Завдання до комп’ютерного практикуму № 3
з дисципліни «Алгоритми та структури даних»
Тема: «Багатопотокові алгоритми»

Мета роботи

Отримати навички роботи з багатопотоковими алгоритмами.

Основні теоретичні відомості

1. Паралелізм і багатопотоковість

Паралелізм (найпростіше визначення) – це одночасне виконання двох або більше операцій.

Паралелізм в обчислювальній системі – коли система виконує кілька незалежних операцій паралельно, а не послідовно.

Історично склалося, що комп’ютери оснащувались одним процесором з одним блоком обробки (ядром). Така машина може виконувати тільки одну задачу в кожний момент часу, однак, може переключатись між задачами багато разів за секунду – це називається *перемикачем задач*.

Комп’ютери з кількома процесорами застосовуються для організації серверів та виконання високопродуктивних обчислень вже багато років. Сучасні персональні комп’ютери мають кілька ядер на одному кристалі (багатоядерні процесори). Неважливо, машина має кілька процесорів або один з кількома ядрами, вона може виконувати більше однієї задачі в кожний момент часу. Це називається *апаратним паралелізмом*.

Паралелізм за рахунок кількох процесів

Кожний процес працює незалежно від усіх інших, і всі процеси можуть виконувати різні послідовності команд.

Переваги:

+ операційна система (ОС) забезпечує захист;

+ процеси можна запускати на різних машинах, об’єднаннях мережею.

Недоліки:

– складна комунікація. ОС має забезпечити захист процесів, щоб ні один не зміг випадково змінити дані іншого;
– витрати на запуск кількох процесів (для запуску необхідний час – ОС повинна виділити ресурси для керування процесом).

Паралелізм за рахунок кількох потоків

Потік можна вважати полегшеним процесом – кожний потік працює незалежно від усіх інших, і всі потоки можуть виконувати різні послідовності команд.

Переваги:

+ єдиний адресний простір (прямий доступ до глобальних змінних; вказівники, посилання на об’єкти можна передавати між потоками);
+ витрати ОС на запуск малі, через відсутність контролю доступу до даних;

Недоліки:

– комунікацію налаштовує програміст: якщо до деякого елемента даних звертаються кілька потоків, програміст має забезпечити узгодженість представлення цього елемента у всіх потоках.

Дві основні причини використання паралелізму:

1) розподіл обов’язків;
2) продуктивність.

Розподіл обов’язків

Якщо згрупувати пов’язані та розділити незв’язні частини коду, то програма стане простішою для розуміння та тестування і, як результат, буде містити менше помилок.

Продуктивність

Виробники збільшують кількість ядер, а не нарощують продуктивність одного ядра.

Є два способи підвищити продуктивність, реалізувавши:

1. *Розпаралелювання за задачами*. Розбити задачу на частини і запустити їх паралельно, зменшивши загальний час виконання.
2. *Розпаралелювання за даними*. Кожний потік виконує одну і ту саму операцію, над різними даними.

Паралелізм використовувати недоцільно:

– коли витрати перевищують виграш (паралельний алгоритм важчий для сприйняття – більше помилок, а отже дорожчий у супроводі);
– якщо потенційний приріст продуктивності недостатньо великий;
– потоки – обмежений ресурс, якщо їх занадто багато ОС уповільнює роботу.

2. Клас для роботи з багатопотоковістю std::thread стандарту 2011 року мови програмування C++

Клас **std::thread** являє собою єдиний потік виконання. Потоки дозволяють виконувати декілька функцій одночасно.

Потоки починають виконання одразу після побудови пов'язаного об'єкта потоку, починаючи з функції, наданої як аргумент конструктора.

Лістинг 1. Проста багатопотокова програма

```
1 #include <stdio.h>
2 #include <thread>
3 void do_sth(){
4   printf("Hello IASA!\n");
5 }
6 int main(){
7   std::thread t(do_sth);//побудова об'єкта потока,
    //який має виконати ф-ю do_sth()
8   t.join(); // чекає завершення виконання потоку
9   return 0;
10  }
```

Присутня нова директива **#include <thread>**. Усе необхідне для підртимки багатопотоковосі оголошене в файлі **<thread>**.

Код виведення повідомлення переміщений в окрему функцію **do_sth()**, рядок 3, лістинга 1. У кожного потоку має бути ***початкова функція***, в якій починається виконання потоку. Для першого потоку програми це функція **main()**, а для всіх інших початкова функція передається у конструктор об'єкта **std::thread.** Тобто рядок 7, лістинга 1, запускає новий потік і загальна кількість потоків дорівнює двом (головний потік функції **main()** і додатковий, що починає роботу в функції **do_sth()**).

У рядку 8 головний потік чекає завершення додаткового потоку, що асоційований з обєктом **std::thread**, у даному випадку, змінною **t**, якщо цього не зробити потік функції **main()** може завершити роботу програми, можливо, ще до початку виконання функції **do_sth()**.

Метод **join()** об'єкта потоку чекає, поки потік завершить виконання.

Якщо не потрібно очікувати завершення потоку, необхідно гарантувати, що дані, до яких потік звертається, залишаються дійсними поки вони йому потрібні.

Метод **detach()** дозволяє потоку працювати незалежно від головного потоку.

Передача аргументів функції потоку

Передача аргументів функції потоку зводиться до передачі додаткових аргументів конструктору **std::thread**

Лістинг 2. Передача аргументів функції потоку

```
1 #include <stdio.h>
2 #include <thread>
3 void do_sth(const char * msg){
4   printf(msg);
5 }
6 int main(){
7   std::thread t(do_sth, "Hello IASA!\n");
    //побудова об'єкта потока, який має виконати
    //функцію do_sth("Hello IASA!\n")
8   t.join(); // чекає завершення виконання потоку
9   return 0;
10  }
```

3. Приклад реалізації багатопотокового алгоритму

Завдання

Реалізувати емуляцію роботи програвача. Програвач випадково генерує числа з діапазону від 0 до N, користувач може зупиняти програвач на паузу і завершувати роботу програвача.

Словесний опис алгоритму

Програвач має два принципових набори обов'язків:

1. Генерувати дані.
2. Реагувати на дії користувача: «Пауза», «Вихід».

Доцільно розділити реалізацію на два потоки, які будуть виконують тільки свої обов'язки.

Для керування потоками використаємо цілочислову змінну **`state`**, в якій будемо зберігати стан програвача: 0 – пауза, 1 – програвання, 2 – вихід.

Блок-схеми алгоритмів

На рис. 1 блок-схема алгоритму випадкового генерування даних, реалізованого у вигляді функції **`random`**.

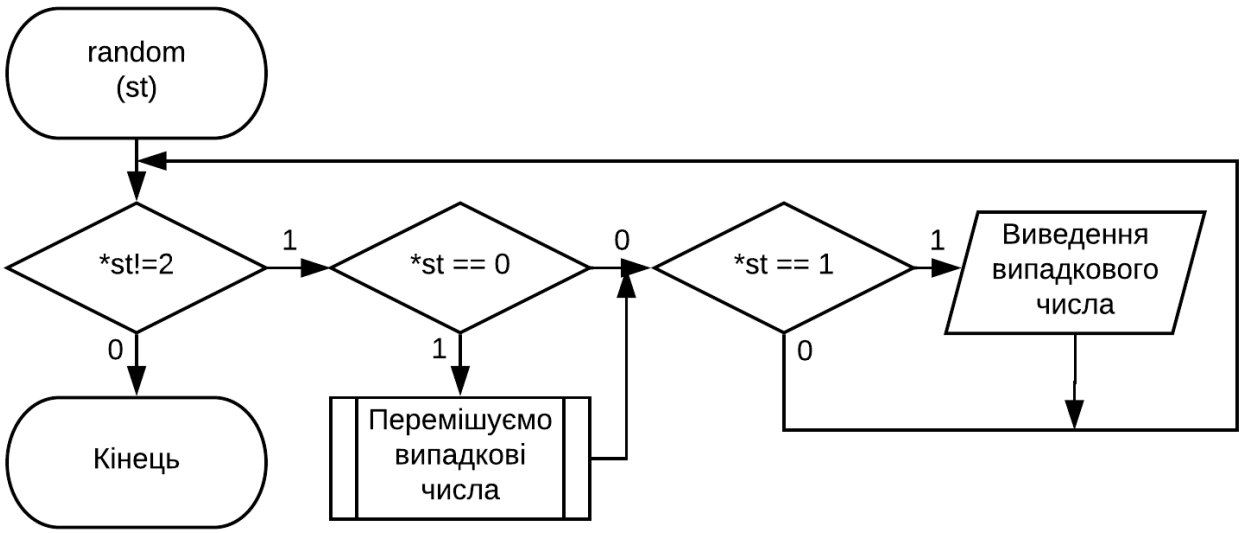


Рис. 1. Алгоритм генерування даних

Функція **`random`**:

– друкує псевдовипадкові числа, якщо стан, на який посилається **`st`**, дорівнює «1» (програвання);
– припиняє друк і перемішує псевдовипадкову послідовність, якщо стан, на який посилається **`st`**, є «0» (пауза);
– завершує роботу, якщо стан, на який посилається **`st`**, дорівнює «2» (вихід).

Алгоритм, що реагує на дії користувача, реалізований у вигляді функції **`pause`**, рис.2. Функція реагує на дії користувача, а саме:

– натискання клавіші *«Enter»* змінює стан програвача, на який посилається **`st`**, на протилежний: ставить на паузу або відновлює роботу програвача;
– натискання клавіші *«q»* (і підтвердження вибору натисканням *«Enter»*) змінює стан програвача, на який посилається **`st`**, на значення «2»: завершує роботу програвача.

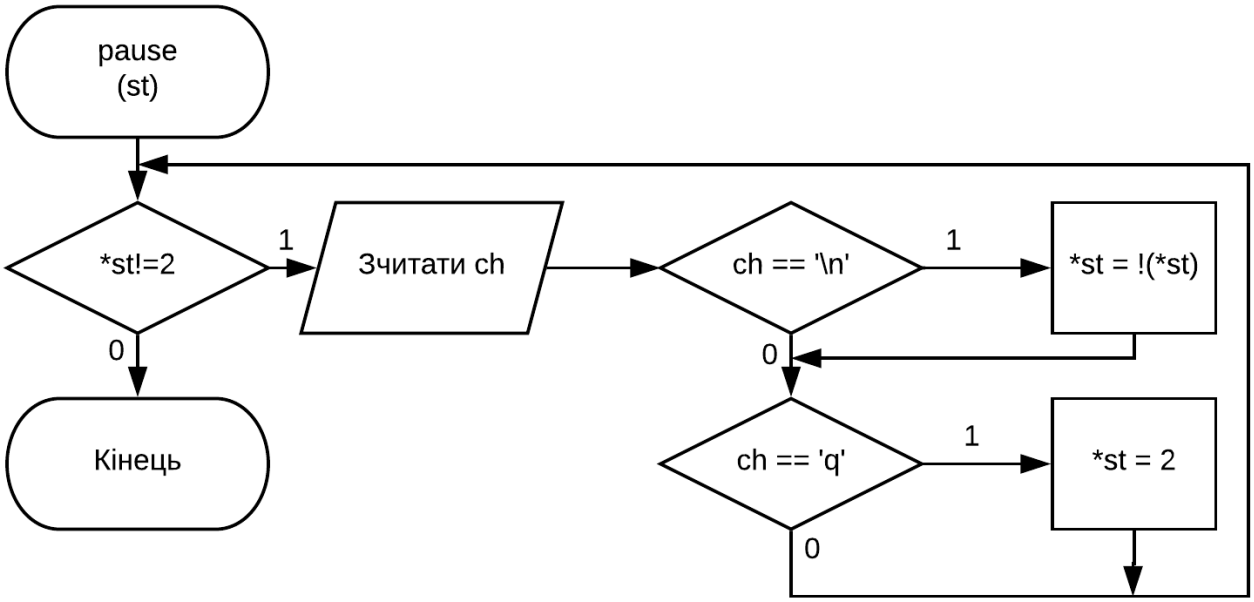


Рис. 2. Алгоритм реагування на дії користувача

Алгоритм запуску функцій у головній функції програми, рис. 3.

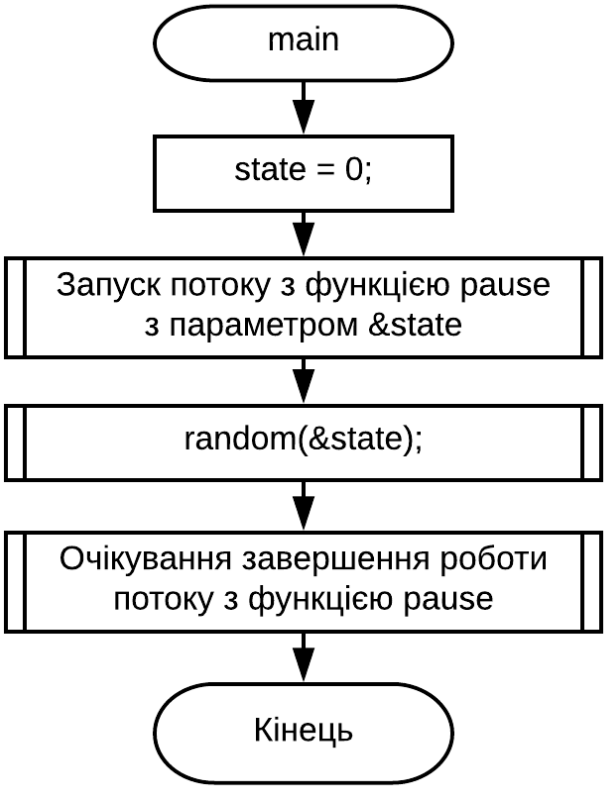


Рис. 3. Алгоритм головної програми

Лістинг 3. Програмна реалізація розроблених алгоритмів

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <thread>

void pause(int* st){
    char ch;
    while(*st!=2){
        ch = getchar();
        if (ch == '\n') *st = !(*st);
        if (ch=='q') {
            *st = 2;
        }
    }
}

void random(int* st){
    while(*st!=2){
        if (*st==0) srand(time(NULL));
        if (*st==1) printf("%d ", rand()%10);
    }
}
```

Лістинг 3 продовження

```
int main(){
    int state = 0;

    std::thread t(pause, &state);
    random(&state);
    t.join();

    return 0;
}
```

Результати роботи

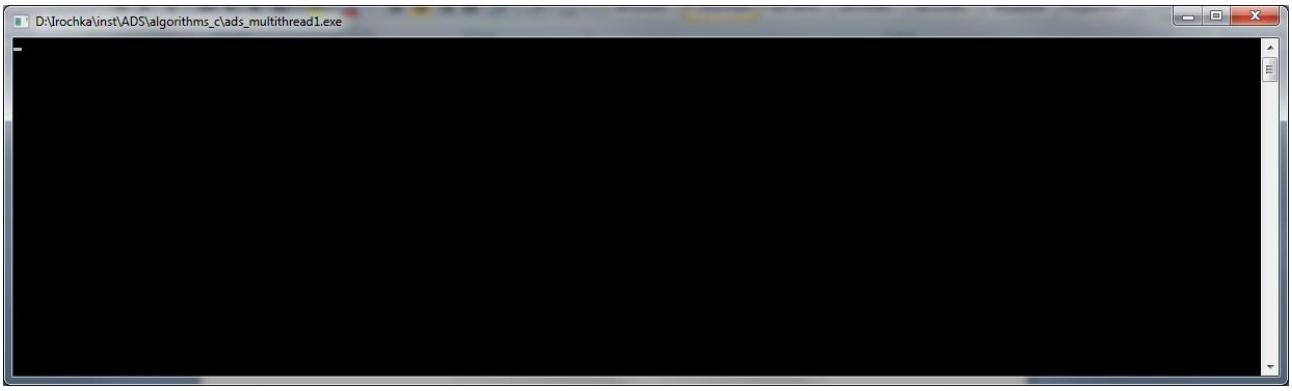


Рис. 4. Початкове вікно програми

Рис. 5. Генерування даних

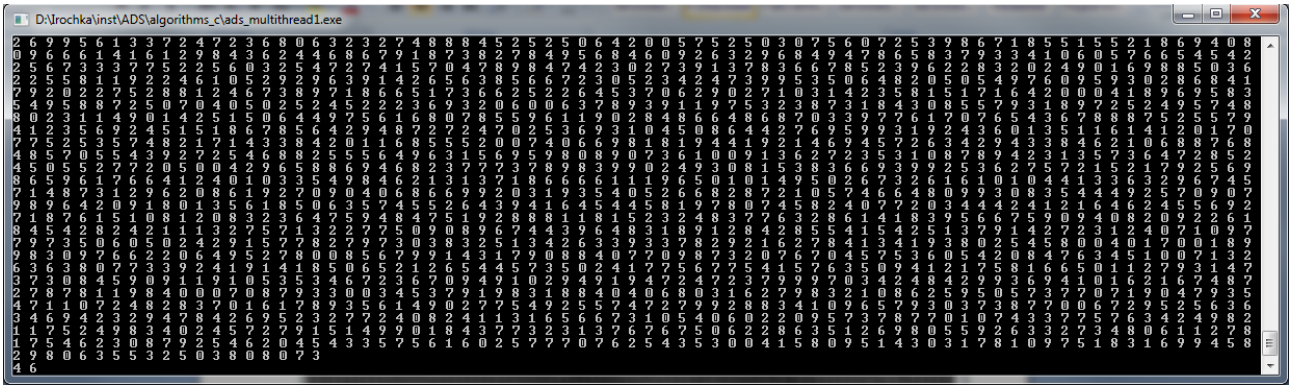


Рис. 6 Вікно програми після натискання клавіші «Enter»

Рис. 7. Вікно програми після натискання клавіші «q» та підтвердження, натисканням клавіші «Enter»

Висновки

У даному випадку кількість потоків не залежить від кількості ядер, програма розділена на потоки для розподілу обов'язків, а не для збільшення продуктивності.

Перевага: проста реалізація.

Недолік: змішані структурне та об'єкт-орієнтовне програмування. Програма написана у «стилі С» з використанням екземпляра об'єкта класу **`std::thread`** мови програмування С++ стандарта 2011 року.

Лістинг 4. Програмна реалізація розроблених алгоритмів на чистому C

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void pause(int* st){
    char ch;
    while(*st!=2){
        ch = getchar();
        if (ch == '\n') *st = !(*st);
        if (ch == 'q')   *st = 2;
    }
}

void random(int* st){

    while(*st!=2){
        if (*st==0) srand(time(NULL));
        if (*st==1) printf("%d ", rand()%10);
    }
}

int main(){
    int state = 0;

    pthread_t t;
    pthread_create(&t, NULL, pause, &state);
    random(&state);
    pthread_join(t,NULL);

    return 0;
}
```

Строк виконання комп’ютерного практикуму № 3

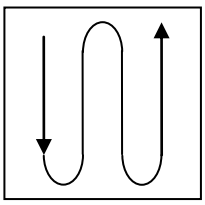
15 травня 2019 року

4. Порядок виконання роботи

1. Розробити послідовний алгоритм, відповідно варіанту завдання.
2. Реалізувати багатопотокову версію розробленого алгоритму, використавши, розпаралелювання за задачами або/і за даними.
3. Створити програми, відповідно розробленим алгоритмам.
4. Порівняти час виконання програм для послідовного та багатопотокового алгоритмів.
5. Результати оформити в звіт:
    1. Титульний лист.
    2. Варіант завдання.
    3. Завдання.
    4. Реалізований алгоритм у вигляді БС.
    5. Лістинг програми.
    6. Результати роботи програм.
    7. Висновки.

5. Варіанти завдань

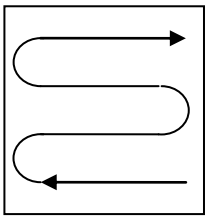
1. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



2. Обчислити $\sum_{i=1}^{100} \sum_{j=1}^{50} \frac{1}{i + j^2}$, вивести окремо внутрішні суми.

3. Задано матрицю, відсортувати кожний із рядків за зростанням методом обміну.

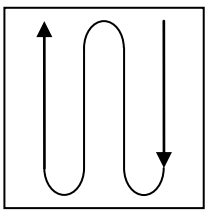
4. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



5. Обчислити $\sum_{i=1}^{100} \sum_{j=1}^{50} \sin\left(i^3 + j^4\right)$, вивести окремо внутрішні суми.

6. Задано матрицю, відсортувати кожний із стовпчиків за зростанням шейкерним методом.

7. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:

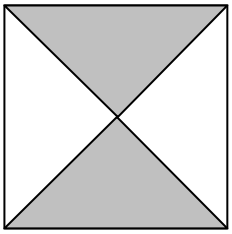


8. Задано матрицю, відсортувати кожний із стовпчиків за зростанням методом вставки.

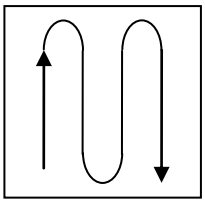
9. Обчислити $\sum_{i=1}^{100} \sum_{j=i}^{100} \frac{j - i + 1}{i + j}$, вивести окремо внутрішні суми.

10. Задано матрицю, відсортувати кожний із рядків за зростанням методом вставки.

11. Є дійсна квадратна матриця порядку n. Знайти мінімальний елемент, розташований у заштрихованій області:



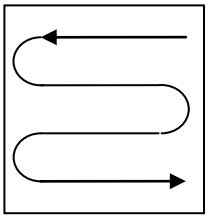
12. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



13. Для наданих x, k обчислити $\sum_{k=1}^{n} \sum_{m=k}^{n} \frac{x+k}{m}$, вивести окремо внутрішні суми.

14. Надані два натуральних числа, m, n. Знайти всі числа, менші за n, сума цифр яких дорівнює m.

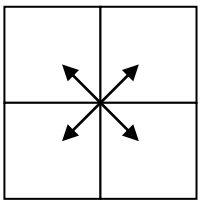
15. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



16. Обчислити $\sum_{i=1}^{100} \sum_{j=1}^{i} \frac{1}{i+2j}$, вивести окремо внутрішні суми.

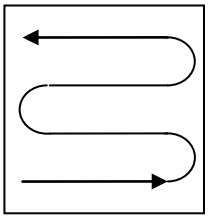
17. Задано матрицю, відсортувати кожний із рядків за спаданням шейкерним методом.

18. З дійсної квадратної матриці порядку 2n. отримати нову матрицю такою перестановкою її блоків розміром $n \times n$:



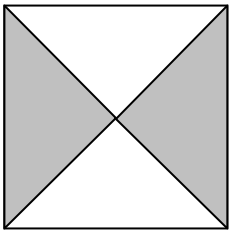
19. Задано матрицю, відсортувати кожний із стовпчиків за зростанням методом вибору.

20. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



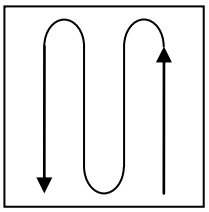
21. Обчислити суму ряду $s = \sum_{i=1}^{n} (-1)^{i+1} x_i$, значення x_i зберігаються у масиві.

22. Є дійсна квадратна матриця порядку n. Знайти максимальний елемент, розташований у заштрихованій області:



23. Задано матрицю, відсортувати кожний із рядків за зростанням методом обміну з прапорцем.

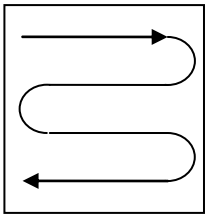
24. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



25. Задано матрицю, відсортувати кожний із стовпчиків за спаданням методом обміну.

26. Задано матрицю, елементами якої є натуральні числа, визначити кількість елементів, які є квадратами цілих чисел.

27. Утворити матрицю $n \times n$, елементами якої є натуральні числа, розташовані наступним чином:



28. Задано матрицю, відсортувати кожний із стовпчиків за спаданням методом обміну з прапорцем.

29. Задано матрицю, відсортувати кожний із рядків за зростанням методом вибору.

30. Дано дійсні числа x, ε $(x \neq 0, 0 < \varepsilon < 1)$. Обчислити з точністю до ε значення суми: $\sum_{k=1}^{\infty} (-1)^k \frac{k \sin kx}{k}$.

Варіанти завдань на вибір

31. Реалізувати сортування одновимірного масиву за зростанням методом швидкого сортування.
32. Задано матрицю, відсортувати кожний із рядків за зростанням методом швидкого сортування.
33. Задано матрицю, відсортувати кожний із стовпців за спаданням методом швидкого сортування.
34. Реалізувати сортування одновимірного масиву за спаданням пірамідальним методом.
35. Задано матрицю, відсортувати кожний із стовпців за зростанням пірамідальним методом.
36. Задано матрицю, відсортувати кожний із рядків за спаданням пірамідальним методом.
37. Реалізувати сортування одновимірного масиву за зростанням методом сортування злиттям.
38. Задано матрицю, відсортувати кожний із стовпців за спаданням методом сортування злиттям.
39. Задано матрицю, відсортувати кожний із рядків за зростанням методом сортування злиттям.

6. Контрольні запитання

1. Що таке паралелізм?
2. Що таке перемикач задач?
3. Що таке апаратний паралелізм?
4. Які організації паралелізму Ви знаєте?
5. Переваги та недоліки різних організацій паралелізму.
6. Навіщо потрібний паралелізм?
7. Що таке розпаралелювання за задачами?
8. Що таке розпаралелювання за даними?
9. Коли паралелізм використовувати недоцільно?