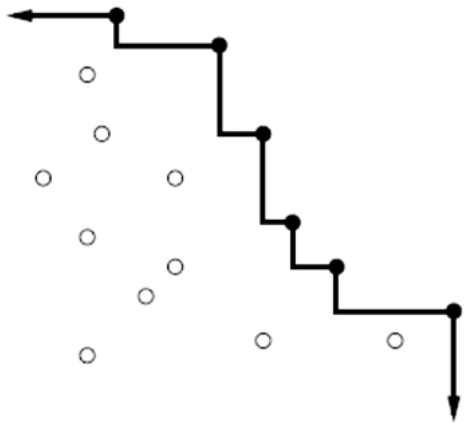
ПРИМЕРНО КОНТРОЛНО № 2 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 2. КУРС, 1. ПОТОК
(СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2019 / 2020 УЧ. Г.)

Задача 1. Да се реши по порядък рекурентното уравнение

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + 1. \qquad \textbf{(2 точки)}$$

Задача 2. Дадени са числовите масиви $X[1...n]$ и $Y[1...n]$ с еднаква дължина. Те съдържат съответно абсцисите и ординатите на n точки в равнината, т.е. k-тата точка е $(X[k]\,;\,Y[k])$. Казваме, че k-тата точка е покрита, ако съществува друга точка (например с индекс j), такава че $X[k] < X[j]$ и $Y[k] < Y[j]$. Непокритите точки образуват стълбичка.



а) Предложете алгоритъм, който намира всички непокрити точки
и построява стълбичката за време $O(n \log n)$ в най-лошия случай. **(2 точки)**
б) Анализирайте времевата сложност на алгоритъма. **(1 точка)**
в) Пресметнете математическото очакване (средния брой) на непокритите точки
от общо n случайно и независимо избрани точки. **(2 точки)**

Задача 3. Дадени са m изречения на един език и преводите им (m изречения) на друг език. Известно е кое изречение от единия език на кое изречение от другия език съответства. В изреченията от единия език има общо n различни думи; в изреченията от другия език също има общо n различни думи. Всяко изречение има същия брой думи като своя превод. (Броят се само важните думи: частици, предлози и т.н. са изрити предварително.) Не знаем коя дума от единия език на коя дума от другия език съответства. Това не може да се разбере от съпоставка на изреченията, защото словоредът в двата езика може да бъде различен: k-тата дума от едно изречение не съответства непременно на k-тата дума от неговия превод. Разглеждаме изреченията като множества (не редици) от думи, а словореда пренебрегваме.

Предложете алгоритъм, който за време $O(mn)$ да намира съответствието на думите от двата езика. Всяко от числата m и n е от порядъка на няколко хиляди. **(3 точки)**

РЕШЕНИЯ

Задача 1. Развиваме уравнението

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + 1,$$

като на всяка стъпка заместваме в дясната страна само събираемото с по-голям аргумент.

$$T(n) = 2T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + 2 = 3T\left(\frac{n}{8}\right) + 2T\left(\frac{n}{16}\right) + 4 = 5T\left(\frac{n}{16}\right) + 3T\left(\frac{n}{32}\right) + 7 = \ldots$$

Забелязваме, че коефициентите на уравнението в хода на развиването приемат поредни стойности от редицата на Фибоначи: $F_1 = 1,\ F_2 = 1,\ F_k = F_{k-1} + F_{k-2}$ за всяко цяло $k \geq 3$. По-точно, на k-тата стъпка уравнението има следния вид:

$$T(n) = F_{k+1}\,T\left(\frac{n}{2^k}\right) + F_k\,T\left(\frac{n}{2^{k+1}}\right) + F_{k+2} - 1$$

(за първа стъпка броим уравнението, което е дадено по условие). Тази формула се доказва с индукция по k. Развиваме уравнението, докато по-малкият аргумент на T стане от порядъка на единицата, тоест

$$\frac{n}{2^{k+1}} \approx 1 \ \Leftrightarrow\ k \approx -1 + \log_2 n \ \Leftrightarrow\ \frac{n}{2^k} \approx 2.$$

За тази стойност на k важи равенството

$$T(n) \approx F_{k+1}\,T(2) + F_k\,T(1) + F_{k+2} - 1.$$

Грешката на приближението е малка: аргументите на T се различават от истинската стойност с не повече от една единица (грешка от закръгляне). Това не влияе на порядъка на $T(n)$, защото $T(1)$, $T(2)$, $T(3)$ и т.н. са константни множители и имаме право да ги пренебрегнем:

$$T(n) \asymp F_{k+1} + F_k + F_{k+2}.$$

От известната формула $\quad F_k = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right]$

следва, че $\quad F_k \asymp F_{k+1} \asymp F_{k+2} \asymp \left(\frac{1+\sqrt{5}}{2}\right)^k,$

откъдето $\quad T(n) \asymp 3F_k \asymp F_k \asymp \left(\frac{1+\sqrt{5}}{2}\right)^k.$

Заместваме $\quad k \approx -1 + \log_2 n.$

Грешката на приближението (от закръгляне) е под 1. Ограничено събираемо в показателя съответства на ограничен множител пред степента, а той не променя порядъка. По същата причина пренебрегваме умалителя на k (константата 1) и оставяме само логаритъма:

$$T(n) \asymp \left(\frac{1+\sqrt{5}}{2}\right)^{\log_2 n} = n^{\log_2\left(\frac{1+\sqrt{5}}{2}\right)}.$$

Това е отговорът на задачата. Степенният показател на n е приблизително равен на 0,7.

Забележка: Последното равенство се основава на тъждеството $a^{\log_b c} = c^{\log_b a}$. Доказва се най-лесно чрез логаритмуване на двете страни при основа b: $\log_b c \,.\, \log_b a = \log_b a \,.\, \log_b c$.

Получената формула е очевидно следствие от разместителното свойство на умножението. А щом логаритмите на две числа са равни, то и самите числа са равни.

Задача 2.

а) Алгоритъм, който намира всички непокрити точки и построява стълбичката, може да бъде получен поне по два начина.

*Първи начин — по схемата **"разделяй и владей".***

Първи етап — намиране на непокритите точки:

1) Намираме медианата Md на абсцисите с помощта на алгоритъма PICK.
2) Md разделя точките на леви и десни. При няколко точки с x = Md горните са леви, долните са десни (делим ги с PICK тъй, че да има равен общ брой леви и десни точки).
3) Рекурсивно намираме непокритите десни точки.
4) Намираме maxY = най-голямата от ординатите на десните точки.
5) Обхождаме левите точки и изтриваме (обявяваме за покрити) тези с ордината < maxY.
6) Сред неизтритите леви точки рекурсивно търсим непокрити.

Забележка: Дъното на рекурсията е при $n = 1$. Сама точка винаги е непокрита.

Втори етап — образуване на стълбичката:

1) Сортираме непокритите точки по нарастване на абсцисите, а тези с равни абсциси — в низходящ ред на ординатите, например с пирамидално сортиране.
2) Свързваме последователните непокрити точки с начупена линия.

Обосновка за коректност на първия етап: Всяка дясна точка може да бъде покрита само от друга дясна точка. Затова непокритите десни точки се намират рекурсивно (т. 3). Ако една лява точка P е покрита изобщо от някоя дясна точка, то P е покрита също така от дясната точка Q с най-голяма ордината. Затова е достатъчно да проверим само за Q кои леви точки покрива (т. 4 и т. 5). Никоя от останалите (неизтритите) леви точки не се покрива от никоя дясна точка, тоест останалите леви точки могат да бъдат покрити само от други леви точки, което обосновава т. 6 от алгоритъма.

Обосновка за коректност на втория етап: Стълбичката върви надолу и надясно, тоест по нарастващ ред на абсцисите и по намаляващ ред на ординатите на непокритите точки.

*Втори начин — единствено със **сортиране:***

1) С бърз алгоритъм, например пирамидално сортиране, нареждаме точките по възходящ ред на абсцисите, а точките с равни абсциси — по низходящ ред на ординатите.
2) Търсим най-голямата ордината, обхождайки масива по намаляващ ред на абсцисите. Всяка точка, чиято ордината е по-голяма или равна на текущия максимум, обявяваме за непокрита. В частност, най-дясната точка винаги е непокрита.
3) Едновременно с търсенето на непокритите точки чертаем и стълбичката:
   след всяка непокрита точка чертаем водоравна отсечка наляво от нея.
   Когато намерим нова непокрита точка с ордината, по-голяма от текущия максимум, чертаем отвесна отсечка нагоре до новата стойност на текущия максимум.

Коректността на втория алгоритъм се получава от тази инварианта на цикъла от т. 2: Текущото множество точки, отбелязани като непокрити, съдържа тъкмо непокритите обходени точки, а текущият максимум е най-голямата измежду обходените ординати.

б) Анализ на времевата сложност на втория алгоритъм: Сортирането изисква време $n \log n$, а обхождането на масива — време n. Общото време е $\Theta(n \log n)$.

Анализ на времевата сложност на първия алгоритъм: Нека $T(n)$ е времето на първия етап в най-лошия случай: когато всички точки са непокрити. Този случай е най-лош, понеже т. 5 не изтрива нищо и рекурсията от т. 6 се изпълнява върху половината масив. Колкото до рекурсията от т. 3, тя във всички случаи се изпълнява върху половината масив, защото разделянето на точките на леви и десни е извършено относно медианата на абсцисите. Тъй като т. 1, т. 2, т. 4 и т. 5 изразходват линейно време, то за общото време на първия етап получаваме рекурентното уравнение

$$T(n) = 2T\left(\frac{n}{2}\right) + n.$$

От втория случай на мастър-теоремата следва, че решението на това уравнение е

$$T(n) = \Theta(n \log n).$$

Това е времевата сложност на първия етап в най-лошия случай.

Най-лошият случай за втория етап е да има възможно най-много непокрити точки, тоест всички дадени точки да са непокрити. Сортировката от т. 1 изразходва време $\Theta(n \log n)$, а обхождането от т. 2 — време $\Theta(n)$. Общото време на втория етап е $\Theta(n \log n)$.

Времето на целия алгоритъм (първия и втория етап заедно) е $\Theta(n \log n)$.

в) Да означим математическото очакване (средния брой) на непокритите точки с $f(n)$. Вторият алгоритъм ни подсеща как да съставим рекурентно уравнение за функцията f. Нека S е най-дясната точка. Тя със сигурност е непокрита, затова $f(n)$ е равно на единица плюс броя на непокритите измежду останалите $n - 1$ точки. Нека т. S покрива k от тях. Това означава, че ординатата на т. S се намира на $(k + 1)$-о място в списъка на ординатите на точките, ако ординатите са подредени в нарастващ ред. Точката S може да се намира на всяко място в списъка с еднаква вероятност, тоест k може да приема стойностите $0, 1, 2, 3, \ldots, n - 1$ с еднаква вероятност: $1 / n$. За всяко k, след като премахнем покритите k точки, остават $n - 1 - k$ точки, непокрити от т. S. Може обаче някои от тях да са покрити от други измежду същите $n - 1 - k$ точки. Средно колко от тях ще останат непокрити, е същата задача, но за по-малък брой дадени точки, затова отговорът се дава от същата функция: $f(n - 1 - k)$. Понеже k е случайна величина, трябва да вземем средното претеглено на всевъзможните стойности на $f(n - 1 - k)$, като за тегла служат вероятностите на различните стойности на k. Ала k има равномерно разпределение, т.е. стойностите на k имат еднаква вероятност $(1 / n)$, затова средното претеглено съвпада със средното аритметично. Ето защо функцията f удовлетворява рекурентното уравнение

$$f(n) = 1 + \frac{f(0) + f(1) + f(2) + f(3) + \cdots + f(n-2) + f(n-1)}{n}.$$

Преработваме: $n \,.\, f(n) = n + f(0) + f(1) + f(2) + f(3) + \cdots + f(n-2) + f(n-1)$.

Значи, $(n-1) \,.\, f(n-1) = (n-1) + f(0) + f(1) + f(2) + f(3) + \cdots + f(n-2)$.

Изваждаме последните две уравнения: $n \,.\, f(n) - (n-1) \,.\, f(n-1) = 1 + f(n-1)$,

тоест $n \,.\, f(n) = 1 + n \,.\, f(n-1)$, което е равносилно на $f(n) = f(n-1) + \frac{1}{n}$.

Развиваме това уравнение и вземаме предвид тривиалното начално условие $f(0) = 0$.

Краен резултат: $f(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \asymp \log n$.

Задача 3. Номерираме думите с числата от 1 до n, а изреченията — с числата от 1 до m; съответните изречения от двата езика (оригиналът и преводът) получават един и същи номер. На множеството от изреченията на всеки от двата езика съпоставяме двоична матрица с m реда и n стълба, показваща коя дума в кои изречения участва. В условието на задачата се подразбира, че това кодиране е извършено предварително, тоест входът на алгоритъма се състои от двете двоични матрици. (В противен случай времето $\Theta(mn)$ може да не стигне, а сложността ще зависи не само от m и n, но още от размера на азбуката и броя на буквите в думите и в изреченията.)

Наличието на взаимноеднозначно съответствие (превод) между думите на двата езика означава, че едната матрица поражда другата чрез неизвестна пермутация на стълбовете. Тъкмо тази пермутация търсим. Ще я получим като композиция на две други пермутации, чрез които всяка от двете матрици се свежда до някакъв стандартен вид. Ако тълкуваме всяка матрица като масив с n елемента — стълбовете от клетки, то трябва да сортираме тези два масива (тоест двете матрици). Двете пермутации, чиято композиция дава отговора, са пермутациите, които сортират масивите от стълбове. По-просто казано, след сортирането k-тият стълб на едната матрица съответства на k-тия стълб на втората матрица за всяко k. А тъй като стълбовете съответстват на думите, то по този начин получаваме съответствието на думите от двата езика.

Разместването на два стълба е бавно, защото m е голямо число. Затова ще разместваме не самите стълбове, а указатели към тях (тоест масивите, представящи двете матрици, ще бъдат масиви от указатели към стълбовете на матриците).

За да има смисъл понятието сортиране, трябва да уточним каква наредба на стълбовете ще използваме. Стълбовете са двоични вектори, поради което могат да бъдат сравнявани с помощта на лексикографската наредба.

Остава да изберем алгоритъма за сортиране. Методът на бройните системи е най-удобен за тази задача. Неговата времева сложност е $\Theta(mn)$, защото сортира стълбовете чрез броене m пъти — по веднъж за всеки ред от матрицата, — като всеки пас изразходва време $\Theta(n)$.

Естествено, можем да тълкуваме стълбовете като записи на цели неотрицателни числа в двоична бройна система. На пръв поглед можем да използваме някоя от сортировките с време $\Theta(n \log n)$, например пирамидалното сортиране. Теоретично не можем да сравним двете сложности, защото множителите m и $\log n$ са независими. На практика обаче можем да ги сравним, като използваме информацията, че m и n са от порядъка на няколко хиляди. В такъв случай $\log n$ (ако го тълкуваме като двоичен логаритъм) е около десет, тоест $\log n$ е много по-малко от m. (Ако за основа на логаритъма изберем число, по-голямо от 2, тогава той ще намалее още повече.) Така стигаме до извода, че $\log n$ е много по-малко от m, тоест пирамидалното сортиране е по-бързо от метода на бройните системи.

Този извод не е верен! Наистина, $\log n$ е много по-малко от m, обаче времевата сложност на пирамидалното сортиране (и на сортирането чрез сливане) в тази задача не е $\Theta(n \log n)$. Сложността $\Theta(n \log n)$ е изведена при предположението, че всеки два елемента на масива се сравняват за константно време. Това е така, ако елементите са например не много големи цели числа. “Не много големи” означава, че всяко от тях се побира в една машинна дума, тоест броят на цифрите му в двоична бройна система е няколко десетки. Обаче в тази задача имаме m-цифрени числа (стълбовете на матриците), където m е няколко хиляди, тоест тези числа не се побират в една машинна дума. Ако работим например с 64-битов компютър, то всяко m-цифрено число ще изисква $m / 64 = \Theta(m)$ машинни думи. Следователно времето за едно сравняване на две m-цифрени числа ще бъде $\Theta(m)$, откъдето времевата сложност на пирамидалното сортиране и на сортирането чрез сливане ще стане равна на $\Theta(mn \log n)$, а не на $\Theta(n \log n)$. Очевидно $mn \log n$ е по-голямо по порядък от mn, тоест сортирането по метода на бройните системи е по-бързо от другите споменати сортировки.

СХЕМА ЗА ТОЧКУВАНЕ

10 точки, разпределени, както е указано по-долу.

Задача 1 се оценява с 2 точки — по 1 точка за всяка от следните стъпки:
— развиване на уравнението и получаване на израз с числата на Фибоначи за стъпка № k;
— получаване на отговора на задачата.

Задача 2 носи 5 точки:
а) 2 точки за съставяне на алгоритъм;
б) 1 точка за анализ на времевата сложност на алгоритъма;
в) 2 точки за намиране на търсения среден брой — по 1 точка за всяка от следните стъпки:
— съставяне на рекурентно уравнение;
— решаване на рекурентното уравнение.

Задача 3 се оценява с 3 точки, ако е решена правилно. Частични решения по тази задача не се очакват.