УДК: 004.42, 330.4 DOI 10.33111/mise.103.15

Потапенко С. Д.,
к.е.н., доцент кафедри системного аналізу та кібербезпеки,
КНЕУ ім. В. Гетьмана

Potapenko S. D.,
Candidate of Economic Science, Associate Professor at
the Department of Systems Analysis and Cybersecurity,
KNEU named after V. Hetman

ПРО ДЕЯКІ АСПЕКТИ АСИМПТОТИЧНОГО ОЦІНЮВАННЯ СКЛАДНОСТІ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ

ABOUT SOME ASPECTS OF ASYMPTOTIC COMPLEXITY ASSESSMENT OF COMPUTING ALGORITHMS

Анотація. *Асимптотична оцінка математичних функцій є методом спрощеного опису визначення граничних значень їх поведінки. Для оцінки кількості ітерацій або кроків виконання алгоритмів найчастіше використовують наближену оцінку верхньої межі значень відповідних характеристик алгоритмів. У даній публікації пропонується розглянути процес аналітичного визначення мажорант за зміненою схемою. Застосування зазначеної схеми дасть змогу підвищити точність та обґрунтованість аналітичної форми визначених мажорант. Для опису математичного апарату асимптотичної оцінки складності обчислювальних алгоритмів запропоновано систему з однієї леми та дванадцяти теорем. Як приклад застосування проведено асимптотичну оцінку складності деяких обчислювальних алгоритмів. У статті визначено лему про наявність багатьох мажорант, теорему про складність постійного обчислення, теорему про ступінь залежності складності обчислення, теорему про додавання константи до числа обчислень, теорема про кратність трудомісткості обчислень, теорема про транзитивність трудомісткості обчислень, теорема про додавання обсягів обчислень, теорема про додавання обчислювальної трудомісткості, теорема про додавання постійної обчислювальної складності, теорема про множення обчислювальної складності, теорема про поліноміальну обчислювальну складність, теорема про експоненціально-поліноміальну обчислювальну складність, теорема про логарифмічну обчислювальну складність. Розглянутий матеріал наочно демонструє практичні особливості визначення мажорант за зміненою схемою та показує його практичне значення. Наведені приклади визначення обчислювальної складності одних відомих алгоритмів можуть бути покладені в основу визначення обчислювальної складності інших алгоритмів. Крім того, наведену систему теорем можна розширити та застосувати для оцінки комплексного застосування алгоритмів, які доповнюють або є частинами один одного.*

Ключові слова. *Асимптотична оцінка, складність алгоритму, мажоранта, синтез алгоритмів.*

Abstract. *Asymptotic evaluation of mathematical functions is a method of simplified description of determining the limit values of their behavior. To estimate the number of iterations or steps of execution of algorithms, an approximate estimation of the upper limit of the values of the corresponding characteristics of the algorithms is most often used. In this publication, it is proposed to consider the process of analytical determination of majorants according to the changed scheme. The application of the specified scheme will make it possible to increase the accuracy and justification of the analytical form of the determined majorants. A system of one lemma and twelve theorems is proposed to describe the mathematical apparatus for asymptotic assessment of the complexity of computational algorithms. As an example of application, an asymptotic assessment of the complexity of some computational algorithms was performed. the article defines the lemma on the presence of many majorants, the theorem on the complexity of a constant calculation, the theorem on the degree dependence of the complexity of a calculation, the theorem on the addition of a constant to the number of calculations, the theorem on the multiplicity of the complexity of calculations, the theorem on the transitivity of the complexity of calculations, the theorem on the addition of the volumes of calculations, the theorem on addition of calculation complexity, theorem on addition of constant calculation complexity, theorem on multiplication of calculation complexity, theorem on polynomial calculation complexity, theorem on exponential-polynomial calculation complexity, theorem on logarithmic calculation complexity. The considered material clearly demonstrates the practical features of determining majorants according to the changed scheme and shows its practical significance. The given examples of determining the computational complexity of some known algorithms can be used as a basis for determining the computational complexity of other algorithms. In addition, the given system of theorems can be extended and applied to evaluate the complex application of algorithms that complement or are parts of each other.*
Keywords. *Asymptotic evaluation, complexity of the algorithm, majorant, synthesis of algorithms.*

Асимптотичне оцінювання математичних функцій є способом спрощеного опису визначення граничних значень їх поведінки. З практичних міркувань використовуються різноманітні форми асимптотичних наближень. Так для оцінки кількості ітерацій або кроків виконання алгоритмів найчастіше застосовується наближене оцінювання верхньої межі значень відповідних характеристик алгоритмів. Традиційною формою оцінювання верхньої межі поведінки функції є нотація *O*-велике, що є математичною концепцією, яка описує граничну поведінку функції, коли аргумент прагне до певного значення або до нескінченності. Позначення *O*-велике є членом сімейства нотацій, винайдених німецькими математиками Паулем Бахманом [6], Едмундом Ландау [7] та іншими, які разом сьогодні називають нотаціями Бахмана-— Ландау або асимптотичними нотаціями та широко використовуються у визначені складності обчислень задач інформатики та математики.

Актуальним напрямом застосування апарату визначення складності обчислень є задача визначення властивостей алгоритмів,

що реалізуються. Обґрунтоване обрання алгоритмів та їх синтез у вирішені різноманітних задач обумовлює ефективність застосування обраних алгоритмів. Так висвітленню особливостей оцінювання складності алгоритмів в нотаціях Бахмана–Ландау приділено особливу увагу в [1-3], оцінюванню складності обчислення алгоритмів з виділенням класів складності приділено увагу в [1, 4, 5] тощо.

Традиційним підходом до оцінювання обчислювальної складності алгоритму у відповідності до нотації *O*-велике є визначення мажоранти $g(n)$ деякої функції $f(n)$, що описує кількість роботи алгоритму у вигляді $f(n) \le c \cdot g(n)$, де c є деякою константою. Визначення мажорант у зазначений спосіб має спрощений характер, тому часто на практиці носить інтуїтивний характер. У даній публікації пропонується розглянути процес аналітичного визначення мажорант у вигляді $f(n) \le c_1 + c_2 \cdot g(n)$, де c_1 та c_2 є деякими константами. Застосування зазначеної схеми дасть змогу вплинути на збільшення точності та обґрунтування аналітичного вигляду мажорант, що визначаються.

Для опису математичного апарату асимптотичного оцінювання складності обчислювальних алгоритмів пропонується система з однієї леми та 12 теорем. В якості прикладу застосування здійснено асимптотичне оцінювання складності виконання деяких обчислювальних алгоритмів.

Основні результати є наступними:

Теорема 1. *Теорема про складність константного обчислення* Для будь-яких $n \in N$, $n = \text{const}$ є вірним $f(n) = O(1)$.

Доведення. Оскільки, за визначенням оцінки O-велике: $f(n) \le c_1 + c_2 \cdot g(n)$ — то, є відповідності до умов, — $f(n) = const \le c_1 + c_2 \cdot g(n) = c_1 + c_2 \cdot 1$. Звідки отримуємо $g(n) = 1$ або $f(n) = O(1)$.

Теорема 2. *Теорема про ступеневу залежність складності обчислення*

Для будь-яких $r \in R, s \in R, r \le s$ та $n > 1$ є вірним $n^r = O(n^s)$.

Доведення. Відомо, що функція $\ln(x)$ зростає при збільшенні значення свого аргументу. Тоді $a \le b$ тоді і тільки тоді коли $\ln(a) \le \ln(b)$. Якщо $a = n^r$ та $b = n^s$ — то $\ln(n^r) \le \ln(n^s)$ або $r \cdot \ln(n) \le s \cdot \ln(n)$. Оскільки за умови що $n > 1$ значення $\ln(n)$ має додатне значення — то розділивши ліву та праву частину останньої

нерівності на $\ln(n)$ отримаємо: $r \le s$. Остання отримана нерівність збігається з початковими умовами, що доводить справедливість невірності $n^r \le n^s$. Крім того, оскільки $n^r \le n^s$, очевидним є висновок, що n^s є мажорантою для n^r. Звідки отримуємо $n^r = O(n^s)$.

Теорема 3. *Теорема про додавання константи до кількості обчислень*

Для будь-яких $k \in R$ є вірним $f(n) = O(g(n))$, $k + f(n) = O(g(n))$.

Доведення. За визначенням оцінки O-велике — $g(n)$ мажорує $f(n)$ якщо $f(n) \le c_1 + c_2 \cdot g(n)$. Або, у відповідності до початковим умов, — $k + f(n) \le c_1 + c_2 \cdot g(n)$, звідки: $f(n) \le c_1 - k + c_2 \cdot g(n)$. Оскільки значення $c_1 - k$ та c_2 в останньому виразі є константами — то, за визначенням оцінки O-велике, $g(n)$ є мажорантою для $f(n)$. Звідки отримуємо $f(n) = O(g(n))$ та $k + f(n) = O(g(n))$.

Теорема 4. *Теорема про кратність складності обчислень*

Для будь-яких $k \in R$ є вірним $f(n) = O(g(n))$, $k \cdot f(n) = O(g(n))$.

Доведення. За визначенням оцінки *O*-велике — $g(n)$ мажорує $f(n)$ якщо $f(n) \le c_1 + c_2 \cdot g(n)$. Або, у відповідності до початковим умов, — $k \cdot f(n) \le c_1 + c_2 \cdot g(n)$, звідки: $f(n) \le c_1 / k + c_2 / k \cdot g(n)$. Оскільки значення c_1 / k та c_2 / k в останньому виразі є константами — то, за визначенням оцінки O-велике, $g(n)$ є мажорантою для $f(n)$. Звідки отримуємо $f(n) = O(g(n))$ та $k \cdot f(n) = O(g(n))$.

Лема 1. *Лема про наявність багатьох мажорант*

Якщо $f(n) = O(g(n))$ та $g(n) = O(h(n))$ — то як $g(n)$ так і $h(n)$ є мажорантами для $f(n)$.

Доведення. За визначенням оцінки O-велике $f(n) \le c_1 + c_2 \cdot g(n)$ та $g(n) \le c_3 + c_4 \cdot h(n)$. Або $f(n) \le c_1 + c_2 \cdot g(n) = c_1 + c_2 \cdot (c_3 + c_4 \cdot h(n)) = \underbrace{c_1 + c_2 \cdot c_3}_{const} + \underbrace{c_2 \cdot c_4}_{const} \cdot h(n)$. Звідки $h(n)$ є мажорантою для $f(n)$ або $f(n) = O\big(h(n)\big)$. Оскільки з умов леми маємо $f(n) = O(g(n))$ та з доведеного твер-

дження отримано $f(n) = O\big(h(n)\big)$ — то $g(n)$ та $h(n)$ є мажорантами для $f(n)$.

Теорема 5. *Теорема про транзитивність складності обчислень*

Для $f(n) = O(g(n))$ та $g(n) = O(h(n))$ є вірним $f(n) = O(h(n))$.

Доведення. З умов теореми та з леми (1) випливає, що $h(n)$ є мажорантою для $f(n)$, що доводить $f(n) = O(h(n))$.

Теорема 6. *Теорема про додавання обсягів обчислень*

Для $f(n) = O(g(n))$ та $h(n) = O(g(n))$ є вірним $f(n) + h(n) = O(g(n))$.

Доведення. За визначенням оцінки O-велике $f(n) \le c_1 + c_2 \cdot g(n)$ та $h(n) \le c_3 + c_4 \cdot g(n)$. Тоді, склавши окремо ліві та окремо праві частини зазначених нерівностей, отримаємо $f(n) + h(n) \le \underbrace{c_1 + c_3}_{const} + \underbrace{(c_2 + c_4)}_{const} \cdot g(n)$. Звідки отримуємо $f(n) + h(n) = O(g(n))$.

Теорема 7. *Теорема про додавання складності обчислень*

Для $f(n) = O(g(n))$, $h(n) = O(e(n))$, $g(n) = O(e(n))$ є вірним $f(n) + h(n) = O(e(n))$.

Доведення. Оскільки з умови теореми $e(n)$ є мажорантою для $g(n)$ — то, за теоремою 5, отримуємо $f(n) = O(e(n))$. Звідки, за теоремою 6, отримуємо $f(n) + h(n) = O(e(n))$.

Теорема 8. *Теорема про додавання константної складності обчислень*

Для $f(n) = O(g(n))$ та $h(n) = O(1)$ є вірним $f(n) + h(n) = O(g(n))$.

Доведення. За визначенням оцінки O-велике $f(n) \le c_1 + c_2 \cdot g(n)$ та $h(n) \le c_3 + c_4 \cdot 1$. Тоді, склавши окремо ліві та окремо праві частини зазначених нерівностей, отримаємо $f(n) + h(n) \le \underbrace{c_1 + c_3 + c_4 \cdot 1}_{const} + \underbrace{c_2}_{const} \cdot g(n)$. Звідки отримуємо $f(n) + h(n) = O(g(n))$.

Теорема 9. *Теорема про множення складності обчислень*

Для $f(n) = O(g(n))$ та $h(n) = O(e(n))$ є вірним $f(n) \cdot h(n) = O(g(n) \cdot e(n))$.

Доведення. За визначенням оцінки O-велике $f(n) \le c_1 + c_2 \cdot g(n)$ та $h(n) \le c_3 + c_4 \cdot e(n)$. Тоді, помноживши між собою окремо ліві та окремо праві частини зазначених нерівностей, отримаємо $f(n) \cdot h(n) \le (c_1 + c_2 \cdot g(n)) \cdot (c_3 + c_4 \cdot e(n))$ або $f(n) \times h(n) \le c_1 \cdot c_3 + c_1 \cdot c_4 \cdot e(n) + c_2 \cdot c_3 \cdot g(n) + c_2 \cdot c_4 \cdot g(n) \cdot e(n)$. За вказаних умов теореми та за означенням оцінки O-велике можна стверджувати, що знайдуться такі $k_1 \ge 1$, $k_2 \ge 1$, $k_3 \ge 1$ та $k_4 \ge 1$ для яких $g(n) \le k_1 + k_2 \cdot g(n) \cdot e(n)$ та для яких $e(n) \le k_3 + k_4 \cdot g(n) \cdot e(n)$. Звідки вираз $g(n) \cdot e(n)$ є мажорантою, як для $g(n)$ так і для $e(n)$. Тоді за теоремою 1 маємо $c_1 \cdot c_3 = O(1)$ та за теоремою 4 маємо $c_1 \cdot c_4 \cdot e(n) = O(g(n) \cdot e(n))$, $c_2 \cdot c_3 \cdot g(n) = O(g(n) \cdot e(n))$ та, відповідно, $c_2 \cdot c_4 \cdot g(n) \cdot e(n) = O(g(n) \cdot e(n))$.

Або $f(n) \cdot h(n) = O(1) + 3 \cdot O(g(n) \cdot e(n))$. Звідки за теоремами 4 та 8 отримуємо $f(n) \cdot h(n) = O(g(n) \cdot e(n))$.

Теорема 10. *Теорема про поліноміальну складність обчислень*

Для $f(n) = \sum_{i=0}^{k} a_i n^i$ є вірним $f(n) = O(n^k)$.

Доведення. $f(n) = a_0 + a_1 \cdot n + a_2 \cdot n^2 + ... + a_{k-1} n^{k-1} + a_k \cdot n^k$. За визначенням оцінки *O*-велике можна стверджувати, що знайдуться такі $c_1 \ge 1$ та $c_2 \ge 1$ для яких $n^k \le c_1 + c_2 \cdot n^k$. Отже вираз n^k э також мажорантою для n^k. Разом з цим, з огляду на те, що $1 \le 2 \le ... \le k-1 \le k$ — то, за теоремою 2, n^k э мажорантою для n, мажорантою для n^2, ..., мажорантою для n^{k-1}. Тоді складність обчислення $f(n)$ можемо записати як $f(n) = O(1) + k \cdot O(n^k)$. Звідки за теоремами 4 та 8 отримуємо $f(n) = O(n^k)$.

Теорема 11. *Теорема про експоненційно-поліноміальну складність обчислень*

Для будь-яких $n \in N$ є вірним $f(n) = n! = O(n^n)$.

Доведення. За визначенням оцінки O-велике можна стверджувати, що знайдуться такі $c_1 \geq 1$ та $c_2 \geq 1$ для яких $n! \leq c_1 + c_2 \cdot n^n$. Крім того у виразах $n!$ та n^n однакова кількість множників значення яких у $n!$ спадають у порівнянні з n^n, що свідчить про те, що $n! \leq n^n$ для будь-яких $n \geq 0$. Отже n^n є мажорантою для $n!$. Звідки отримуємо $f(n) = n! = O(n^n)$.

Теорема 12. *Теорема про логарифмічну складність обчислень*

Для будь-яких a та b є вірним $f(n) = \log_a n = O(\log_b n)$.

Доведення. За визначенням оцінки O-велике та з властивостей логарифмічної функції

$$f(n) = \log_a n = \log_b n / \log_a b \leq \underbrace{c_1}_{const} + \underbrace{c_2 \cdot (1/\log_a b)}_{const} \cdot \log_b n .$$

Звідки $\log_b n$ є мажорантою для $\log_a n$ або $f(n) = \log_a n = O(\log_b n)$.

Приклади доведення обчислювальної складності деяких алгоритмів.

Приклад 1. Розглянемо алгоритм пошуку мінімального та алгоритм пошуку максимального значення серед елементів числової сукупності без попередньої підготовки останньої. Код таких алгоритмів може бути описаний так, як це показано на рис. 1.

```
1  def searchMin(a):
2      l = len(a)
3      if l > 0:
4          v = a[0]
5          for i in range(1, l):
6              if v > a[i]:
7                  v = a[i];
8          return v;
9      return False
10
11 def searchMax(a):
12     l = len(a)
13     if l > 0:
14         v = a[0]
15         for i in range(1, l):
16             if v < a[i]:
17                 v = a[i];
18         return v;
19     return False
```

Рис. 1. Код алгоритмів пошуку мінімального та максимального значень серед елементів числової непідготовленої сукупності

Оскільки сукупність числових елементів не підготовлена — то для пошуку потрібного числового елементу може бути необхідним здійснення від 1 до n звернень до елементів сукупності. Тому через це та у відповідності до теореми 5 можемо стверджувати, що n може бути мажорантою кількості звернень до неупорядкованої сукупності елементів. Справді, у відповідності до правила визначення оцінки O-велике $\exists c_1 \in R$, $\exists c_2 \in R$, $f(n) \le c_1 + c_2 \cdot h(n)$ та якщо $h(n) = n$ — то завжди знайдуться такі $c_1 \in R$ та $c_2 \in R$, що $n \le c_1 + c_2 \cdot h(n)$. Звідки $f(n) = O(n)$.

Приклад 2. Розглянемо алгоритм сортування розташування елементів числової сукупності методом бульбашки. Код такого алгоритму може бути описаний так, як це показано на рис. 2.

```
1  def bubbleSort(a):
2      b = a.copy()
3      l = len(b)
4      if l > 1:
5          for i in range(0, l):
6              for j in range(0, l - i - 1):
7                  if b[j] > b[j + 1]:
8                      t = b[j + 1]
9                      b[j + 1] = b[j]
10                     b[j] = t
11     return b
```

Рис. 2. Код алгоритму сортування елементів числової сукупності методом бульбашки

Даний алгоритм є ітераційним і потребує $\sum_{i=1}^{n} n = \frac{n^2 + n}{2}$ звернень до елементів числової сукупності. У відповідності до правила визначення оцінки O-велике $\exists c_1 \in R$, $\exists c_2 \in R$, $f(n) \le c_1 + c_2 \cdot h(n)$. Оскільки вже відомо, що $f(n) = \frac{n^2 + n}{2}$ – то завжди знайдуться такі $c_1 \in R$ та $c_2 \in R$, що $f(n) = \frac{n^2 + n}{2} \le c_1 + c_2 \cdot \frac{n^2 + n}{2}$ звідки $f(n) = O\left(\frac{n^2 + n}{2}\right)$. Але з властивостей визначення правила оцінки O-велике $\exists c_1 \in R$,

$\exists c_2 \in R,\ f(n) \le \underbrace{c_1}_{const} + \underbrace{\frac{c_2}{2}}_{const} \cdot (n^2 + n)$. Тому $f(n) = O(n^2 + n)$. Звідки, оскільки n^2 є мажорантою для n, то з теорем 5 та 7 отримаємо $f(n) = O(n^2)$.

Приклад 3. Розглянемо алгоритм бінарного пошуку потрібного числового елементу серед заздалегідь упорядкованих за значенням елементів. Код такого алгоритму може бути описаний так, як це показано на рис. 3.

```
1  def binarySearch(a, v):
2      l = 0
3      r = len(a) - 1
4      while l <= r:
5          i = l + (r - l) // 2;
6          if a[i] == v:
7              return i;
8          if a[i] > v:
9              r = i - 1
10         else:
11             l = i + 1
12     return False
```

Рис. 3. Код алгоритму бінарного пошуку
числового елементу в упорядкованій сукупності

Алгоритм ітераційний, але оскільки сукупність елементів упорядкована за їх значеннями — то перша ітерація потребує доступу до всього початкового набору елементів, друга та наступні ітерації — потребують доступу до половини обсягу елементів у порівнянні з попередньою ітерацією і так доти, поки коло пошуку не звузиться до одного елементу.

Таким чином маємо залежності $\forall n \in Z,\ n \ge 0,\ \forall s \in N,\ 2^s \ge n$, де n — кількість елементів числової сукупності, s — мінімальна кількість кроків алгоритму, яка достатня для задоволення вказаних залежностей. Звідки $s \cdot \log(2) \ge \log(n)$ або $s \ge \frac{\log(n)}{\log(2)} = \log_2(n)$. Так як з умов алгоритму $s \in N$ та значення s повинно бути мінімальним серед можливих значень — то $f(n) = s \le \lfloor \log_2(n) \rfloor + 1$. Звідки з останньої нерівності, з правила визначення оцінки *O*-велике та з теореми 12 отримуємо $f(n) = O(\log(n))$.

Приклад 4. Розглянемо алгоритм отримання значень довільних членів ряду чисел Фібоначі. Код такого алгоритму може бути описаний так, як це показано на рис. 4.

```
1 def getFib(n: int):
2     return n if n < 2 else getFib(n - 1) + getFib(n - 2)
```

Рис. 4. Код алгоритму отримання значення *n*-го члену ряду чисел Фібоначі

Алгоритм має рекурсивну форму тому обчислення виконуються рекурентно. Визначення асимптотичної складності рекурентних обчислень зручно описати у наступній формі:

$$f(n) = \begin{cases} O(...), & s = S; \\ \overset{c}{\uparrow} f(...)_s, & 1 \le s < S, \end{cases}$$

де S — кількість кроків виконання розрахунків; s — номер окремого кроку; $O(...)$ — асимптотична оцінка обчислювальної складності за умови, що рекурентні звернення не застосовуються; c — константа кратність виконання рекурентних звернень; $\overset{c}{\uparrow} f(...)_s$ — довільні рекурентні звернення у кількості c на s-му кроці виконання розрахунків.

Наведена схема наочно демонструє, що зі зростанням кількості кроків виконання розрахунків кількість рекурентних звернень буде збільшуватись кратно c на кожному новому кроці крім останнього для якого таке збільшення вже не потребується. Отже $f(n) = O(c^s) + c \cdot O(...)$. Звідки, за теоремою 4 отримаємо $f(...) = O(c^s) + O(...)$.

Особливістю алгоритму отримання значення n-го члену ряду чисел Фібоначі, що наведений на рис. 4, є кратність виконання рекурентних звернень $c = 2$ при $n \ge 2$, складність обчислень дорівнює $O(1)$ при $n < 2$. З останнього отримуємо $f(n) = O(2^n) + O(1)$ звідки за теоремою 8 маємо $f(n) = O(2^n)$.

Розглянутий матеріал наочно демонструє практичні особливості визначення мажорант за схемою $f(n) \le c_1 + c_2 \cdot g(n)$. Наведені приклади визначення обчислювальної складності деяких відомих алгоритмів можуть бути використані в якості ілюстративного ма-

теріалу під час викладання матеріалу відповідного направлення та основи визначення обчислювальної складності у досліджені інших алгоритмів. Крім того, наведена система теорем може бути розширена і бути застосована для оцінювання комплексного застосування алгоритмів, які взаємодоповнюють або є частинами один одного тощо.

Бібліографічні посилання

1. Бородкіна І. Л., Бородкін Г. О. Теорія алгоритмів. — Київ: Центр учбової літератури, 2021. — 184 с.
2. Матвієнко М. П. Теорія алгоритмів. — Київ: Видавництво Ліра-К, 2022. — 340 с.
3. Провотар О.І. Конкретна алгоритміка. Київ. Наукова думка. 2017. 168 с.
4. Троцько В. В. Теорія алгоритмів: Навчально. — Київ: Університет економіки та права «КРОК», 2023. — 123 с.
5. Шкільняк С. С. Математична логіка, основи теорії алгоритмів. — Київ: ДП «Вид. дім «Персонал», 2009. — 280 с.
6. Bachmann P. Die analytische zahlentheorie. — Leipzig: B. G. Teubner, 1894. — 494 p.
7. Landau E. Handbuch der Lehre von der Verteilung der Primzahlen. — Leipzig: B. G. Teubner, 1909. — 564 p.

Статтю подано до редакції: 21.11.2023