Изпит по "Дизайн и анализ на алгоритми" (поправка — СУ, ФМИ, юни 2016 г.)

Име: _____ ФН:_____ Спец.:____ Курс:___

Задача	1а	1б	1в	2а	2б	3	4а	4б	5	Общо
получени точки										
максимум точки	5	5	10	10	10	20	40	10	20	130

Забележка: За отлична оценка са достатъчни 100 точки.

Задача 1. Да се решат рекурентните уравнения с точност до порядък:

а) $T(n) = T(n-1) + \log\ n$; б) $T(n) = 256\ T\left(\frac{n}{2}\right) + 7n^8$; в) $T(n) = \sum_{k=1}^{n-1} 3\,T(k)$.

Задача 2. Експертна система разполага с m доказателства на n теореми. Всяко доказателство представлява извеждане на една теорема от друга. Системата умее да комбинира известните ѝ доказателства: ако от теоремата X следва теоремата Y, а от теоремата Y следва теоремата Z, то от теоремата X следва теоремата Z.

За всеки от следните проблеми предложете алгоритъм с линейна времева сложност $\Theta(m+n)$:

а) По две дадени теореми системата трябва да определи дали едната теорема следва от другата; ако да, тогава трябва да се състави доказателство чрез комбиниране на минимален брой от известните m доказателства.

б) Множеството от n теореми да се разбие на подмножества, всяко от които се състои от еквивалентни теореми.

Задача 3. Дадени са n отсечки с дължини a_1 , a_2 , ... , a_n . Съставете алгоритъм, който за максимално време $o\left(n^2\right)$ намира три отсечки, образуващи триъгълник (ако има такива).

Задача 4. Математиците в една държава въвели собствена парична единица — непер. От тази валута има само банкноти с номинал 1 , 7 и 10 непера.

а) Съставете алгоритъм, който намира най-малкия брой банкноти, с които може да се изплати сумата n непера точно (без ресто).

За скорост на алгоритъма $O(n)$ ще получите 20 точки, а за скорост $O(1)$ — 40 точки.

б) Разширете алгоритъма така, че да предлага начин на плащане с минимален брой банкноти.

Забележка: Демонстрирайте работата на всички предложени алгоритми при $n = 15$ непера.

Задача 5. Да се докаже, че е NP-трудна следната алгоритмична задача:
"Дадени са целите положителни числа k, a_1 , a_2 , ... , a_n и цялото число $S \geq 0$. Да се провери съществува ли множество $M \subseteq \left\{ 1\ ,\ 2\ ,\ 3\ ,\ \ldots\ ,\ n \right\}$, такова, че $\sum_{i \in M} \left(a_i\right)^k = S$."

РЕШЕНИЯ

Задача 1. а) Развиваме уравнението: $T(n) = T(0) + \sum_{k=1}^{n} \log\, k \asymp \log\,(n!) \asymp n\, \log\, n$.

б) С мастър-теоремата: $k = \log_2 256 = 8, \quad n^k \asymp 7n^8$. Следователно $T(n) = \Theta\big(n^8\, \log\, n\big)$.

в) $T(n) = \sum_{k=1}^{n-1} 3\,T(k)$. Заместваме n със $n-1$: $T(n-1) = \sum_{k=1}^{n-2} 3\,T(k)$. Това уравнение го вадим от оригиналното: $T(n) - T(n-1) = 3\,T(n-1)$, т.е. $T(n) = 4\,T(n-1)$. Развиваме полученото уравнение: $T(n) = 4\,.\,4\,.\,T(n-2) = \ldots = 4^n\,T(0) = \Theta\Big(4^n\Big)$.

Задача 2. Данните могат да бъдат представени чрез насочен граф с n върха и m ребра: върховете съответстват на теоремите, а ребрата — на доказателствата.

а) Извод с най-малък брой стъпки на една теорема от друга съответства на най-къс път от един даден връх до друг даден връх (където дължината на пътя е равна на броя на ребрата). Такъв път може да бъде намерен чрез търсене в ширина.

б) Еквивалентните теореми следват една от друга. Съответстват им силно свързани върхове (всеки от които може да бъде достигнат от другия). На класовете от еквивалентни теореми съответстват компонентите на силна свързаност, които могат да бъдат намерени чрез известния алгоритъм, извършващ обхождане в дълбочина.

И двата предложени алгоритъма имат линейна времева сложност $\Theta(m+n)$.

Задача 4 може да се реши по различни начини.

Първи начин: чрез динамично програмиране.

а) Ако се интересуваме само от минималния общ брой банкноти, е достатъчно да го пазим в един масив `total` и да увеличаваме бройката с единица при добавянето на всяка банкнота.

```
ALG_1A (n)
 1  total[−9 ... n]: array of integers   // минимален общ брой банкноти
 2  for k ← −9 to −1
 3      total[k] ← +∞
 4  total[0] ← 0
 5  for k ← 1 to n
 6      total[k] ← 1 + min( total[k − 1] , total[k − 7] , total[k − 10] )
 7  return total[n]
```

Демонстрация на алгоритъма при $n = 15$ непера:

k	-9	$\ldots$	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`total`	$+\infty$	$\ldots$	$+\infty$	0	1	2	3	4	5	6	1	2	3	1	2	3	4	2	3

Извод: За сумата $n = 15$ непера са нужни най-малко три банкноти.

Сложността на алгоритъма по време и памет е $\Theta(n)$. Количеството допълнителна памет (но не и времето на работа) може да бъде намалено по порядък благодарение на това, че новата стойност в масива зависи най-много от десет предишни. Следователно можем да пазим само тях, т.е. можем да решим задачата с константен брой променливи от примитивен тип.

б) Алгоритъмът може да бъде разширен така, че да връща не само общия брой банкноти, но и броя на банкнотите от всеки вид. За краткост нека функцията `min` връща наредена двойка, чийто първи елемент е най-малката от стойностите, а втори елемент е поредният ѝ номер. С други думи, `min(p,q,r)` връща `(p,1)` или `(q,2)`, или `(r,3)`.

```
Alg_1b (n)
 1  total[−9 ... n]: array of integers   // минимален общ брой банкноти
 2  ones[0 ... n]: array of integers   // оптимален брой банкноти от 1 непер
 3  sevens[0 ... n]: array of integers   // оптимален брой банкноти от 7 непера
 4  for k ← −9 to −1
 5      total[k] ← +∞
 6  total[0] ← 0
 7  ones[0] ← 0
 8  sevens[0] ← 0
 9  for k ← 1 to n
10      ( total[k] , kind ) ← min ( total[k − 1] , total[k − 7] , total[k − 10] )
11      total[k] ← total[k] + 1
12      switch
13          case kind = 1 :   ones[k] ← ones[k − 1] + 1 ;   sevens[k] ← sevens[k − 1]
14          case kind = 2 :   ones[k] ← ones[k − 7] ;        sevens[k] ← sevens[k − 7] + 1
15          case kind = 3 :   ones[k] ← ones[k − 10] ;       sevens[k] ← sevens[k − 10]
16  print "Брой банкноти от 10 непера: ", total[n] − ones[n] − sevens[n]
17  print "Брой банкноти от 7 непера: ", sevens[n]
18  print "Брой банкноти от 1 непер: ", ones[n]
19  return total[n]
```

За коректността на алгоритъма няма значение коя най-малка стойност връща функцията `min` при наличие на няколко такива. За определеност приемаме, че в този случай функцията `min` връща последната по ред най-малка стойност, т.е. алгоритъмът предпочита банкноти с по-голям номинал.

Демонстрация на алгоритъма при $n = 15$ непера:

k	-9	...	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`total`	$+\infty$	...	$+\infty$	0	1	2	3	4	5	6	1	2	3	1	2	3	4	2	3
`ones`				0	1	2	3	4	5	6	0	1	2	0	1	2	3	0	1
`sevens`				0	0	0	0	0	0	0	1	1	1	0	0	0	0	2	2

Извод: За сумата $n = 15$ непера са нужни най-малко три банкноти:
две банкноти от 7 непера и една банкнота от 1 непер.

Сложността на алгоритъма по време и памет е $\Theta(n)$. Количеството допълнителна памет (но не и времето на работа) може да бъде намалено по порядък благодарение на това, че новата стойност във всеки масив зависи най-много от десет предишни. Следователно можем да пазим само тях; така задачата се решава с константен брой променливи от примитивен тип.

Втори начин: Има алгоритъм със сложност $\Theta(1)$ по време и памет. Достатъчни са следните съображения: десет банкноти от 7 непера могат да се заменят със седем банкноти от 10 непера, а седем банкноти от 1 непер могат да бъдат заменени с една банкнота от 7 непера. Следователно всяко решение с най-малък брой банкноти съдържа не повече от девет банкноти от 7 непера и не повече от шест банкноти от 1 непер. Получават се общо $10 \times 7 = 70$ варианта, които могат да бъдат проверени за време, независещо от n.

```
Alg_2 (n)
 1  minTotal ← +∞
 2  for cnt1 ← 0 to 6
 3      for cnt7 ← 0 to 9
 4          r ← n − cnt1 − 7 × cnt7
 5          if r ≥ 0 and r mod 10 = 0
 6              cnt10 ← r / 10
 7              cntTotal ← cnt1 + cnt7 + cnt10
 8              if cntTotal < minTotal
 9                  opt1 ← cnt1
10                  opt7 ← cnt7
11                  opt10 ← cnt10
12                  minTotal ← cntTotal
13  print "Брой банкноти от 10 непера: ", opt10
14  print "Брой банкноти от 7 непера: ", opt7
15  print "Брой банкноти от 1 непер: ", opt1
16  return minTotal
```

При всяко n алгоритъмът изпробва всичките 70 варианта. При $n = 15$ алгоритъмът намира два варианта, когато r е неотрицателно и кратно на 10:
1) две банкноти от 7 непера и една от 1 непер (общо три банкноти);
2) нула банкноти от 7 непера, пет от 1 непер и една от 10 непера (общо шест банкноти).
Първият вариант е по-добър, т.е. нужни са минимум три банкноти.

Трети начин: Въпреки че Alg_2 е най-бърз по порядък (тъй като има константна сложност), той не е най-бързият възможен алгоритъм: константата може да бъде намалена още. Не е нужно при всяко n да бъдат изпробвани всичките 70 случая. Щом оптималното решение съдържа най-много шест банкноти от 1 непер и девет банкноти от 7 непера, то общата стойност на тези банкноти е не повече от $9 \times 7 + 6 \times 1 = 69$ непера. Тогава банкнотите от 10 непера имат обща стойност поне $n - 69$ непера, т.е. поне $n - 69$, закръглено нагоре до най-близкото число, кратно на 10. Остатъкът (не повече от 69 непера) се изплаща по таблица, съдържаща оптималните начини за образуване на сумите от 0 до 69 непера. Тази таблица не се изчислява всеки път. Тя е пресметната еднократно (без значение как) и е записана в кода на алгоритъма.

Пример: При $n = 15$ непера оптималното решение се взима наготово от таблицата: нужни са минимум три банкноти (две от 7 непера и една от 1 непер).

Пример: При $n = 247$ непера: $n - 69 = 178$. Закръглено нагоре до кратно на 10, дава 180 непера, т.е. 18 банкноти от 10 непера. Остатъкът $247 - 180 = 67$ непера се изплаща по таблицата: шест банкноти от 10 непера и една от 7 непера. Получава се минимален брой двайсет и пет банкноти — двайсет и четири от 10 непера и една от 7 непера.

```
ALG_3 (n)
 1  cnt10 ← max(0 ; ⌈(n − 69)/10⌉)
 2  r ← n − 10 × cnt10        // 0 ≤ r ≤ 69
 3  // начало на таблицата
 4  switch
 5      case r = 0 : cnt1 ← 0;  cnt7 ← 0   // cnt10 не се променя
 6      ..........
 7      case r = 15 : cnt1 ← 1;  cnt7 ← 2   // cnt10 не се променя
 8      ..........
 9      case r = 67 : cnt1 ← 0;  cnt7 ← 1;  cnt10 ← 6 + cnt10
10      ..........
11      case r = 69 : cnt1 ← 2;  cnt7 ← 1;  cnt10 ← 6 + cnt10
12  // край на таблицата
13  cntTotal ← cnt10 + cnt7 + cnt1
14  print "Брой банкноти от 10 непера: ", cnt10
15  print "Брой банкноти от  7  непера: ", cnt7
16  print "Брой банкноти от  1  непер: ", cnt1
17  return cntTotal
```

Този алгоритъм е по-бърз от ALG_2 (макар и не по порядък), защото при всяко n разглежда само един случай, а не всичките седемдесет. Недостатък на ALG_3 е дългият програмен код.

Четвъртият и петият начин са оптимизации съответно на втория и третия алгоритъм. С малко по-внимателни разсъждения можем да намалим броя на случаите доста под седемдесет. Например няма смисъл да използваме повече от две банкноти от 7 непера, защото три банкноти от 7 непера могат да се заменят с две банкноти от 10 непера и една банкнота от 1 непер: $3 \times 7 = 21 = 2 \times 10 + 1 \times 1$. Така едно оптимално решение (с три банкноти) се заменя с друго оптимално решение (със същия брой банкноти). Ограничавайки до 0, 1 или 2 броя на банкнотите от седем непера, губим някои от оптималните решения, но не всички. Това е допустимо, понеже търсим едно оптимално решение, а не всички такива.

Можем да ограничим и броя на банкнотите от един непер. Броят им е не повече от шест (защото седем банкноти от един непер могат да се заменят с една банкнота от седем непера), и то само ако няма банкноти от седем непера. А ако има, то броят на банкнотите от един непер е максимум две (защото три банкноти от един непер и една от седем непера могат да се заменят с една банкнота от десет непера).

Накратко: Ако cnt1 и cnt7 са съответно броят на банкнотите от един и седем непера, то за всяко n съществува оптимално решение измежду следните наредени двойки (cnt7; cnt1): (0; 0), (0; 1), (0; 2), (0; 3), (0; 4), (0; 5), (0; 6); (1; 0), (1; 1), (1; 2); (2; 0), (2; 1), (2; 2).

Алгоритъмът ALG_4 е вариант на ALG_2, променен така, че да изпробва само тези тринайсет случая вместо всичките седемдесет.

Алгоритъмът ALG_5 е оптимизация на ALG_3 с таблица от 0 до 16 вместо от 0 до 69. Границата 16 се определя от наредената двойка (2; 2): $2 \times 7 + 2 \times 1 = 16$.

Шести начин: чрез поправен алчен алгоритъм.

В желанието си да съставим бързодействащ и къс програмен код е естествено да изпробваме алчен алгоритъм:
1) Използваме колкото може повече банкноти от 10 непера.
2) Като остане сума, по-малка от 10 непера, използваме (ако може) една банкнота от 7 непера.
3) Остатъка (ако има такъв) изплащаме с банкноти от 1 непер.

За съжаление, този алгоритъм е грешен. Например при $n = 15$ той намира представяне с шест банкноти (10+1+1+1+1+1 = 15) вместо с минималния брой три (7+7+1 = 15).

За щастие, алгоритъмът може да бъде поправен. За да се ориентираме, нека разгледаме базовите случаи. Според доказаното по-горе е достатъчно да разгледаме сумите от 0 до 16 непера. За удобство при следващите разсъждения ще разгледаме сумите от 0 до 20 непера.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Минимален общ брой банкноти	0	1	2	3	4	5	6	1	2	3	1	2	3	4	2	3	4	2	3	4	2
Брой банкноти от 10 непера	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	2
Брой банкноти от 7 непера	0	0	0	0	0	0	0	1	1	1	0	0	0	0	2	2	2	1	1	1	0
Брой банкноти от 1 непер	0	1	2	3	4	5	6	0	1	2	0	1	2	3	0	1	2	0	1	2	0

Не е трудно да се види, че от случаите, изброени в таблицата, алчният алгоритъм греши само при $n = 14$, 15 и 16. Понеже използва банкноти от 10 непера, докогато е възможно, той греши при всички стойности на n, завършващи на 4, 5 или 6 (и различни от 4, 5 и 6). В останалите случаи алчният алгоритъм работи коректно.

Следователно алчният алгоритъм може да бъде поправен така: ако числото n завършва на някоя от цифрите 4, 5 и 6 (и е различно от 4, 5 и 6), то трябва да използваме една банкнота от 10 непера по-малко, а вместо нея си служим с две банкноти от 7 непера. Остатъка (ако има) изплащаме с банкноти от 1 непер.

```
Alg_6 (n)
 1  cnt10 ← ⌊n/10⌋   // използваме колкото може повече банкноти от 10 непера
 2  if cnt10 > 0 and n завършва на цифрата 4, 5 или 6
 3        cnt10 ← cnt10 − 1   // поправка
 4  r ← n − 10 × cnt10   // оставаща сума
 5  cnt7 ← ⌊r/7⌋   // използваме колкото може повече банкноти от 7 непера
 6  cnt1 ← r − 7 × cnt7   // остатъка изплащаме с банкноти от 1 непер
 7  print "Брой банкноти от 10 непера: ", cnt10
 8  print "Брой банкноти от 7 непера: ", cnt7
 9  print "Брой банкноти от 1 непер: ", cnt1
10  return cnt10 + cnt7 + cnt1
```

При $n = 15$ отначало cnt10 = 1, но след поправката cnt10 става 0, така че сумата 15 непера се изплаща само с банкноти от 1 и 7 непера: $2 \times 7 + 1 \times 1$ — общо три банкноти, което е минимумът.

Когато n завършва на 0, 1, 2 или 3, няма поправка: използваме колкото може повече банкноти от 10 непера. В тези случаи остатъкът е по-малък от 7 непера, затова се изплаща с банкноти от 1 непер.

Примери:
$253 = 250 + 3 = 25 \times 10 + 3 \times 1$ — общо 28 банкноти.
$461 = 460 + 1 = 46 \times 10 + 1 \times 1$ — общо 47 банкноти.

Когато n завършва на 7, 8 или 9, пак няма поправка: използваме колкото може повече банкноти от 10 непера. Сега остатъкът е поне 7 непера, затова освен банкнотите от 1 непер има и една от 7 непера.

Примери:
$519 = 510 + 9 = 51 \times 10 + 1 \times 7 + 2 \times 1$ — общо 54 банкноти.
$367 = 360 + 7 = 36 \times 10 + 1 \times 7$ — общо 37 банкноти.

Когато n завършва на 4, 5 или 6, има поправка: банкнотите от 10 непера са с една по-малко. Вместо нея използваме две банкноти от 7 непера. Евентуалния остатък изплащаме с банкноти от 1 непер.

Примери:
$106 = 90 + 16 = 9 \times 10 + 2 \times 7 + 2 \times 1$ — общо 13 банкноти.
$875 = 860 + 15 = 86 \times 10 + 2 \times 7 + 1 \times 1$ — общо 89 банкноти.

Алгоритъмът Alg_6 бе предложен от хон. ас. Стефан Фотев. Този алгоритъм е оптимален: има константна сложност, константата е минимална (алгоритъмът не разглежда излишни случаи) и програмният код е възможно най-къс.

Задача 5. В частния случай $k = 1$ се получава известната NP-трудна задача SubsetSum. Редукцията е полиномиална, защото присвояването $k = 1$ се извършва за константно време. Тоест разглежданата алгоритмична задача е обобщение на NP-трудната задача SubsetSum и значи също е NP-трудна.

Задача 3. Търсим три отсечки, за които сборът на двете по-малки да е по-голям от третата.

```
FindTrigon(A[1...n])
1  Sort(A)
2  for j ← 2 to n − 1
3      if A[j − 1] + A[j] > A[j + 1]
4          print "Страни на триъгълник: ", A[j − 1], A[j], A[j + 1]
5          return
6  print "Няма триъгълник"
```

След сортирането търсим индекси i, j, k, удовлетворяващи неравенствата $1 \leq i < j < k \leq n$ и $A[i] + A[j] > A[k]$. Следователно i трябва да е максимално, а k — минимално. Тоест достатъчно е да разгледаме само случая $i = j - 1$, $k = j + 1$.

Времевата сложност на сортирането е $\Theta(n \log n)$, на цикъла е $\Theta(n)$, на целия алгоритъм — сборът $\Theta(n \log n) + \Theta(n) = \Theta(n \log n) = o(n^2)$.