да има повече от четири листа, откъдето веднага следва, че $\mathcal{S}'$ не може да различи повече от четири възможности. С други думи, $\mathcal{S}'$ не може да различи повече от четири пермутации. Само че общо пермутациите са шест, следователно $\mathcal{S}'$ не е коректна схема от въпроси. Полученото противоречие показва, че такова дърво T' не съществува.

[†] *Full binary tree* е двоично дърво, в което всеки вътрешен връх има точно две деца. *Binary tree* е дърво, в което всеки вътрешен връх има най-много две деца.

[‡] "Везната" в случая с Insertion Sort е `key < A[j]` .

Забележете, че този анализ не разглежда конкретните особености на хипотетичното дърво T'. Със сигурност бихме могли да разгледаме едно след друго всички дървета за вземане на решение за вход с размер три и да установим, че всяко от тях има височина поне три. Само че това би било твърде отегчително, и освен това този подход не се мащабира за произволен размер на входа. Докато не-конструктивният анализ се мащабира за произволен размер на входа: той извежда, че листата на хипотетичното дърво са по-малко от възможностите-пермутации, така че по принципа на Дирихле, поне едно листо е асоциирано с повече от една пермутация.

Ето как се мащабира не-конструктивният анализ за вход с размер n. За всеки вход с размер n има схема от въпроси, която установява сортираната пермутация, като въпросите са от вида $a_i \overset{?}{<} a_j$. Знаем, че такива схеми има, защото съществуват коректни сортиращи алгоритми, базирани на директни сравнения, а на всеки сортиращ алгоритъм съответства безкрайно множество от дървета на вземане на решения, по едно за всяка големина на входа. Размерите на тези дървета растат експлозивно с нарастването на n, защото броят на листата е $\geqslant n!$. Следователно, не е практично да се опитваме да рисуваме такова дърво за $n > 3$. Но това не е важно. Важното е, че ако разгледаме множеството $\mathfrak{T}_n$ от всички възможни дървета на вземане на решение за размер на входа n, за произволно n, **дървото с най-малка височина от $\mathfrak{T}_n$ задава долна граница за броя на сравненията за задачата Sorting върху вход с големина n**. Тогава долна граница за височината на кое да е дърво от $\mathfrak{T}_n$ е долна граница за изчислителната задача Сортиране, базирано на директни сравнения.

И така, всяко дърво от $\mathfrak{T}_n$ трябва да има поне $n!$ листа, защото толкова са възможните пермутации. Тъй като дървото е бинарно, височината му е поне логаритмична при основа 2 в броя на листата. Тогава всяко дърво от $\mathfrak{T}_n$ има височина $\geqslant \log_2 n!$. Както вече показахме в минала лекция, $\log_2 n! = \Theta(n \lg n)$. Веднага следва долна граница $\Omega(n \lg n)$ за изчислителната задача.

4 Долна граница $\Omega(n \lg n)$ за Уникалност на елементи

Изчислителна задача Element Uniqueness
Параметри: Числа $a_1, a_2, \ldots, a_n$
Въпрос: Дали всички числа са уникални?

Разглеждаме задачата само като базирана на директни сравнения. Алгоритъм за задачата Element Uniqueness е базиран на директни сравнения, ако достъпът до елементите става само по един начин: чрез изпълнения на операция на сравнения $a_i \overset{?}{<} a_j$ с бинарен изход. Изходите от тези сравнения определят и работата на алгоритъма. Наивният, очевиден алгоритъм е сравняването на всички ненаредени двойки от входа. Сложността на този подход очевидно е $\Theta(n^2)$. По-изтънчен подход е първо да сортираме входа във време $\Theta(n \lg n)$ и след това с едно "премитане" (на английски, *linear sweep*) да установим във време $\Theta(n)$ дали няма или има повтаряния: ако има повтарящи се елементи, те задължително образуват непрекъсната последователност. Сложността по време е $\Theta(n \lg n) + \Theta(n) = \Theta(n \lg n)$.

```
Comparison-based Element Uniqueness(A[1,...,n]: array of integers)
1  Sort(A[])
2  are_unique ← True
3  for i ← 2 to n
4      if A[i − 1] = A[i]
5          are_unique ← False
6  return are_unique
```

Ще докажем, че всеки алгоритъм за Element Uniqueness, базиран на директни сравнения, има сложност $\Omega(n \lg n)$. Ще направим доказателство с дърво на вземане на решения. Както вече посочихме при сортирането, всеки алгоритъм, базиран на директни сравнения, има безкрайно множество от дървета за вземане на решения, асоциирани с него, по едно за всяка големина на входа. Ние ще разгледаме общия елемент на това множество, който е дървото, асоциирано с вход n. За да докажем желаното твърдение, достатъчно е да докажем, че листата на дървото са $\geqslant n!$ на брой и после да приложим същите разсъждения, които направихме на тази страница за сортирането, базирано на директни сравнения.

За да докажем, че дървото има $\geqslant n!$ листа, достатъчно е да покажем, че то различава[†] всички пермутации – в смисъл, че всяко листо е асоциирано с не повече от една пермутация. По отношение на сортирането, това е тривиално: ако дървото има листо, асоциирано с повече от една пермутация, съответният алгоритъм не е коректен сортиращ алгоритъм. По отношение на уникалността на елементите, това не е толкова очевидно. Доказателството надолу следва доказателството от [SW11].

Лема 1. *Нека* X *е произволен алгоритъм за* Element Uniqueness, *базиран на директни сравнения. Нека неговото дърво за вземане на решения за вход с големина* n *е* T. *Всяко листо на* T *е асоциирано с най-много една пермутация на входните елементи.*

Доказателство:
Можем да допуснем, че входните елементи са уникални, защото правим анализ на най-лошия случай, а това това ограничение се мащабира: за всяко n има вход с големина n, състоящ се от уникални елементи, и щом X е коректен алгоритъм, той трябва да работи коректно и върху този вход.

Да допуснем, че най-малката стойност във входа е b_1, втората най-малка е b_2, и така нататък, най-голямата стойност е b_n. С други думи,

$$b_1 < b_2 < \ldots < b_n$$

Факт 1. *За всяко* k, *такова че* $1 \leqslant k < n$, X *сравнява* b_{k-1} *с* b_k *в даден момент.* Да допуснем противното. Тогава съществува вход $A[1,\ldots,n]$ от уникални елементи, такива че X не сравнява $(k-1)$-ия и k-тия по големина елемент. Щом X е коректен сортиращ алгоритъм и елементите на $A[\,]$ са уникални, $X(A[1,\ldots,n])$ връща True. Нека $A[i]$ е елементът от входа със стойност b_{k-1} и $A[j]$ е елементът от входа със стойност b_k. Да преобразуваме $A[1,\ldots,n]$ в $A'[1,\ldots,n]$, присвоявайки b_k на $A[i]$[‡] и да изпълним $X(A'[\,])$. Понеже X е коректен алгоритъм, той трябва да върне False – забележете, че $A'[\,]$ има повтарящи се елементи. От друга страна обаче, работата на X върху $A'[\,]$ е напълно идентична на работата му върху $A[\,]$, понеже по допускане той никога не сравнява $A[i]$ с $A[j]$, а резултатите от всички други сравнения са едни и същи[§] и за двата входа. Но тогава $X(A[1,\ldots,n])$ и $X(A'[1,\ldots,n])$ връщат една и съща стойност. ↯

Доказахме, че X задължително сравнява всяка ненаредена двойка (има $n-1$ такива) от елементи **със съседни стойности**.

Факт 2. *Нека* $a_1, a_2, \ldots, a_n$ *са различни числа и* $a'_1, a'_2, \ldots, a'_n$ *е произволна тяхна пермутация, различна от идентитета. Нека* b_1 *е най-малката стойност измежду тях,* b_2 *е втората най-малка, и така нататък. Тогава съществува* k, *такова че* $1 \leqslant k < n$ *и за наредената двойка* $(k-1, k)$ *е в сила следното: ако* a_i *е елементът със стойност* b_{k-1} *и* a_j *е елементът със стойност* b_k[¶], *то* $a'_i > a'_j$. Този факт е тривиален, въпреки че формулировката звучи плашещо. Да разгледаме друга негова формулировка. Да си представим процес от две стъпки: първо числата $a_1, a_2, \ldots, a_n$ са подредени в линейна наредба, и на след това че биват разместени от "истинска" пермутация (различна от идентитет). Твърди се, че има съседни **по големина** елементи—забележете, не съседни по разположение, а **по големина**—в началото (преди пермутацията) със следното свойство. Тези елементи в началото са били на позиции i и j и този на позиция i е бил по-малък от този на позиция j. След пермутацията, на позиции i и j има такива елементи, че този на позиция i е по-голям от този на позиция j. Примерно, нека $n = 8$. Нека преди пермутацията сме имали

$$4, 11, 5, 2, 14, 18, 22, 7$$

а след пермутацията,

$$7, 4, 11, 5, 22, 18, 2, 14$$

[†] Трябва да поясним в какъв смисъл алгоритъм за уникалност на елементи различава пермутации. При сортирането е ясно, защото резултатът от процеса на сортиране е пермутация, точно една на брой. При уникалността обаче резултатът е булева стойност, а не пермутация, така че може би не е очевидно в какъв смисъл алгоритъмът различава пермутации. Но припомнете си, че алгоритъмът за уникалносте е базиран на **директни сравнения**. В самото начало всички пермутации са възможни в смисъл, че не знаем кой елемент е най-малък и т. н., но след като започнем да правим сравнения, след всяко сравнение, множеството от пермутациите, консистентни с изходите от сравненията до момента, намалява. И така, пермутациите, асоциирани с даден връх на дървото са точно тези, които са консистентни с изходите от сравненията до момента.

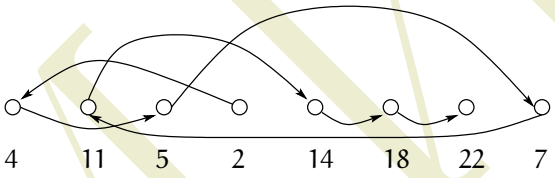
[‡] С други думи, правим $A[i] \leftarrow A[j]$ в самото начало.

[§] $A[i]$, чиято стойност беше повишена, си остава по-малък от елементите без $A[j]$, от които беше по-малък преди повишаването, и по-голям от елементите, от които беше по-голям преди повишаването.

[¶] Това влече $a_i < a_j$.

Тогава такава наредена двойка е $(4, 5)$. Четвъртият по големина елемент е $a_8 = 7$, а петият по големина е $a_2 = 11$. Очевидно $a_8 < a_2$. След пермутацията, $a'_8 = 14$ и $a'_2 = 4$, така че $a'_8 > a'_2$.

Доказателството на този факт става чрез допускане на противното. Противното е, за всеки две позиции, съдържащи елементи със съседни по големина стойности преди пермутацията, след пермутацията, посоката на неравенството между елементите на тези позиции се запазва. Но тогава следва, че пермутацията оставя всички елементи по местата им. Защо? Всяко линейно разполагане на n числа може да се опиша **еднозначно** чрез ориентиран граф, чиито върхове са позициите, и който има $n - 1$ ребра, построени така: от позиция i към позиция j има ребро тогава и само тогава, когато елементът на позиция i е непосредствен предшественик на елемента на позиция j. Да наречем този граф, *инкременталния граф* на списъка с числата. Примерно, инкременталният граф на последователността $(4, 11, 5, 2, 14, 18, 22, 7)$ е



Ако допуснем, че **Факт 2** не е истина, директно извеждаме, че две различни пермутации имат един и същи инкрементален граф, което е невъзможно. ↯

След като сме доказали **Факт 1** и **Факт 2**, доказателството на лемата става лесно. Да допуснем, че дървото на вземане на решения T на алгоритъм за уникалност на елементи X (базиран на директни сравнения) асоциира някое листо u с поне две пермутации. Тъй като те са различни пермутации, за коя да е от тях можем да мислим като за пермутация-неидентитет на другата. Съгласно втория факт, съществуват съседни по големина елементи в първата пермутация, такива че на техните съответни позиции i и j във втората пермутация има елементи, неравенството между които е в обратната посока (спрямо неравенството между елементите на позиции i и j в първата).

Да обясним това по-подробно. Има две пермутации $a_1, a_2, \ldots, a_n$ и $a'_1, a'_2, \ldots, a'_n$, асоциирани с листото u. Това означава, че и двете са консистентни с въпросите и отговорите досега (от корена надолу до u). Нека $b_1, \ldots, b_n$ са стойностите от входа, в нарастващ ред. Вторият факт казва, че за някои b_{k-1} и b_k, ако те са на позиции съответно i и j в $a_1, a_2, \ldots, a_n$, то в $a'_1, a'_2, \ldots, a'_n$, елементите на позиции i и j (съответно a'_i и a'_j) са такива, че $a'_i > a'_j$. Но в първата пермутация, $a_i < a_j$. Сега си припомняме първия факт. Той имплицира, че X трябва да е задал въпроса *"Дали* $b_{k-1} < b_k$*?"*. Само че **дървото на X не може да задава въпроси за съседни стойности директно**[†]. Както знаем, въпросите в дървото са от вида $a_i \overset{?}{<} a_j$. Ключовото наблюдение е, че по отношение на дървото, сравняването на b_{k-1} с b_k става чрез въпрос от типа $a_i \overset{?}{<} a_j$. Но за една от двете пермутации отговорът на този въпрос е Да, а за другата е Не. Тогава няма как листото u да е асоциирано и с двете, защото в някой връх v, предшественик на u, едното дете на v е асоциирано с едната пермутация, а другото дете, с другата.

Доказахме, че никое листо не може да бъде асоциирано с повече от една пермутация. □

След като сме доказали лемата, долната граница $\Omega(n \lg n)$ за задачата Уникалност на елементи, базирана на директни сравнения, следва от същите съображения като тази на Сортиране.

5 Долна граница $\Omega(n \lg n)$ за Най-близка двойка елементи

Изчислителна задача Closest Pair
Общ пример: Числа $a_1, a_2, \ldots, a_n$
Изход: Минималното $|a_i - a_j|$ за $1 \leqslant i < j \leqslant n$.

Това е едномерният вариант на задачата: входът са числа. Съществува и 2-мерен вариант, в който входът са наредени двойки от числа и се търси минимално разстояние между две двойки числа, където разстоянието може да се дефинира по различни начини, примерно Евклидово разстояние, Манхатънско разстояние и така нататък. Тук разглеждаме едномерния вариант.

[†] Ако можеше да задава въпроси за съседни стойности директно, задачата щеше да е решима в линейно време.

Без съмнение съществува директно доказателство на долната граница $\Omega(n \lg n)$ за тази задача с дърво на вземане на решение. Но, както видяхме при задачата Уникалност на Елементи, директното доказателство може да е дълго и тежко. Тук ще извършим *редукция*. А именно, ще редуцираме задачата Уникалност на Елементи до задачата Най-близка двойка елементи. По този начин ще използваме наготово резултата за долна граница на Уникалност на елементи. Редукцията става по следния начин.

Нека AlgCP е произволен алгоритъм за задачата Най-близка двойка елементи (базиран на директни сравнения, естествено). Да разгледаме следния алгоритъм AlgX.

```
AlgX(Числа a_1, a_2, ..., a_n)
1   d ← AlgCP(a_1, a_2, ..., a_n)
2   if d = 0
3       return False
4   else
5       return True
```

Очевидно AlgX решава задачата Уникалност на Елементи. Нещо повече, неговата сложност по време е равна, в асимптотичния смисъл, на сложността на AlgCP, защото извикването на AlgCP на ред 1 и последващото връщане, AlgX извършва само константна допълнителна работа (редове 2–5). **Ако** алгоритъмът AlgCP имаше сложност по време $o(n \lg n)$, то и AlgX щеше има сложност по време $o(n \lg n)$, а оттук и задачата Уникалност на Елементи щеше да има сложност $o(n \lg n)$. Само че ние вече знаем, че Уникалност на Елементи има долна граница $\Omega(n \lg n)$. Следователно, невъзможно е AlgCP да работи във време $o(n \lg n)$. Оттук имаме и долна граница $\Omega(n \lg n)$ за задачата Най-близка двойка елементи.

Казваме, че задачата Уникалност на Елементи *се редуцира* до задачата Най-близка двойка елементи. Точна дефиниция на понятието "редукция" ще дадем в следваща лекция. Тук просто споменаваме очевидния факт, че задачата Уникалност на Елементи има решение, което се състои в решение на задачата Най-близка двойка елементи, плюс пренебрежимо малко допълнителна работа.

За произволни изчислителни задачи Π_1 и Π_2 използваме нотацията $\Pi_1 \propto \Pi_2$, за да означим факта, че Π_1 се редуцира до Π_2. Грубо казано, от редукцията правим два извода:

- Ако Π_1 е тежка задача (каквото и да значи това), то Π_2 също е тежка. Няма как някаква вътрешна, иманентна сложност да "изчезне" при редукцията.
- Ако Π_2 е лесна (каквото и точно да значи това), то Π_1 също е лесна. По същата причина: не може иманентната сложност да изчезне при редукцията.

Важно е да знаем, че следните изводи **не са верни**.

- Ако Π_1 е лека, то Π_2 също е лека. **Такъв извод няма**. Винаги можем да направим нещата произволно сложни.
- Ако Π_2 е тежка, то Π_1 също е тежка. **Такъв извод няма**. Причината отново е, че винаги можем да направим нещата сложни.

Да запомним. Трансформирайки задача в задача, можем да добавим колкото си искаме допълнителна сложност отгоре. Това, което е невъзможно, е при редукция да намалим иманентната сложност.

Литература

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011.