Харківський національний університет імені В. Н. Каразіна Факультет комп'ютерних наук
Кафедра штучного інтелекту та програмного забезпечення

ЗВІТ
Практична робота №1
дисципліна: «Теорія алгоритмів»

Виконав: студент групи КС-22
Ковальов Андрій
Перевірив: викладач
Олешко Олег

Харків

Завдання.

Реалізувати алгоритми Евкліда (найбільшого загального дільника двох цілих чисел - GCD) та Ератосфена (знаходження всіх простих чисел до деякого цілого числа n).

Для кожного з алгоритмів:

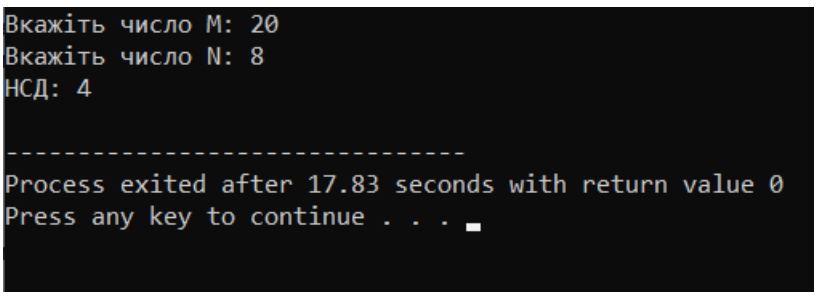
1. Розробити блок-схему алгоритму
2. Привести текстовий опис (по кроках) алгоритму.
3. Виконати програмну реалізацію алгоритму у вигляді функції мовою С, навести приклади роботи алгоритму.

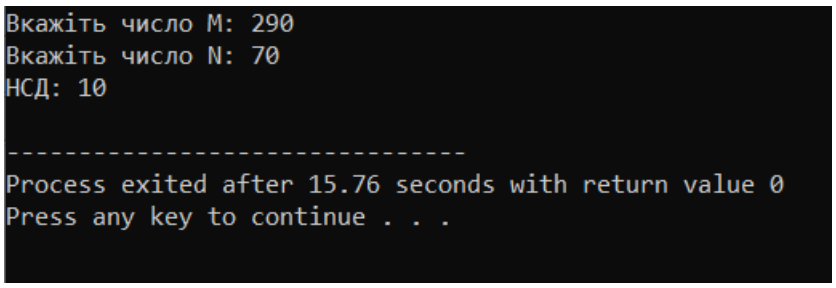
Додаткові бали: знайдіть, опишіть та реалізуйте інші варіанти "решета", окрім вищезгаданого решета Ератосфена.

Алогиртм Евкліда:

```
#include <stdio.h>

int main() {
        int M, N;
        printf("Вкажіть число M: ");
        scanf("%d", &M);
        printf("Вкажіть число N: ");
        scanf("%d", &N);

   while (M != N) {
      if (M > N) M -= N;
      else N -= M;
   }

   printf("НСД: %d\n", M);
   return 0;
}
```

Лістинг 1 – код програми





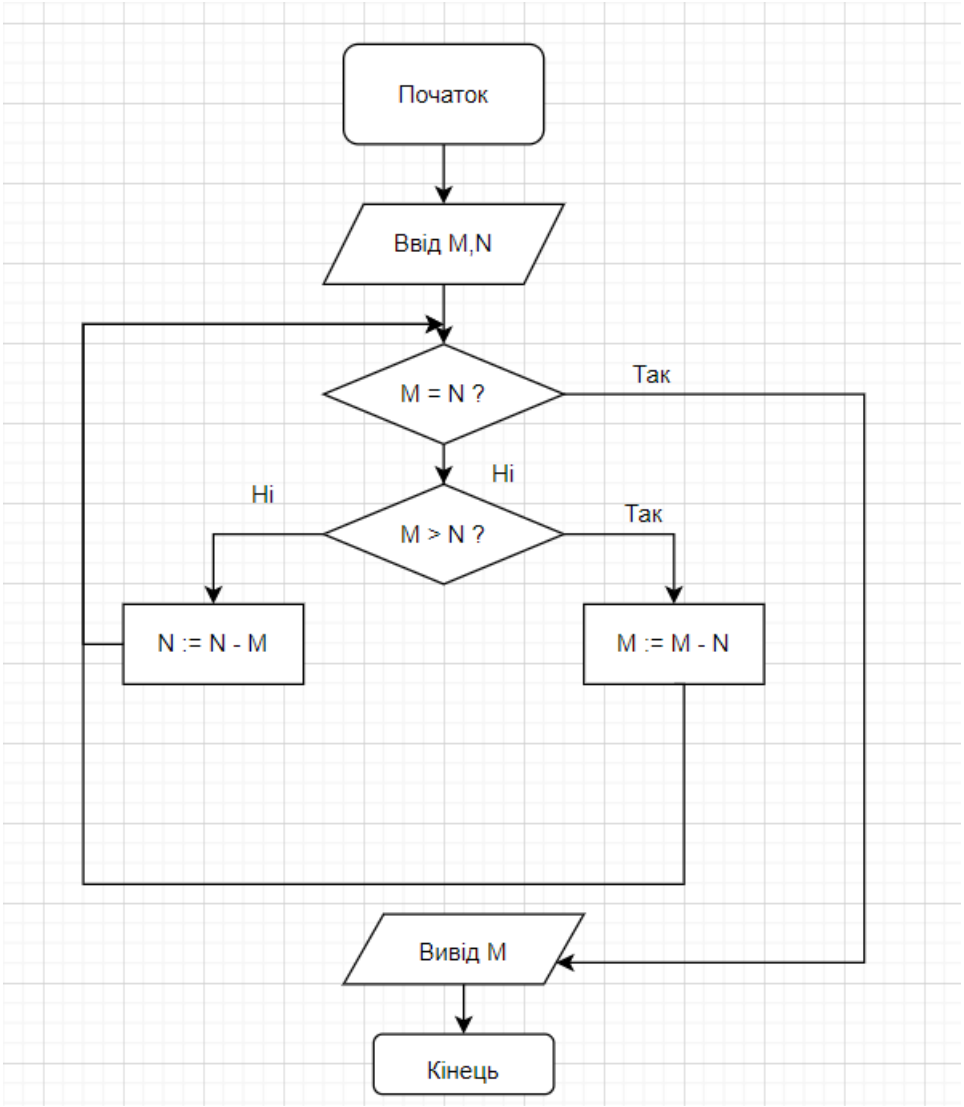
Скріншот 1-2 – виконання програми

Текстовий опис алгоритму

1. Введіть два числа M і N.
2. Після того, як користувач ввів числа, програма переходить до циклу while, який продовжуватиме виконуватись доти, доки значення M і N не стануть рівними.

- Якщо значення M більше, ніж N, з M віднімається N.
- Якщо ж N більше за M, то з N віднімається M.

3. Цикл завершується, коли M і N стають рівними. Це значення є найбільшим спільним дільником (НСД) двох чисел. Після завершення циклу програма виводить результат — НСД чисел M і N — за допомогою printf().



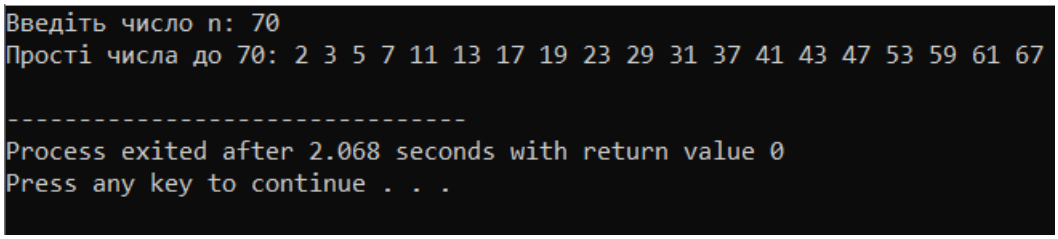
Блоксхема алгоритму

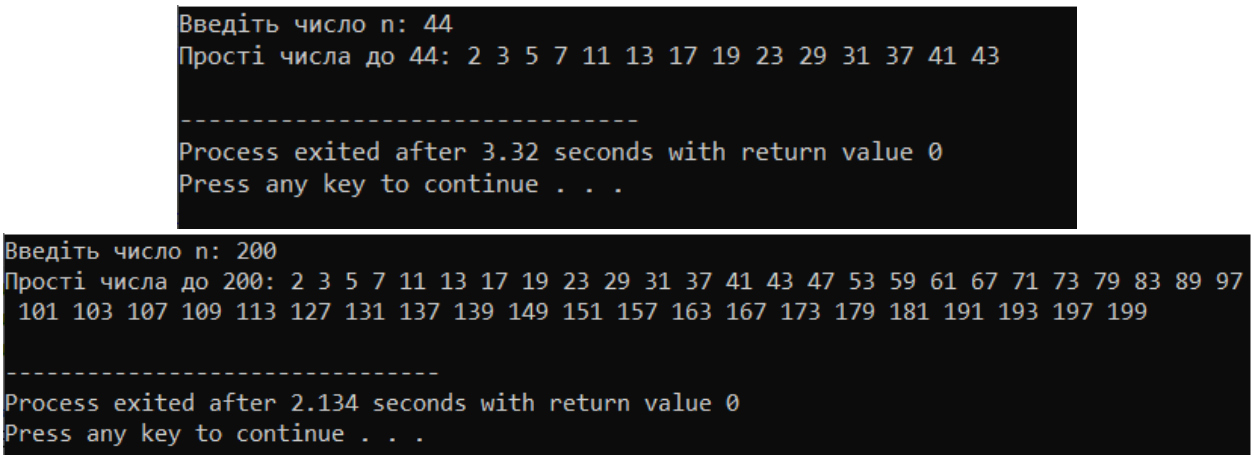
Алгоритм Ератосфена:

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

// Функція для реалізації сітки Ератосфена
void Eratosthenes(int n) {
    bool isPrime[n+1];
    for (int i = 0; i <= n; i++) {
        isPrime[i] = true;
```

```
    }
    isPrime[0] = isPrime[1] = false;

    for (int i = 2; i <= sqrt(n); i++) {
        if (isPrime[i]) {
            for (int j = i * i; j <= n; j += i) {
                isPrime[j] = false;
            }
        }
    }

    printf("Прості числа до %d:\n", n);
    for (int i = 2; i <= n; i++) {
        if (isPrime[i]) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

int main() {
    int n;
    printf("Введіть число n: ");
    scanf("%d", &n);
    Eratosthenes(n);

    return 0;
}
```

Лістинг 2 – код програми





Скріншот 3-5 – результат виконання програми

Текстовий опис алгоритму

1. Введіть число n.
2. Створється булевий масив, розміром n + 1, де кожен елемент спочатку відповідає числу від 0 до n та всі числа вважаються простими.
3. Алгоритм починає з числа 2 (перше просте число). Для кожного наступного числа i , якщо воно відмічене як просте(true), а всі його кратні позначаються як непрості(false).
4. Після завершення основного циклу алгоритм виводить всі числа, які залишились відміченими як прості.

Додаткове завдання. Алгоритм Сундарама:

Опис алгоритму Сундарама:

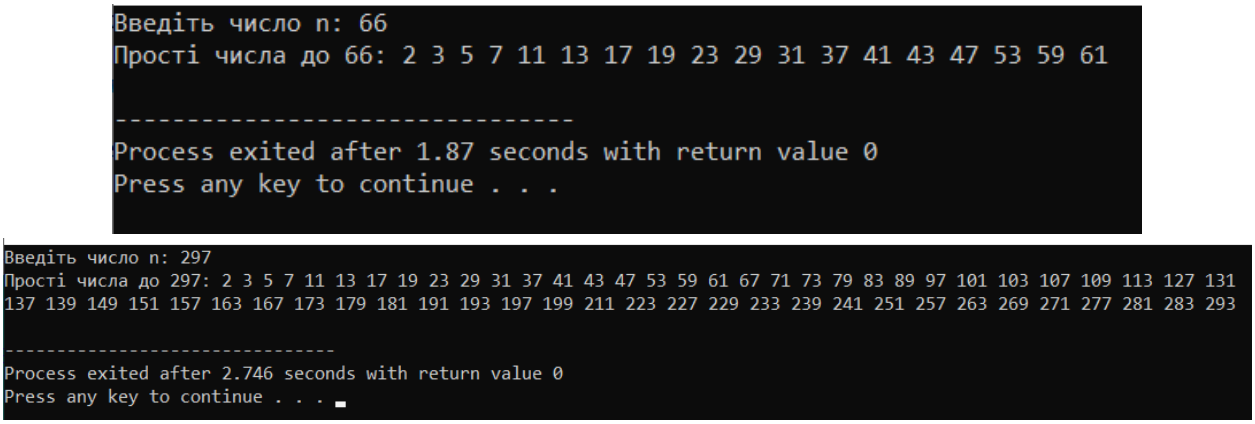
Алгоритм Сундарама — це ще один метод для знаходження простих чисел, схожий на алгоритм Ератосфена, але трохи простіший. Він працює для пошуку всіх простих чисел менших за певне число n, але замість прямої роботи з усіма числами, працює з меншими індексами, які потім перетворюються у прості числа.

```
#include <stdio.h>
#include <stdbool.h>

void Sundaram(int n) {
```

```
    int limit = (n - 1) / 2;
    bool marked[limit + 1];

    // Ініціалізуємо всі числа як прості
    for (int i = 0; i <= limit; i++) {
        marked[i] = false;
    }

    // Викреслюємо числа за допомогою формули i + j + 2 * i * j
    for (int i = 1; i <= limit; i++) {
        for (int j = i; (i + j + 2 * i * j) <= limit; j++) {
            marked[i + j + 2 * i * j] = true;
        }
    }

    // Виводимо прості числа
    printf("Прості числа до %d: 2 ", n);
    for (int i = 1; i <= limit; i++) {
        if (!marked[i]) {
            printf("%d ", 2 * i + 1);
        }
    }
    printf("\n");
}

int main() {
    int n;
    printf("Введіть число n: ");
    scanf("%d", &n);
    Sundaram(n);
    return 0;
}
```

Лістинг 3 – код програми

```
Введіть число n: 95
Прості числа до 95: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89

--------------------------------
Process exited after 4.492 seconds with return value 0
Press any key to continue . . .
```



Скріншот 6-8 – результат виконання програми

Алгоритм по кроках:

1. Користувач вводить значення n, яке є верхньою межею для пошуку простих чисел.
2. Створюється масив булевих значень marked[] довжиною limit+1, де всі елементи ініціалізуються як false. Цей масив використовується для позначення чисел, які не є простими.
3. Два вкладених цикли використовуються для пошуку індексів, які виключаються. Для кожної пари i та j, виконується формула: index=i+j+2·i·j.Якщо індекс не перевищує межу, він позначається як true в масиві, що означає, що це число не є простим.
4. Вивести всі прості числа до n.

Алгоритм Аткіна:

Опис алгоритму:

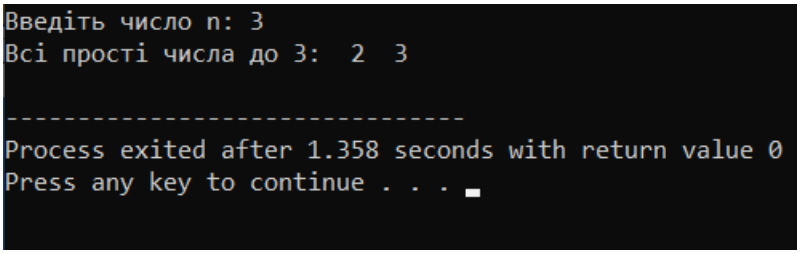
Алгоритм Аткіна — це ефективний метод знаходження всіх простих чисел до певного числа nnn. Він є покращенням класичного решета Ератосфена з точки зору швидкості виконання, використовуючи математичні властивості для виключення великих груп чисел, які точно не є простими.

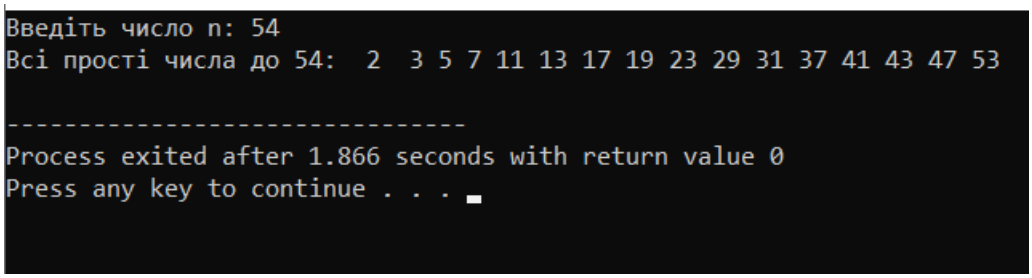
```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>

void Atkin(int n) {
    if (n < 2) {
        printf("Простих чисел немає.\n");
        return;
```

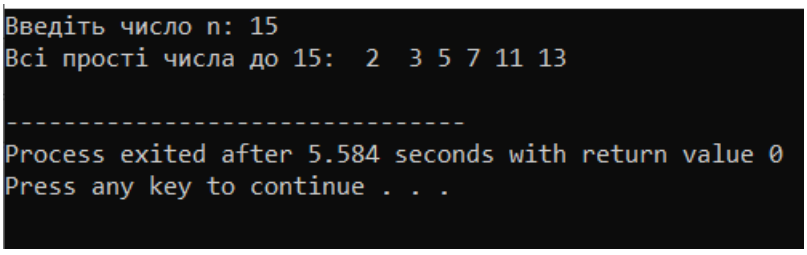
```
}
        printf("Всі прості числа до %d: ", n);
        if (n >= 2) printf(" 2 ");
if (n >= 3) printf(" 3 ");

bool sieve[n + 1];
for (int i = 0; i <= n; i++) {
   sieve[i] = false;
}

// Основні формули для знаходження простих чисел
for (int x = 1; x * x <= n; x++) {
   for (int y = 1; y * y <= n; y++) {
      int num = 4 * x * x + y * y;
      if (num <= n && (num % 12 == 1 || num % 12 == 5)) {
         sieve[num] ^= true;
      }

      num = 3 * x * x + y * y;
      if (num <= n && num % 12 == 7) {
         sieve[num] ^= true;
      }

      num = 3 * x * x - y * y;
      if (x > y && num <= n && num % 12 == 11) {
         sieve[num] ^= true;
      }
   }
}

// Викреслюємо всі кратні квадратів
for (int i = 5; i * i <= n; i++) {
   if (sieve[i]) {
      for (int j = i * i; j <= n; j += i * i) {
         sieve[j] = false;
      }
   }
}

// Виводимо всі прості числа
```

```
    for (int i = 5; i <= n; i++) {
        if (sieve[i]) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

int main() {
    int n;
    printf("Введіть число n: ");
    scanf("%d", &n);
    Atkin(n);
    return 0;
}
```

Лістинг 4 – код програми







Скріншот 9-11 – результат виконання програми