Име: ..... ФN: ..... Специалност: ..... Гр.: ....

Задача	1	2	3	4	5	6	Общо
получени точки							
от максимално	20	20	20	20	20	20	120

За максимална оценка са достатъчни 100 *точки. Ако получите повече от* 100 *точки, това ще бъде бонус, който ще бъде отразен в таблицата с оценките.*

Зад. 1 Функциите, които разглеждаме, са асимптотично положителни функции с домейн $\mathbb{N}$. Докажете или опровергайте всяко от следните твърдения. Дайте подробна аргументация на отговорите си, като започнете от дефинициите на асимптотичните нотации.

10 т. а) Ако $f_1(n) = O(g_1(n))$ и $f_2(n) = O(g_2(n))$, то $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

10 т. б) Ако $f_1(n) = O(g_1(n))$ и $f_2(n) = \Omega(g_2(n))$, то $f_1(n) + g_2(n) = O(f_2(n) + g_1(n))$.

Зад. 2 Наредете по асимптотично нарастване следните осем функции. Обосновете отговорите си кратко. Напишете в явен вид самата наредба.

$$\frac{(\lg n)^3}{\lg\lg n}, \qquad n^3, \qquad 4^{n-\sqrt{n}}, \qquad (n+1)^n$$
$$\lg(n^9), \qquad \big((\lg n)^2\big)(\lg\lg n), \qquad 3^{n+\sqrt{n}}, \qquad n^n$$

Зад. 3 *Мода* на масив от данни се нарича всеки най-често срещан елемент в масива. Забележете, че модата не е непременно уникална; примерно, ако нито два елемента не са еднакви, то всеки елемент е мода. Можете да ползвате функция Sort с гарантирана коректност и сложност $O(n \lg n)$, без да пишете нейния псевдокод.

10 т. **а)** Напишете алгоритъм AlgM, който намира мода на масив $A[1, 2, \ldots, n]$ от цели числа. AlgM трябва да е написан на псевдокод и да работи във време $O(n \lg n)$.

10 т. **б)** Докажете коректността на AlgM.

бонус 10 т. **в)** Докажете, че всеки алгоритъм, намиращ мода и базиран върху директни сравнения, има долна граница на сложността $\Omega(n \lg n)$.

Зад. 4 Масивът $A[1, 2, \ldots, n]$ съдържа m инверсии. Докажете, че алгоритъмът InsertionSort сортира A за $\Theta(m+n)$ стъпки. *Инверсия* е всяка двойка индекси (i, j), такива че $1 \le i < j \le n$ и $A[i] > A[j]$.

Зад. 5 Нека MySort е произволен сортиращ алгоритъм, базиран върху директни сравнения, който работи над вход с големина $n > 2$. Докажете или опровергайте всяко от следните твърдения:

6 т. **а)** Всеки елемент на масива участва в поне едно сравнение, извършено от MySort.

7 т. **б)** Всеки елемент на масива участва в поне две сравнения, извършени от MySort.

7 т. **в)** Съществува елемент на масива, който участва в поне две сравнения, извършени от MySort.

Зад. 6 Разгледайте алгоритъм Alg-2. Определете стойността на изхода s като функция на n. Вашият отговор трябва да бъде формула над основните аритметични действия, която се оценява във време $\Theta(1)$. Можете да ползвате наготово формулите $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ и $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$.

```
Alg-2(n: int)
 1  s ← 0
 2  for i ← 1 to n
 3      for j ← i to n
 4          for k ← j to n
 5              s ← s + 1
 6  return s
```

Решение на зад. 1:

а) Твърдението е вярно. Съгласно дефиницията на $O(g_1(n))$ съществуват константи $n_1 > 0$ и $c_1 > 0$, такива че $\forall n > n_1$ е изпълнено неравенството $0 < f_1(n) \leq c_1.g_1(n)$. С аналогични разсъждения за $O(g_2(n))$ извеждаме съществуването на константи $n_2 > 0$ и $c_2 > 0$, такива, че за $\forall n > n_2$ е изпълнено неравенството $0 < f_2(n) \leq c_2.g_2(n)$. Нека $n_3 = \max(n_1, n_2)$ и $d = \max(c_1, c_2)$. Тогава за всички $n > n_3$ са изпълнени двете неравенства

$$0 < f_1(n) \leq d.g_1(n)$$
$$0 < f_2(n) \leq d.g_2(n)$$

Събираме ги и получаваме $0 < f_1(n) + f_2(n) \leq d(g_1(n) + g_2(n))$, от което следва $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

б) Твърдението е вярно. Съгласно дефиницията на $\Omega(g_2(n))$ съществуват константи $n_2 > 0$ и $c_2 > 0$, такива че $\forall n > n_2$ е изпълнено неравенството $0 < c_2.g_2(n) \leq f_2(n)$, което е еквивалентно на $0 < g_2(n) \leq \frac{f_2(n)}{c_2}$. Последното неравенство ни дава основание да твърдим, че $g_2(n) = O(f_2(n))$.

Прилагаме твърдението от условие а) към двойката зависимости $f_1(n) = O(g_1(n))$ и $g_2(n) = O(f_2(n))$. От верността на а) следва $f_1(n) + g_2(n) = O(f_2(n) + g_1(n))$.

Решение на зад. 3:

а) Една възможна реализация на алгоритъма ALGM е:

```
ALGM(A[1...n])
 1  SORT(A)
 2  i ← 1
 3  m ← 0
 4  mode ← A[1]
 5  while i ≤ n do
 6      t ← i + 1
 7      while (t ≤ n) ∧ (A[i] = A[t]) do
 8          t ← t + 1
 9      if t − i > m
10          m ← t − i
11          mode ← A[i]
12      i ← t
13  return mode
```

б) Инварианта за външния **while** цикъл (редове 5–12) е:

Всеки път, когато изпълнението е на ред 10:

1. Интервалът $A[i \ldots t-1]$ съдържа еднакви елементи.
2. В сила са следните неравенства:

$$\forall x \in A[1 \ldots i-1]\ \forall y \in A[i \ldots t-1] : x < y$$
$$\forall y \in A[i \ldots t-1]\ \forall z \in A[t \ldots n] : y < z$$

3. Променливата mode съдържа мода на подмасива $A[1 \ldots i-1]$.

в) Изчислителната задача УНИКАЛНОСТ НА ЕЛЕМЕНТИ—дали в даден n-елементен масив елементите са два по два различни—се свежда до едно извикване на алгоритъм за намиране на мода на масива, последван от проверка дали изчислената мода се среща само веднъж. Очевидно въпросната проверка може да бъде направена в $\Theta(n)$. Да допуснем, че разполагаме с алгоритъм за мода със сложност по време $f(n)$. Тогава задачата УНИКАЛНОСТ НА ЕЛЕМЕНТИ можем да решим за време $f(n) + c_0 n$. Известно е, че тази задача има дорна граница $\Omega(n \lg n)$, следователно $f(n) + c_0 n = \Omega(n \lg n)$.

От дефиницията на класа $\Omega(n \lg n)$ следва, че съществуват константи $c_1 > 0$ и n_0 такива, че за $n > n_0$ е изпълнено неравенството $f(n) + c_0 n \geq c_1 n \lg n$. То е еквивалентно на $f(n) \geq c_1 n \lg n - c_0 n$, което пък можем да запишем като асимптотично неравенство $f(n) \succeq c_1 n \lg n - c_0 n$.

Очевидно $c_1 n \lg n \succ c_0 n$, откъдето следва $c_1 n \lg n - c_0 n \asymp c_1 n \lg n$.

От веригата асимптотични равенства и неравенстава $f(n) \succeq c_1 n \lg n - c_0 n \asymp c_1 n \lg n \asymp n \lg n$ следва $f(n) \succeq n \lg n$, което е търсената долна граница.

Решение на зад. 4: Една възможна реализация на алгоритъма InsertionSort е:

```
InsertionSort(A[1...n])
1  for i ← 2 to n
2      j ← i − 1
3      while (j > 0) ∧ (A[j] > A[j + 1]) do
4          swap(A[j], A[j + 1])
5          j ← j − 1
```

Очевидно всяко изпълнение на ред 4 намалява броя на инверсиите с единица. Следователно, тялото на цикъла **while** се изпълнява точно m пъти общо, за цялото изпълнение на InsertionSort. За всяка стойност на i, ред 3 се изпълнява, докато има инверсии в подмасива $A[1 \dots i]$ и още веднъж, когато инверсиите са свършили, за да се напусне цикълът. Сумарно за цялата сортировка това ще даде $m+n-1$ изпълнения на ред 3. Редове 1 и 2 се изпълняват $n - 1$ пъти.

Сумираме горните оценки и получаваме време за изпълнение $\Theta(m + n)$.

Решение на зад. 5:

а) Твърдението е вярно, защото ако има елемент, който не участва в сравнение, алгоритъмът няма да може да определи мястото му в сортирания масив.

б) Твърдението е невярно. Нека подадем на алгоритъма InsertionSort вече сортиран масив. Минималният и максималният елементи ще участват в по едно единствено сравнение.

в) Твърдението е вярно, защото всеки сортиращ алгоритъм трябва да извърши поне едно сравнение върху всяка ненаредена двойка от съседни по големина елементи. Да допуснем противното. Нека сортиращият алгоритъм, базиран на директни сравнения, се казва Sort. Да разгледаме произволен вход $\mathcal{A} = (a_1, a_2, \dots, a_n)$. Нека няма два еднакви елементи във входа. Допускането ни е, че има a_i, a_j, такива че $i \neq j$, a_i и a_j са съседни по големина, и Sort не ги сравнява. Без ограничение на общността, нека $a_i < a_j$. Щом Sort е коректен сортиращ алгоритъм, то Sort($\mathcal{A}$) връща масив, в който a_i и a_j са съседи по наредба, и по-точно a_i е непосредствено вляво от a_j. Тоест, изходът е $\dots, a_i, a_j, \dots$.

Нека $\mathcal{A}'$ е друг вход, който се получава от $\mathcal{A}$ чрез swap на a_i и a_j. Да разгледаме Sort($\mathcal{A}'$). Но резултатът от работата на Sort($\mathcal{A}'$) ще бъде точно същият като резултата от работата на Sort($\mathcal{A}$), понеже резултатът от всяко сравнение ще е точно същият. Тогава изходът пак ще бъде $\dots, a_i, a_j, \dots$, което по отношение на $\mathcal{A}'$ е некоректно—сега $a_i > a_j$. И така, допускането е опровергано.

Следователно, всеки елемент от входа, който не е екстремен по големина, ще участва в поне две сравнения: с елемента непосредствено преди него по големина, и с елемента непосредствено след него по големина.

Решение на зад. 6:

$$\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n}1=\sum_{i=1}^{n}\sum_{j=i}^{n}(n-j+1)=\sum_{i=1}^{n}\sum_{j=i}^{n}(n+1)-\sum_{i=1}^{n}\sum_{j=i}^{n}j=$$

$$\sum_{i=1}^{n}(n+1)\sum_{j=i}^{n}1-\sum_{i=1}^{n}\left(\sum_{j=1}^{n}j-\sum_{j=1}^{i-1}j\right)=\sum_{i=1}^{n}(n+1)(n-i+1)-\sum_{i=1}^{n}\left(\frac{n(n+1)}{2}-\frac{i(i-1)}{2}\right)=$$

$$\sum_{i=1}^{n}(n^2+2n+1-(n+1)i)-\frac{1}{2}\left(\sum_{i=1}^{n}n(n+1)-\sum_{i=1}^{n}i^2+\sum_{i=1}^{n}i\right)=$$

$$\sum_{i=1}^{n}(n+1)^2-(n+1)\sum_{i=1}^{n}i-\frac{1}{2}\left(n^2(n+1)-\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right)=$$

$$n(n+1)^2-\frac{n(n+1)^2}{2}-\frac{n^2(n+1)}{2}+\frac{n(n+1)(2n+1)}{12}-\frac{n(n+1)}{4}=$$

$$\frac{n(n+1)^2}{2}-\frac{n^2(n+1)}{2}+\frac{n(n+1)(2n+1)}{12}-\frac{n(n+1)}{4}=$$

$$\frac{n(n+1)}{2}\left(n+1-n+\frac{2n+1}{6}-\frac{1}{2}\right)=\frac{n(n+1)}{2}\left(\frac{1}{2}+\frac{2n+1}{6}\right)=\frac{n(n+1)}{2}\frac{2n+4}{6}=$$

$$\frac{n(n+1)(n+2)}{6}$$