Семинар №2

Алгоритмы сортировки

1. **Инструментарий:**

---

2. **Цели семинара №2:**
    - Научиться писать различные алгоритмы сортировки
    - Понимать принципы их работы
    - Уметь оценивать их сложность
    - Понимать принцип работы бинарного поиска

    По итогам семинара №2 слушатель должен **знать**:
    - Различные принципы сортировки массивов
    - Правильно оценивать их сложность и возможность применения

    По итогам семинара №2 слушатель должен **уметь**:
    - Писать различные алгоритмы сортировки
    - Использовать бинарный поиск по массиву

Формат проведения семинара

1. Ученики разбиваются на группы по 5 человек в сессионные комнаты зум
2. Один ученик расшаривает экран
3. Обсуждают задание и как его лучше выполнить
4. Ученик, расшаривший экран пишет код, другие ученики участвуют в обсуждении и подсказывают что нужно еще добавить
5. После выполнения первого задания, ученики скидывают архивы с работой для проверки

---

3. **План Содержание:**

Этап урока	**Тайминг, минуты**	**Формат**

Вопросы по лекции, подготовка к семинару	10	Преподаватель спрашивает, есть ли вопросы по пройденному материалу и отвечает на них
Реализуем простой алгоритм сортировки	10	Реализуем один из простых алгоритмов сортировки, таких как пузырьковая, вставками, выбором
Реализуем быструю сортировку	20	Реализуем алгоритм быстрой сортировки массива
Делаем сравнение времени выполнения различных алгоритмов сортировки на большом объеме данных	15	Преподаватель предлагает измерить скорость выполнения алгоритмов и сделать выводы, какой из них будет оптимальнее
Реализуем алгоритм бинарного поиска	15	Реализуем алгоритм бинарного поиска по массиву
Длительность:	**70**	

4. Блок 1.

Задание:
Необходимо написать один из простых алгоритмов сортировки, имеющий сложность $O(n^2)$. Какой именно студент решит реализовывать не принципиально. Можно выбрать из пузырьковой сортировки, сортировки вставками и сортировки выбором. Следует обратить внимание на сложность данных алгоритмов и указать признаки квадратичной сложности для каждого из них.

Пример решения:

```java
public class BubbleSort {
    /**
     * O(N^2)
     */
    public static void sort(int[] array) {
        boolean needSort;
        do {
            needSort = false;
            for (int i = 0; i < array.length - 1; i++) {
                if (array[i] > array[i + 1]) {
                    int temp = array[i];
                    array[i] = array[i + 1];
                    array[i + 1] = temp;
                    needSort = true;
                }
            }
        } while (needSort);
    }
}
```

```java
public class DirectSort {

    /**
     * O(x^2)
     */
    public static void sort(int[] array) {
        for (int i = 0; i < array.length; i++) {
            int minPosition = i;
            for (int j = i + 1; j < array.length; j++) {
                if (array[j] < array[minPosition]) {
                    minPosition = j;
                }
            }
            if (minPosition != i) {
                int temp = array[i];
                array[i] = array[minPosition];
                array[minPosition] = temp;
            }
        }
    }
}
public class InsertionSort {
    public static void sort(int[] array){
        for (int i = 0; i < array.length; i++) {
            for (int j = i + 1; j < array.length; j++) {
                if (array[j] < array[i]) {
                    int temp = array[i];
                    array[i] = array[j];
                    array[j] = temp;
                }
            }
        }
    }
}
```

5. Блок 1.

Задание:
Написать алгоритм быстрого поиска. Стоит еще раз разобрать принцип работы алгоритма, указать на возможность использования рекурсии, т.к. алгоритм работает по принципу «разделяй и властвуй». В зависимости от пожелания студентов, можно писать данный алгоритм вместе с ними, если реализация будет вызывать большие сложности. Но сначала стоит дать попробовать реализовать самостоятельно, согласно схеме.

Пример решения:

```
public class QuickSort {
    public static void sort(int[] array) {
        sort(array, 0, array.length - 1);
    }

    public static void sort(int[] array, int startPosition, int
endPosition) {
        int leftPosition = startPosition;
        int rightPosition = endPosition;
        int pivot = array[(startPosition + endPosition) / 2];
        do {
            while (array[leftPosition] < pivot) {
                leftPosition++;
            }
            while (array[rightPosition] > pivot) {
                rightPosition--;
            }
            if (leftPosition <= rightPosition) {
                if (leftPosition < rightPosition) {
                    int temp = array[leftPosition];
                    array[leftPosition] = array[rightPosition];
                    array[rightPosition] = temp;
                }
                leftPosition++;
                rightPosition--;
            }

        } while (leftPosition <= rightPosition);

        if (leftPosition < endPosition) {
            sort(array, leftPosition, endPosition);
        }
        if (startPosition < rightPosition) {
            sort(array, startPosition, rightPosition);
        }
    }
}
```

Часто встречающиеся ошибки:
Могут возникнуть проблемы с обработкой поинтеров, а именно определения ситуации, когда необходимо закончить итеративные действия в связи с пересечением поинтеров.

6. Блок 1.

Задание:
Пишем тесты для сравнения производительности сортировки больших массивов. Для наглядного результата стоит сравнивать массивы до 100 000 элементов. При таком подходе будет явно видно, какая из сортировок окажется быстрее.

Пример решения:

```
public class AlgorithmDemoApplication {

    public static void main(String[] args) {
        for (int i = 10000; i <= 100000; i = i + 10000) {
            int[] array = new int[i];
            for (int j = 0; j < array.length; j++) {
                array[j] = (int) (Math.random() * 10000);
            }
            Date startDate = new Date();
            BubbleSort.sort(array);
            Date endDate = new Date();
            long bubbleSortDuration = endDate.getTime() -
startDate.getTime();

            for (int j = 0; j < array.length; j++) {
                array[j] = (int) (Math.random() * 10000);
            }
            startDate = new Date();
            QuickSort.sort(array);
            endDate = new Date();
            long quickSortDuration = endDate.getTime() -
startDate.getTime();
            System.out.printf("i: %s, bubble sort duration: %s, quick sort
duration: %s%n", i, bubbleSortDuration,
                    quickSortDuration);
        }
    }
}
```

Примерный результат выполнения:
i: 10000, bubble sort duration: 81, quick sort duration: 0
i: 20000, bubble sort duration: 423, quick sort duration: 1
i: 30000, bubble sort duration: 1129, quick sort duration: 1
i: 40000, bubble sort duration: 2034, quick sort duration: 3
i: 50000, bubble sort duration: 3197, quick sort duration: 2
i: 60000, bubble sort duration: 4733, quick sort duration: 4
i: 70000, bubble sort duration: 6510, quick sort duration: 4

7. Блок 1.

Задание:
После успешной сортировки массива на нем можно использовать бинарный поиск. Необходимо реализовать алгоритм бинарного поиска по элементам. Стоит акцентировать внимание, что т.к. алгоритм использует подход «разделяй и властвуй», его удобно писать с помощью рекурсии. Так что стоит акцентировать внимание на алгоритмическую сложность данного алгоритма, что его выполнение многократно быстрее простого перебора на больших массивах

Пример решения:

```
public static int search(int value, int[] array, int min, int max) {
    int midpoint;
    if (max < min) {
        return -1;
    } else {
        midpoint = (max - min) / 2 + min;
    }

    if (array[midpoint] < value) {
        return search(value, array, midpoint + 1, max);
    } else {
        if (array[midpoint] > value) {
            return search(value, array, min, midpoint - 1);
        } else {
            return midpoint;
        }
    }
}
```

8. Итоги

На этом занятии мы разобрали различные алгоритмы сортировки массивов, рассмотрели их плюсы и минусы, сравнили быстродействие и научились оценивать их сложность. Теперь при принятии решения об использовании сортировки внутри более сложных алгоритмов вы сможете верно оценить ее влияние на общую производительность решения.

9. Домашнее задание

Реализовать алгоритм пирамидальной сортировки (сортировка кучей).

```
public class HeapSort {
   public static void sort(int[] array) {
       // Построение кучи (перегруппируем массив)
       for (int i = array.length / 2 - 1; i >= 0; i--)
           heapify(array, array.length, i);

       // Один за другим извлекаем элементы из кучи
       for (int i = array.length - 1; i >= 0; i--) {
           // Перемещаем текущий корень в конец
           int temp = array[0];
           array[0] = array[i];
           array[i] = temp;

           // Вызываем процедуру heapify на уменьшенной куче
           heapify(array, i, 0);
       }
   }

   private static void heapify(int[] array, int heapSize, int rootIndex) {
```

```
        int largest = rootIndex; // нициализируем наибольший элемент как
корень
        int leftChild = 2 * rootIndex + 1; // левый = 2*rootIndex + 1
        int rightChild = 2 * rootIndex + 2; // правый = 2*rootIndex + 2

        // Если левый дочерний элемент больше корня
        if (leftChild < heapSize && array[leftChild] > array[largest])
            largest = leftChild;

        // Если правый дочерний элемент больше, чем самый большой элемент на
данный момент
        if (rightChild < heapSize && array[rightChild] > array[largest])
            largest = rightChild;
        // Если самый большой элемент не корень
        if (largest != rootIndex) {
            int temp = array[rootIndex];
            array[rootIndex] = array[largest];
            array[largest] = temp;

            // Рекурсивно преобразуем в двоичную кучу затронутое поддерево
            heapify(array, heapSize, largest);
        }
    }
}
```