

## Лабораторное занятие 1.

Разработать программу для сравнения эффективности двух заданных алгоритмов сортировки путем их одновременного запуска на случайном массиве из 50000 целых чисел. Обеспечить вывод отсортированной последовательности в файл. Программа должна отображать ход вычислений, допускать приостановку и прерывания вычислений. Потоки не синхронизировать.

В 11 варианте необходимо реализовать следующие типы сортировок: Пирамидальная, Выбором.

Код:

```
using System;
using System.IO;
using System.Threading;

class Program
{
    private static readonly int arraySize = 50000;

    class SortContext
    {
        public volatile bool PauseRequested;
        public volatile bool CancelRequested;
        public int Progress;
    }

    static void Main(string[] args)
    {
        int[] originalArray = GenerateRandomArray(arraySize);
        int[] heapSortArray = (int[])originalArray.Clone();
        int[] selectionSortArray = (int[])originalArray.Clone();

        var heapContext = new SortContext();
        var selectionContext = new SortContext();

        Thread heapThread = new Thread(() => HeapSort(heapSortArray, heapContext));
        Thread selectionThread = new Thread(() => SelectionSort(selectionSortArray, selectionContext));

        Console.WriteLine("Нажмите ПРОБЕЛ для паузы/возобновления, ESC для отмены.\n");
        Console.WriteLine("Пирамидальная сортировка: 0%");
        Console.WriteLine("Сортировка выбором: 0%");

        heapThread.Start();
        selectionThread.Start();

        bool running = true;
        while (running)
        {
            try
```

```

{
    if (Console.KeyAvailable) // Теперь в try-catch
    {
        var key = Console.ReadKey(true).Key;
        if (key == ConsoleKey.Spacebar)
        {
            heapContext.PauseRequested = !heapContext.PauseRequested;
            selectionContext.PauseRequested = !selectionContext.PauseRequested;
        }
        else if (key == ConsoleKey.Escape)
        {
            heapContext.CancelRequested = true;
            selectionContext.CancelRequested = true;
            running = false;
        }
    }
}

catch (InvalidOperationException)
{
    // Игнорируем ошибку, если консоль недоступна для ввода
    running = false;
}

Console.SetCursorPosition(0, 2);
Console.WriteLine($"Пирамидальная сортировка: {heapContext.Progress}% ");
Console.WriteLine($"Сортировка выбором: {selectionContext.Progress}% ");

Thread.Sleep(50);
}

heapThread.Join();
selectionThread.Join();

if (heapContext.Progress == 100)
    SaveArrayToFile(heapSortArray, "heap_sorted.txt");
if (selectionContext.Progress == 100)
    SaveArrayToFile(selectionSortArray, "selection_sorted.txt");

Console.WriteLine("\nЗавершено. Результаты сохранены в файлы.");
}

static int[] GenerateRandomArray(int size)
{
    Random rnd = new Random();
    int[] arr = new int[size];
    for (int i = 0; i < size; i++)
        arr[i] = rnd.Next();
    return arr;
}

static void SaveArrayToFile(int[] arr, string filename)

```

```

{
    using (StreamWriter sw = new StreamWriter(filename))
    foreach (int num in arr)
    sw.WriteLine(num);
}

static void HeapSort(int[] arr, SortContext context)
{
    int n = arr.Length;

    for (int i = n / 2 - 1; i >= 0; i--)
    {
        CheckStatus(context);
        Heapify(arr, n, i);
        context.Progress = (int)((n / 2 - i) / (double)(n / 2) * 50);
    }

    for (int i = n - 1; i > 0; i--)
    {
        CheckStatus(context);
        Swap(arr, 0, i);
        Heapify(arr, i, 0);
        context.Progress = 50 + (int)((n - i) / (double)n * 50);
    }

    context.Progress = 100;
}

static void Heapify(int[] arr, int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest]) largest = l;
    if (r < n && arr[r] > arr[largest]) largest = r;

    if (largest != i)
    {
        Swap(arr, i, largest);
        Heapify(arr, n, largest);
    }
}

static void SelectionSort(int[] arr, SortContext context)
{
    int n = arr.Length;

    for (int i = 0; i < n - 1; i++)
    {
        CheckStatus(context);

```

```
int minIdx = i;
for (int j = i + 1; j < n; j++)
    if (arr[j] < arr[minIdx])
        minIdx = j;

Swap(arr, i, minIdx);
context.Progress = (int)((i + 1) / (double)(n - 1) * 100);
}
```

```
context.Progress = 100;
}
```

```
static void CheckStatus(SortContext context)
{
    if (context.CancelRequested)
        throw new OperationCanceledException();

    while (context.PauseRequested)
    {
        Thread.Sleep(100);
        if (context.CancelRequested)
            throw new OperationCanceledException();
    }
}
```

```
static void Swap(int[] arr, int i, int j)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
}
```