# Информационные технологии и программирование

## Лекция 7. Основы языка C#

**Содержание лекции:**

- Алгоритмы сортировки
- Алгоритмы поиска
- Большой проект «Анализ текста»

Преподаватель курса: Клюкин Даниил Анатольевич, ст. преподаватель каф. ПМиИТ

# Алгоритмы сортировки

**Сортировка** – процесс упорядочивания элементов по некоторому признаку.

Упорядочивание для людей:

- Список контактов в телефоне (по алфавиту).
- Цены в интернет-магазине (от дешевых к дорогим).
- Результаты поиска в Google (по релевантности).

Упорядочивание для машин:

- Многие быстрые алгоритмы работают только с отсортированными данными.

**Суть алгоритма «Пузырек» (Bubble Sort)**

**Основная идея:**

Мы идем по массиву слева направо и сравниваем **соседние** элементы парами.

**Сравниваем:** Если левый элемент больше правого — мы меняем их местами.

**Двигаемся дальше:** Переходим к следующей паре.

**Итог одного прохода:** За один полный обход массива самый большой элемент «всплывает» в самый конец (как пузырек воздуха в воде).

**Повторение:** Мы повторяем такие проходы до тех пор, пока массив не будет отсортирован.

Визуализация процесса:

Массив: [5, 1, 4, 2]

Сравниваем 5 и 1 Меняем: [1, 5, 4, 2]

Сравниваем 5 и 4 Меняем: [1, 4, 5, 2]

Сравниваем 5 и 2 Меняем: [1, 4, 2, 5]

Самый большой элемент (5) «всплыл» наверх, как пузырек воздуха в воде.

# Сортировка пузырьком

```
void BubbleSort(double[] arr)
{
   for (int i = 0; i < arr.Length; i++)
   {
      for (int j = 0; j < arr.Length - i - 1; j++)
      {
         if (arr[j] > arr[j+1])
         {
            var temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }
}
```

6 5 3 1 8 7 2 4

Профессионалы почти никогда не пишут «пузырек» вручную. В стандартных библиотеках C# используются сложные алгоритмы (например, **QuickSort** или **IntroSort**). Их код занимает сотни строк, и изучать их подробно на 1 курсе рано, но важно знать их мощь.

```
// Для массивов
int[] numbers = { 5, 1, 4, 2 };
Array.Sort(numbers);

// Для списков
List<string> names = new List<string>{ "Tom", "Alice", "Bob" };
names.Sort();
```

Системная сортировка (QuickSort): O(nlogn)

**Известные сортировки:**

Случайная

Пузырьком

Шейкерная

Вставками

По частям

Блинная

Шелла

Слиянием

Выбором

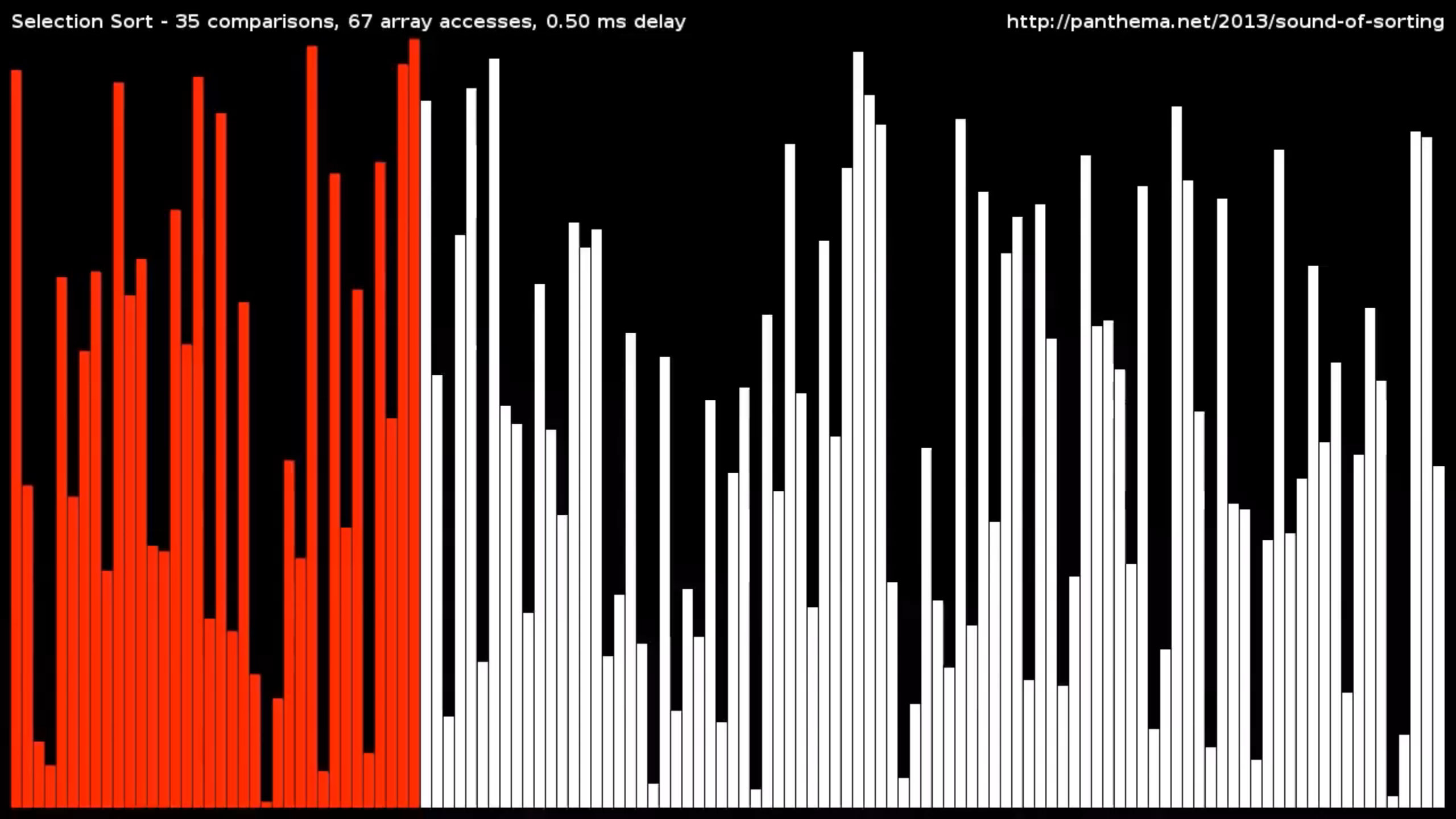
Быстрая

Гномья

Бинарным деревом

Расческой

Подсчетом


Selection Sort - 35 comparisons, 67 array accesses, 0.50 ms delay
http://panthema.net/2013/sound-of-sorting

Полезные ссылки:

1. Быстрая сортировка (https://code-maze.com/csharp-quicksort-algorithm/), сложность алгоритма O(nlog(n))
2. Шейкерная сортировка (https://programm.top/c-sharp/algorithm/array-sort/shaker-sort/), сложность $O(n^2)$
3. Сортировка вставками (https://programm.top/c-sharp/algorithm/array-sort/insertion-sort/), сложность $O(n^2)$
4. Сортировка расческой (https://programm.top/c-sharp/algorithm/array-sort/comb-sort/), сложность O(nlog(n))

# Алгоритмы поиска

**Формулировка** проблемы:

Есть список элементов, нужно найти индекс элемента (сам элемент), который удовлетворяет условию.

Самый простой и интуитивный способ найти что-то — проверить всё подряд.

```
int FindLinear(int[] array, int target)
{
    for (int i = 0; i < array.Length; i++)
    {
        if (array[i] == target)
            return i; // Нашли! Возвращаем индекс
    }
    return -1; // Прошли всё и не нашли
}
```

**Бинарный поиск.** Этот алгоритм в разы быстрее линейного, но он требует соблюдения строгого условия.

**Важное условие:** Массив **ОБЯЗАТЕЛЬНО** должен быть отсортирован.

**Принцип «Загадай число»:**

Представьте, что я загадал число от 1 до 100.

При **линейном поиске** вы бы спрашивали: «Это 1? Это 2? Это 3?..» (До 100 попыток).

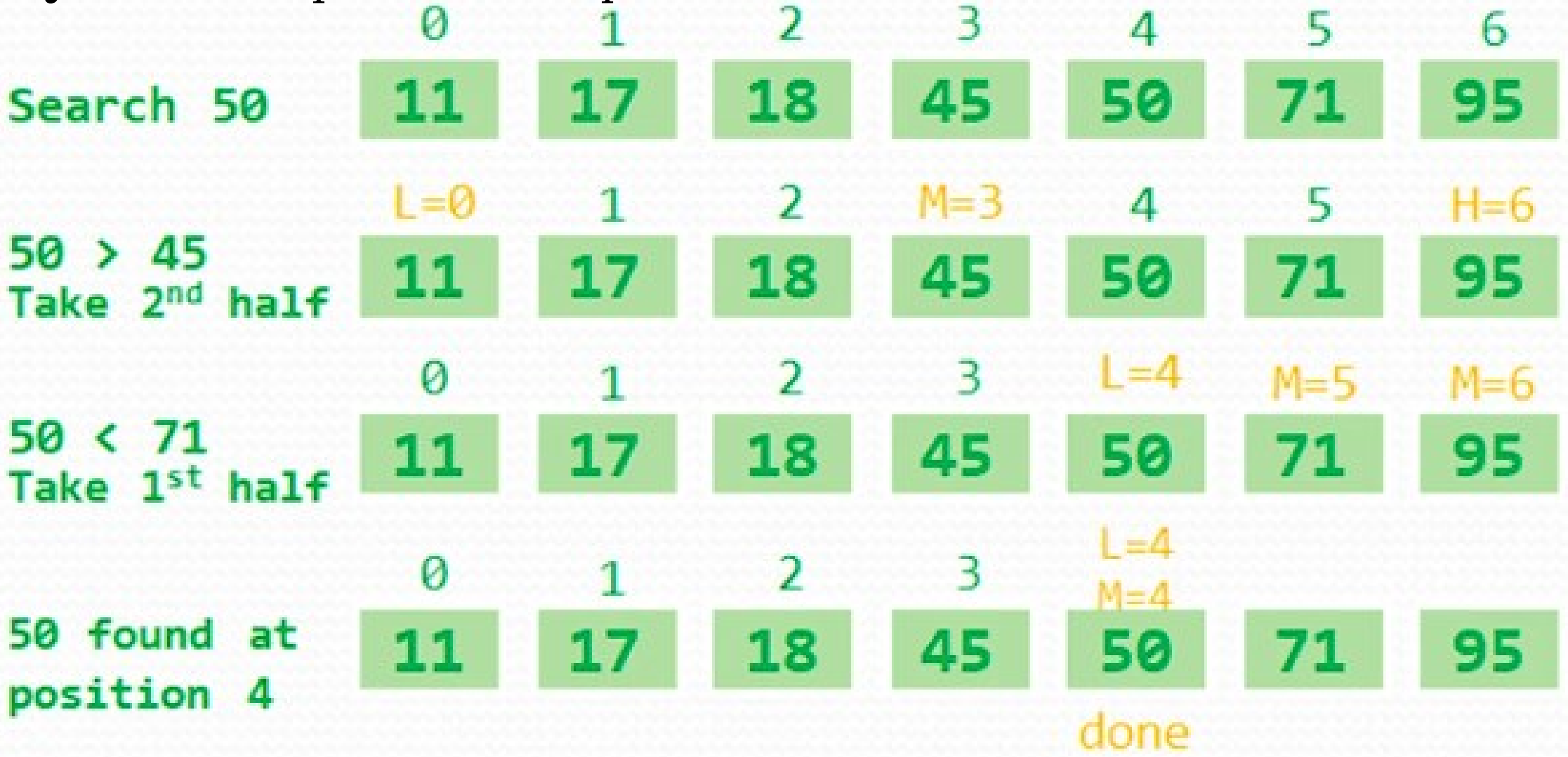
При **бинарном поиске** вы спросите: «Это больше 50?».

Если я скажу «Больше», вы мгновенно отсекаете половину (все числа от 1 до 50) и больше их не проверяете.

Затем вы спрашиваете про середину оставшегося отрезка: «Это больше 75?» и так далее.

Бинарный поиск невероятно быстрый. Чтобы найти число среди **1 000 000** элементов, бинарному поиску нужно всего около **20 шагов**.

**Визуализация** работы бинарного поиска.



```
int BinarySearch(int[] array, int target)
{
    int left = 0;
    int right = array.Length - 1;

    while (left <= right)
    {
        // Находим середину текущего отрезка
        int mid = left + (right - left) / 2;

        if (array[mid] == target)
            return mid; // Нашли!

        if (array[mid] < target)
            left = mid + 1; // Искомое в правой половине
        else
            right = mid - 1; // Искомое в левой половине
    }
    return -1; // Элемент не найден
}
```

# Большой проект «Анализ текста»

**Задача:** У нас есть переменная string text, в которой хранится вся книга (например, «Гарри Поттер»).

Нам нужно превратить этот хаос букв в полезную статистику.

```
string text = "500 страниц текста о Гарри Поттере";
```

**Что мы будем использовать (Весь арсенал 1 семестра):**

**Списки (List)** — чтобы хранить предложения и слова.

**Словари (Dictionary)** — чтобы считать частоту слов.

**Множества (HashSet)** — чтобы мгновенно отсеивать «мусорные» слова (и, в, на).

**StringBuilder** — чтобы эффективно чистить текст от знаков препинания.

**Методы** — чтобы разбить сложную задачу на понятные кирпичики.

**Сложность алгоритмов** — чтобы наша программа не зависла.

**Шаг 0. Подготовка и «чистка» сырого текста**

Прежде чем делить текст на предложения, его нужно превратить в единый поток. Переносы строк (\n) и лишние пробелы сделают наши данные некрасивыми.

**Задача:**

1.Заменить все \n и \r на пробелы.

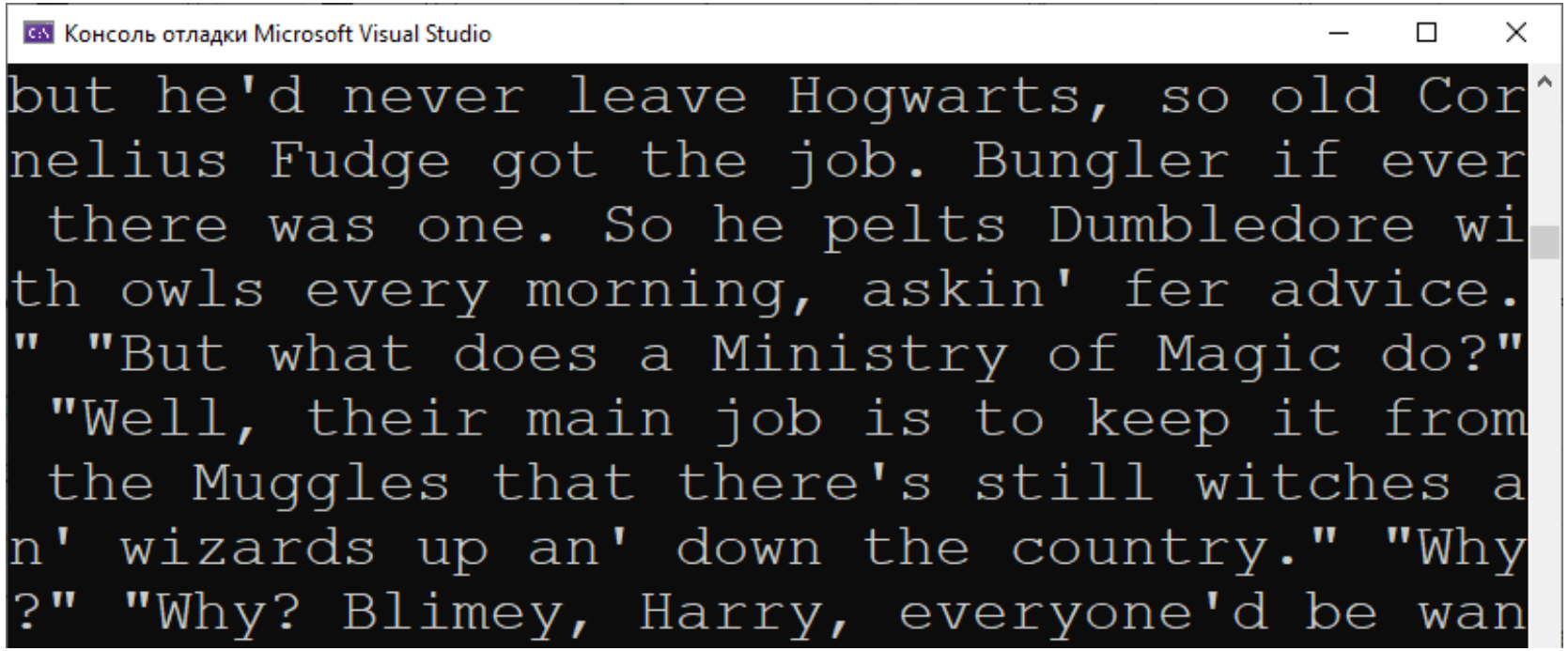
2.Схлопнуть множественные пробелы в один (чтобы не было «пустых» слов).

**Решение (Профессиональный трюк):**

Вместо сложных циклов мы используем комбинацию Split и Join. Это намного быстрее и чище.

```
var clearText = NormalizeText(text);

Console.WriteLine(clearText);
```



**Шаг 1. Разбиваем текст на предложения**

```
List<string> GetSentences(string text)
{
    // Указываем символы завершения предложения
    char[] separators = { '.', '!', '?' };

    // Разбиваем и сразу удаляем пустые строки
    string[] rawSentences = text.Split(separators,
StringSplitOptions.RemoveEmptyEntries);

    // Превращаем массив в удобный список и чистим пробелы по
краям
    List<string> result = new List<string>();
    foreach (var s in rawSentences)
    {
        result.Add(s.Trim());
    }
    return result;
}
```

```
var sentences = GetSentences(text);

Console.WriteLine($"В тексте {sentences.Count} предложений");
```

```
В тексте 6518 предложений
```

**Шаг 2. Очистка от знаков препинания (StringBuilder)**

Внутри предложения слова разделены запятыми, тире, скобками. Нам нужно оставить только буквы.

Почему StringBuilder? Если мы будем менять строку через +=, создание новых строк для книги в 100 000 слов убьет память. StringBuilder сделает это за один проход (O(n)).

```
string CleanText(string sentence)
{
    var sb = new StringBuilder();
    foreach (char c in sentence)
    {
        // Оставляем только буквы и пробелы
        if (char.IsLetter(c) || char.IsWhiteSpace(c) || c == '\'')
        {
            sb.Append(char.ToLower(c)); // Сразу в нижний регистр
        }
    }
    return sb.ToString();
}
```

```
Console.WriteLine(
      $"Очищенное 1-е предложение: {CleanText(sentences[0])}");
```

```
Очищенное 1-е предложение: harry potter and th
e sorcerers stone for jessica who loves storie
s for anne who loved them too and for di who h
eard this one first
```

## Шаг 3. Фильтрация «мусора» (HashSet)

В любом тексте полно слов «и», «в», «не», «он». Они не несут смысла для анализа частоты. Мы назовем их Stop Words.

Почему HashSet? Мы будем проверять каждое слово книги (их 100 000+). В List поиск займет вечность, а в HashSet — мгновенно (O(1)).

```
// Создаем "черный список" слов
HashSet<string> stopWords = new HashSet<string> { "the", "and",
"was", "his", "her", "that", "with", "for", "you", "had" };

List<string> FilterWords(string[] words, HashSet<string>
stopWords)
{
    var filtered = new List<string>();
    foreach (var w in words)
    {
        if (!stopWords.Contains(w) && w.Length > 2)
          // Убираем стоп-слова и короткие союзы
        {
            filtered.Add(w);
        }
    }
    return filtered;
}
```

```
// 1. Чистим от переносов и лишних пробелов
string cleanText = NormalizeText(rawText);

// 2. Бьем на предложения
List<string> sentences = GetSentences(cleanText);

// 3. Собираем все слова для словаря
var allWords = new List<string>();
foreach (var s in sentences)
{
    // Чистим предложение от запятых и бьем на слова
    string onlyLetters = CleanText(s);
    string[] words = onlyLetters.Split(' ');

    // Фильтруем стоп-слова и добавляем в общий список
    allWords.AddRange(FilterWords(words, stopWords));
}
```

```
harry potter and the sorcerers stone for jessica who
loves stories for anne who loved them too and for who
```

## Шаг 4. Частотный словарь (Dictionary) — Сердце программы

Теперь посчитаем, сколько раз встречается каждое слово.

**Логика:** Ключ — это **слово**, значение — это **счетчик**.

```
var wordCounts = new Dictionary<string, int>();

foreach (var word in allWords)
{
    if (wordCounts.ContainsKey(word))
    {
        wordCounts[word]++; // Если слово уже было — увеличим счетчик
    }
    else
    {
        wordCounts[word] = 1; // Если слово впервые — создаем запись
    }
}
```

Теперь посчитаем, сколько раз встречается каждое слово.

**Логика:** Ключ — это **слово**, значение — это **счетчик**.

```
Слово: harry частота: 1195
Слово: potter частота: 90
Слово: sorcerer's частота: 16
Слово: stone частота: 75
Слово: jessica частота: 1
Слово: who частота: 161
Слово: loves частота: 1
Слово: stories частота: 3
Слово: anne частота: 2
Слово: loved частота: 3
```

## Шаг 5. Сортировка и Вывод ТОП-10

```
var statistics = wordCounts.ToArray();

for (int i = 0; i < statistics.Length; i++)
{
    for (int j = i + 1; j < statistics.Length; j++)
    {
        if (statistics[i].Value < statistics[j].Value)
        // Сравниваем частоту
        {
            var temp = statistics[i];
            statistics[i] = statistics[j];
            statistics[j] = temp;
        }
    }
}
```

После сортировки:

```
Слово: harry частота: 1195
Слово: said частота: 794
Слово: they частота: 585
Слово: him частота: 493
Слово: but частота: 456
Слово: ron частота: 408
Слово: all частота: 386
Слово: out частота: 372
Слово: what частота: 335
Слово: hagrid частота: 332
```