ДАА Семинар 3

Сложност по време на итеративни алгоритми

- Често използвани суми
  - $\sum_{i=1}^{n} 1 = n$
  - $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \simeq n^2$
  - $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \simeq n^3$
  - $\sum_{i=1}^{n} \frac{1}{i} \simeq \log n$
  - $\sum_{i=1}^{n} \log i = \log n! \simeq n \log n$
  - $\sum_{i=a}^{b} 1 = b - a + 1$
  - $\sum_{i=1}^{n} i^{\alpha} = \begin{cases} \theta(n^{\alpha+1}), & \alpha > -1 \\ \theta(\log n), & \alpha = -1 \\ \theta(1), & \alpha < -1 \end{cases}$
  - $\sum_{i=1, i \bmod k = 0}^{n} 1 = \left\lfloor \frac{n}{k} \right\rfloor \simeq \frac{n}{k}$

- Оптимален алгоритъм

Ако е доказано/докажем, че кой да е алгоритъм Alg, решаващ изчислителна задача Π, не може да работи „по-бързо" от $f(n)$, то казваме, че всеки алгоритъм за Π, работещ във време $O(f(n))$, е оптимален за Π.

↳ В повечето случаи не гледаме сложността по памет : времето е по-ценен ресурс

- Може да изчислим сложността по време на даден алгоритъм, събирайки броя изпълнения на всички елементарни инструкции - прекалено сложно, не се интересуваме от толкова подробен анализ в повечето случаи - ще търсим асимптотичната сложност

Няма единствен метод за намирането на сложността (наръка), приложим за всеки алгоритъм.

// Изчисляване на брой итерации
- в някои случай асимптотичната сложност по време е пропорционална на броя минавания през for/while циклите (някои сортиращи алгоритми, умножение на матрици, търсене...)

// гледаме инструкцията, която се изчислява най-много пъти и намираме този брой

// на всеки цикъл съпоставяме сума Σ

// Пример:

1. count ← 0
2. for i ← low to high
3. count ← count + 1

↳ съпоставяне следната сума:

$$\sum_{i=low}^{high} \theta(1) = \theta(high - low + 1)$$

- променливата-итератор на цикъла става итератор на сумата
- сложените суми съответстват на сложени цикли

зад① Да се изчисли сложността по време на следния алгоритъм

Alg. Count1
Input: $n = k^2$, $k \in \mathbb{N}$
Output: $\sum_{i=1}^{j} i$ for each perfect square j between 1 and n

1. $k \leftarrow \sqrt{n}$
2. for j = 1 to k
3. sum[j] ← 0
4. for i ← 1 to j^2
5. sum[j] ← sum[j] + i
6. return sum[1...k]

→ Пример за $n = 36 = 6^2$

1	10	45	136	325	666
1	2	3	4	5	6

- считаме, че $\sqrt{n}$ се изчислява за $\theta(1)$

Решение:
Извън работата на цикъла Count1 върши константна работа. Разглеждане цикъла на ред 2. Той се изпълнява $k = \sqrt{n}$ пъти, но всяка негова итерация НЕ се извършва за константно време - имаме сложен цикъл на ред 4, като се изпълнява j^2 пъти. Следователно за сложността $T(n)$ получаваме:

$$T(n) = \theta(1) + \sum_{j=1}^{k} \sum_{i=1}^{j^2} \theta(1) = \theta(1) + \sum_{j=1}^{k} \theta(j^2) = \theta(1) + \theta\left(\frac{k(k+1)(2k+1)}{2}\right) = \theta(k^3) = \theta(n^{1.5})$$

Зад ② Да се изчисли сложността по време на следния алгоритъм

```
Count2 (n∈ℕ⁺)
1. count ← 0
2. for i ← 1 to n
3.     m ← ⌊n/i⌋
4.     for j ← 1 to m
5.         count ← count + 1
6. return count
```

// count изчислява броя изпълнения на ред 5

Решение: Извън вложените цикли Count2 извършва константна работа

$$\Rightarrow T(n) = \sum_{i=1}^{n}\sum_{j=1}^{m} 1 = \sum_{i=1}^{n} m = \sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor \asymp \sum_{i=1}^{n} \frac{n}{i} = n\sum_{i=1}^{n} = n\log n$$

$$\Rightarrow T(n) \asymp n\log n$$

Зад ③ - от семестриално 2014г.

Да се разгледа алгоритъма Alg. Да се определи стойността на изхода s като функция на n. Отговорът трябва да бъде формула над основните аритметични действия, която се оценява във време $\Theta(1)$

```
Alg (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:

↓ начин: $$s = \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} 1 = \ldots = \frac{(n+1)(n+2)n}{6}$$

↓↓ начин: Началната стойност на s е 0 и се увеличава с 1 за всеки път когато се изпълнява ред 5. Ред 5 се достига когато условията за навлизане в трите цикъла на ред 2, 3, 4 са едновременно изпълнени, тоест когато стойностите на i, j и k удовлетворяват следните неравенства:

$$1 \le i \le j \le k \le n \quad (\Delta)$$

Всяко триелементно мултимножество над $\{1,\ldots,n\}$ определя еднозначно наредено решение (i,j,k) на (Δ) и всяко решение на (Δ) определя еднозначно триелементно мултимножество над $\{1,\ldots,n\}$.

Следователно броят решения на (4) съвпада с броя на триелементните мултимножества над $\{1,\dots,n\}$, които е $\binom{(n-1)+3}{3} = \frac{(n+2)!}{3!\,(n+1)!} = \frac{(n+2)(n+1)\,n}{6}$,

колкото и пъти се изпълнява ред 5 $\Rightarrow$ $S = \frac{(n+2)(n+1)n}{6}$; ч.т.

// Обобщение

Колко е стойността на k след изпълнение на следния алгоритъм

```
1. k ← 0
2 for i1 ← 1 to n
3    for i2 ← 1 to i1
.
.
.
m+1.        for im ← 1 to im-1
m+2.            k ← k+1
```

Добавя се 1 към ст-стта на k всеки път когато цикълът се обходи с редица $i_1, i_2, \dots, i_m$: $1 \le i_m \le i_{m-1} \le \dots \le i_1 \le n \Rightarrow k = \binom{(n-1)+m}{m}$

// Заб. дали че е $i_t \leftarrow 1$ to i_{t-1} или $i_t \leftarrow i_{t-1}$ to n няма значение

зад (4) - семестриално контролно

Да се намери асимптотичната сложност по време като функция на n.

Бонус: да се намери стойността на a

```
int f(int n)
{
1. int a = 0;
2. for(i=0 ; i<=2*n ; i+=2)
3.     for(j=0 ; j<=i ; j+=2)
4.         a++;
5. return a;
}
```

Решение:

Броят изпълнения на ред 4. задава асимптотичната сложност на функцията. Всеко изпълнение на този ред увеличава a с единица. Директно че намерим стойността на a на ред 5 (при достигане на ред 5)

Ясно:

Разглеждане външния цикъл - на ред 2. В хода на работата на f i че заема следните стойности: $i = 0, 2, 4, 6, 8, \dots, 2n$, тоест $i \in \{0,2,4,6,8,\dots,2n\}$.

Тоест външният цикъл че се изпълни $(n+1)$ пъти

↳ $(|\{0,2,4,6,8,\dots,2n\}| = |\{0,1,2,3,4,\dots,n\}|$

↳ кардиналността

Нека $i = 2k$, $k \in \{0,\dots,n\}$ е конкретна стойност на i при някое навлизане на външния цикъл на ред 2.

За фиксираната стойност на i вътрешния цикъл ще се изпълни точно (k+1) пъти ($j \in \{0, 2, 4, 6, 8, \ldots, 2k\}$)

$\Rightarrow$ общият брой изпълнения на ред 4. е: $(0+1) + (1+1) + (2+1) + \ldots + (n+1) =$

$= \sum_{k=0}^{n} k+1 = (n+1) + \frac{n(n+1)}{2}$, което число е и стойността на a при достигане на ред 5.

II начин:

Стойността на a на ред 5 се определя от следната сложена сума:

$$\sum_{\substack{i=0 \\ i+=2}}^{2n} \sum_{\substack{j=0 \\ j+=2}}^{i} 1 = \sum_{\substack{i=0 \\ i+=2}}^{2n} \left(\frac{i}{2}+1\right) = \sum_{\substack{i=0 \\ i+=2}}^{2n} 1 + \sum_{\substack{i=0 \\ i+=2}}^{2n} \frac{i}{2} = (n+1) + \sum_{k=0}^{n} k = (n+1) + \frac{n(n+1)}{2} \quad \square.$$

(Δ) (Δ) (Δ) (Δ)

зад ⑤ Да се определи асимптотичното време, за което се изпълняват следните цикли:

```
1. for(int i=1; i<=n; ++i)
2.     for(int j=1; j<=n; ++j)
3.         if (i==j) for(int k=1; k<=n; ++k) ++a;
```

Решение:

Разбиваме изпълнението на програмата на два случая: I сл. при $i \neq j$

II сл. при $i = j$.

II сл. $i = j$

Когато $i=j$ ред 3 се изпълнява за линейна сложност $\theta(n)$. Колко пъти ще е изпълнено условието $i==j$? Ще е изпълнено при $i=j=1$; $i=j=2$; $i=j=3, \ldots, i=j=n$, тоест точно n пъти ред 3 ще работи във време $\theta(n)$

Времето за работа в този случай е $n.\theta(n) \asymp n^2$

I сл. $i \neq j$

Когато $i \neq j$ ред 3 се изпълнява за константна сложност $\theta(1)$. Колко пъти ще е изпълнено $i \neq j$? Общият брой стойности на i и j, които удовлетворяват условията за навлизане в циклите на ред 1 и 2 е n^2. От тях накаме случаите когато $i==j$ е истина $\Rightarrow (n^2-n)$ пъти ред 3 ще работи за константно време.

Времето за работа в този случай е $(n^2-n)\theta(1) \asymp n^2$

$\Rightarrow$ Общото време за работа е $\asymp n^2 + n^2 = 2n^2 \asymp n^2 \Rightarrow \theta(n^2)$

Пример

Анализ на BinarySearch

Alg. BinarySearch

Вход. Масив $A[1...n]$, сортиран ненамаляващо, елемент X

Изход. Позиция j, ако $A[j] == X$, $j \in \{1,...,n\}$ и 0 в противен случай

1. low ← 1; high ← n; j ← 0
2. while (low ≤ high) and (j == 0)
3. mid ← ⌊(low + high)/2⌋
4. if X == A[mid] then j ← mid
5. else if X < A[mid] then high ← mid - 1
6. else low ← mid + 1
7. end while
8. return j.

- минималният брой сравнения е 1 - когато X е в 'средата' на A
- максималният брой сравнения се постига когато $X >= A[n]$

За mid ако големината е четна взимаме левия елемент от средата и дясната част на масива е с по-голям размер от лявата

Търсим максималния брой сравнения при големина n. Нека $X >= A[n]$

Изчисляване големината на останалия масив във втората итерация на while-а: 1(а) n е четно, тогава големината на $A[mid+1,...,n]$ е $\frac{n}{2}$

2(а) n е нечетно, тогава големината на $A[mid+1,...n]$ е точно $\frac{(n-1)}{2}$.

И в двата случая големината е точно $\lfloor \frac{n}{2} \rfloor$.

Аналогично, големината на третата итерация ще е $\lfloor \lfloor \frac{n}{2} \rfloor / 2 \rfloor = \lfloor \frac{n}{4} \rfloor$

По време на i-тата итерация големината ще е $\lfloor \frac{n}{2^{i-1}} \rfloor$

В последната итерация, или X е намерен, или големината е станала единица, което се случи първо.

Максималният брой итерации j се постига когато

(големината е станала 1 за първи път)

$$\lfloor \frac{n}{2^{j-1}} \rfloor = 1 \overset{def \lfloor \rfloor}{\Leftrightarrow}$$

$$1 \le n/2^{j-1} < 2 \Leftrightarrow$$

$$2^{j-1} \le n < 2^j \Leftrightarrow / \log_2$$

$$j-1 \le \log n < j \overset{j \in \mathbb{N}}{\Rightarrow} j = \lfloor \log n \rfloor + 1 = \Theta(\log n)$$

За упражнение:

① Да се изчисли стойността на count след приключване на изпълнение на следния алгоритъм:

```
AlgX (n=2^k, k∈ℕ^+)
1. count ← 0
2. i ← 1
3. while i≤n
4.     for j ← 1 to i
5.         count ← count + 1
7.     i ← 2i
9. return count
```

// Един възможен начин: $i \in \{1,2,4,..,n\} \equiv i \in \{2^0, 2^1, 2^2, .., 2^k=n\}$, тоест $i=2^r$, $r \in \{0,..,k\}$, тогава $r=\log i$. Използвайки положената променлива: $\sum_{r=0}^{k} \sum_{j=1}^{i} 1 = \ldots = \theta(n)$

↳ да се напише подробно

② AlgY ($n = 2^{2^k}$, $k \in \mathbb{N}^+$)

```
1. count ← 0
2. for i ← 1 to n
3.     j ← 2
4.     while j≤n
5.         j ← j^2
6.         count ← count + 1
9. return count
```

↱ да се напише подробно

// „—" $j \in \{2, 2^2; 2^{2^2}=2^4; (2^{2^2})^2=2^8; \ldots, 2^{2^k}\} \equiv j \in \{2^{2^0}; 2^{2^1}; 2^{2^2}; 2^{2^3}; \ldots; 2^{2^{k-1}}; 2^{2^k}\} \Rightarrow j = 2^{2^r}$

тоест $r = \log\log j$: $\sum_{i=1}^{n} \sum_{r=0}^{k} 1 = \ldots = n(\log\log n + 1) = \theta(n.\log\log n)$