# CSE 2046

# ANALYSIS OF ALGORITHMS

# HOMEWORK 2

# REPORT

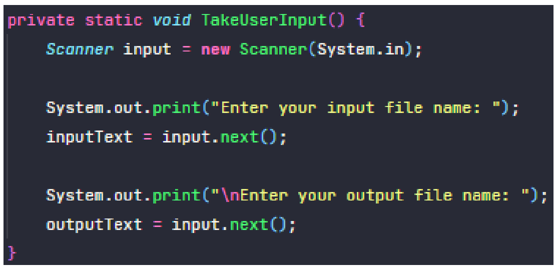
| **STUDENT NAME:** | **ID:** |
|---|---|
| Yasin Tarakçı | 150118055 |
| Yusuf Taha Atalay | 150119040 |
| Ahmet Emre Sağcan | 150119042 |

**Submitted To :** Ömer Korçak

## DESCRIBING OUR PROGRAM

### 1.Taking the Input and Output File:

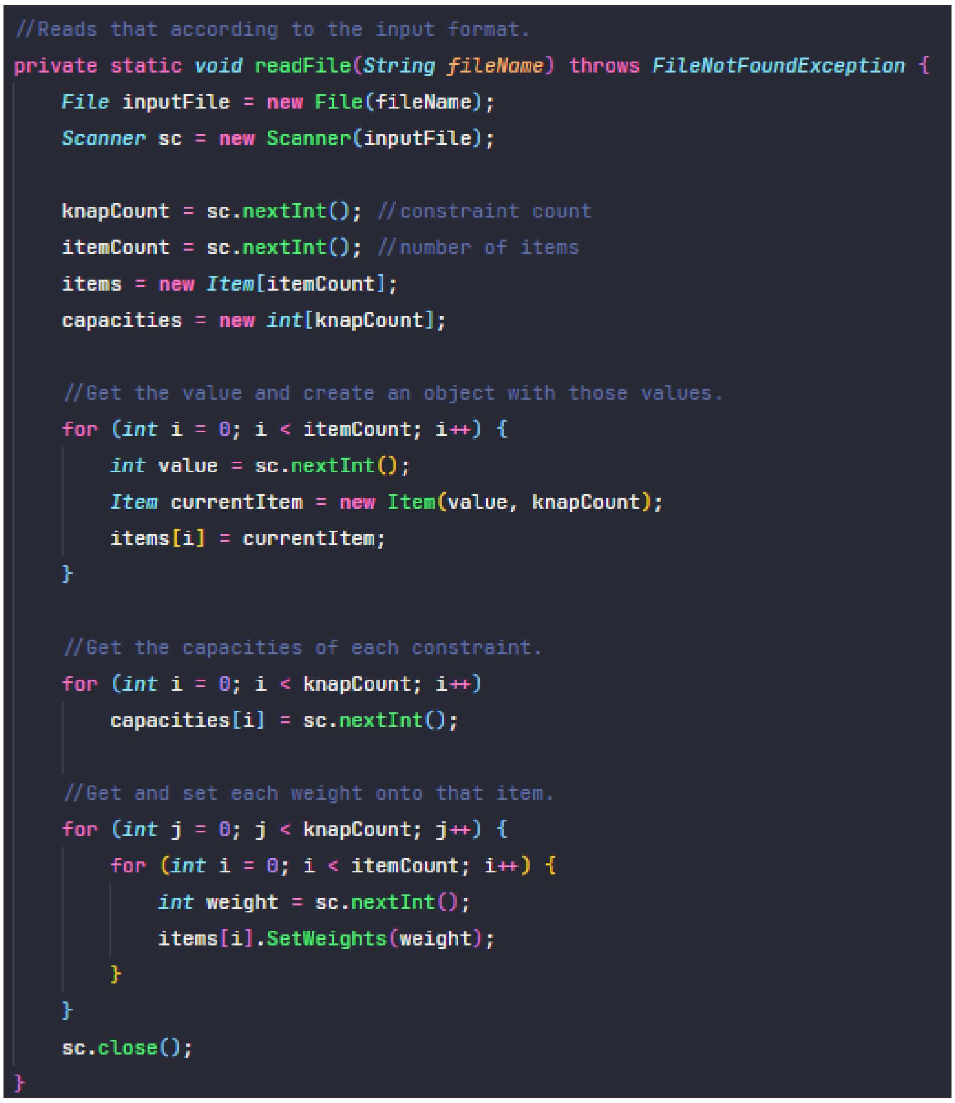
User is asked to enter the name of the input and output file.



```java
private static void TakeUserInput() {
    Scanner input = new Scanner(System.in);

    System.out.print("Enter your input file name: ");
    inputText = input.next();

    System.out.print("\nEnter your output file name: ");
    outputText = input.next();
}
```

### 2.Reading the Input File:

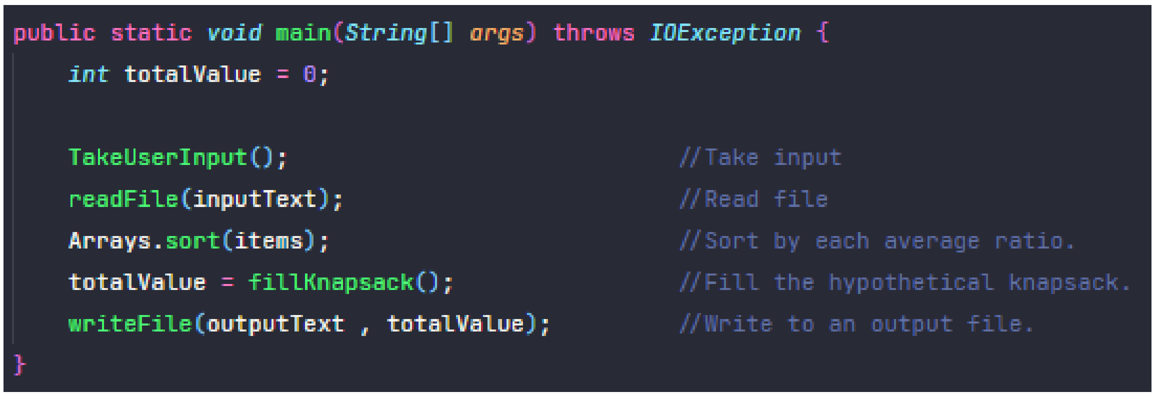
This part reads the input file according to the input format. We store items with their properties on an Item object. Then move them into an items array. Also we store capacities of each constraint on a capacities array.



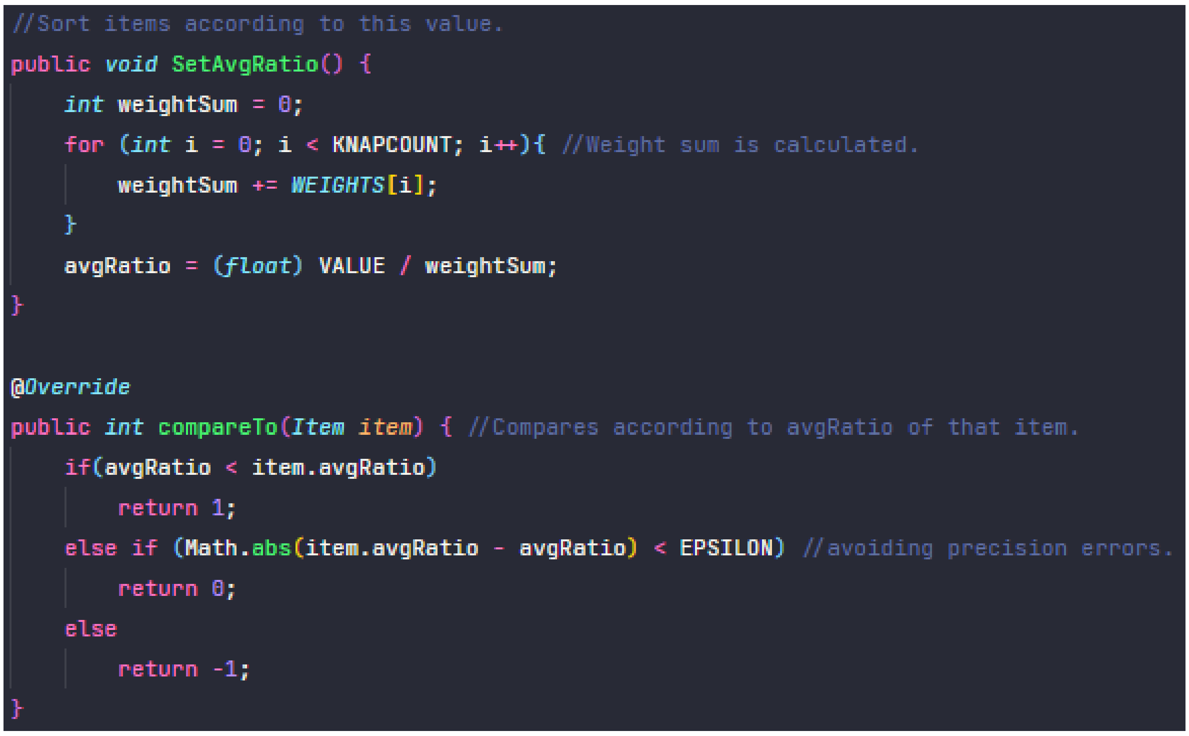
```java
//Reads that according to the input format.
private static void readFile(String fileName) throws FileNotFoundException {
    File inputFile = new File(fileName);
    Scanner sc = new Scanner(inputFile);

    knapCount = sc.nextInt(); //constraint count
    itemCount = sc.nextInt(); //number of items
    items = new Item[itemCount];
    capacities = new int[knapCount];

    //Get the value and create an object with those values.
    for (int i = 0; i < itemCount; i++) {
        int value = sc.nextInt();
        Item currentItem = new Item(value, knapCount);
        items[i] = currentItem;
    }

    //Get the capacities of each constraint.
    for (int i = 0; i < knapCount; i++)
        capacities[i] = sc.nextInt();

    //Get and set each weight onto that item.
    for (int j = 0; j < knapCount; j++) {
        for (int i = 0; i < itemCount; i++) {
            int weight = sc.nextInt();
            items[i].SetWeights(weight);
        }
    }
    sc.close();
}
```

## 3.Sorting the Item Array by a Ratio:

First of all, we sort the array. We are sorting items by their average ratios. When the program is setting the weights of that object, the program also calls the SetAvgRatio() method in that class. It basically sums up the weights of each constraint and then divides it by that object's value. After that; when the sort method is called, compareTo is comparing their avgRatios. Then we have an array that has its elements sorted in a decreasing way.



```java
public static void main(String[] args) throws IOException {
    int totalValue = 0;

    TakeUserInput();                                //Take input
    readFile(inputText);                            //Read file
    Arrays.sort(items);                             //Sort by each average ratio.
    totalValue = fillKnapsack();                    //Fill the hypothetical knapsack.
    writeFile(outputText , totalValue);             //Write to an output file.
}
```

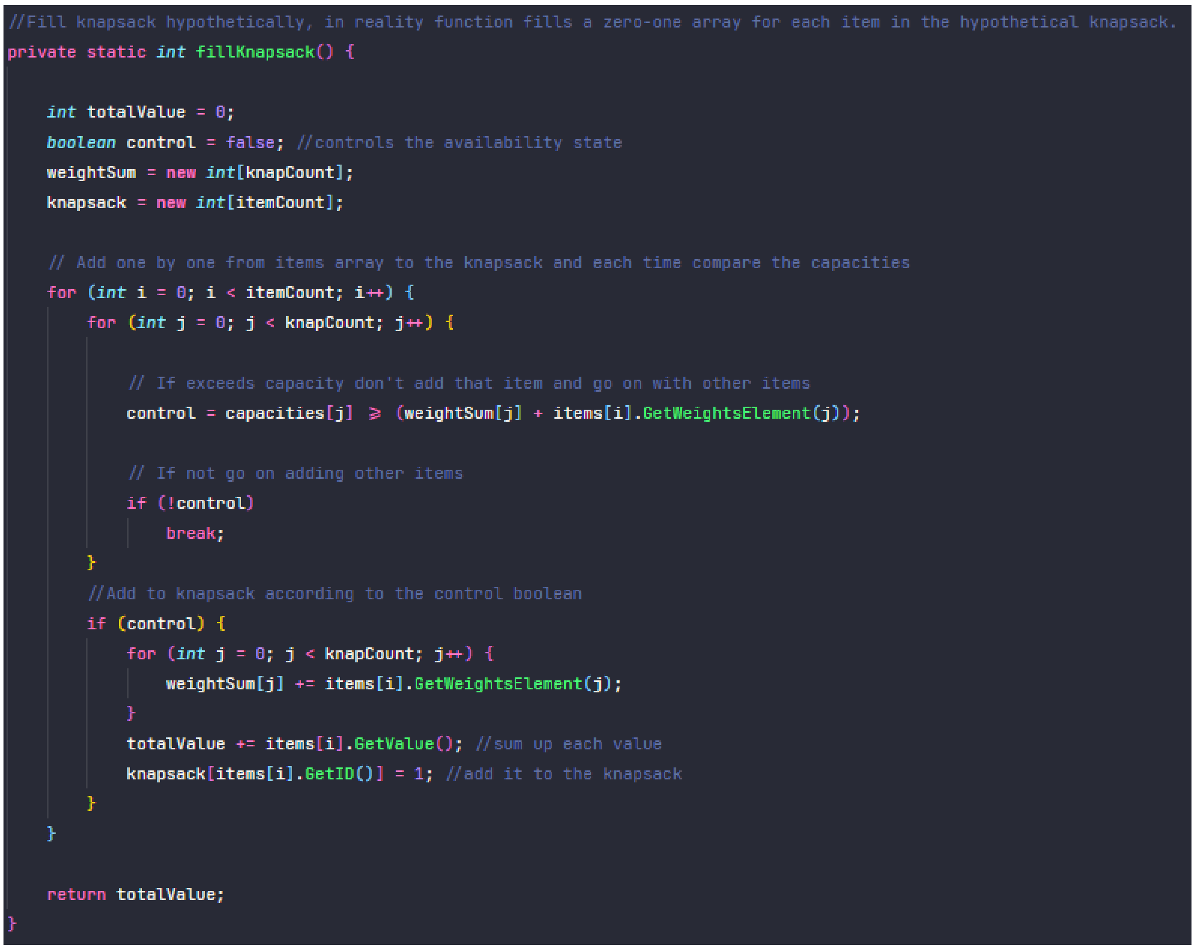


```java
//Sort items according to this value.
public void SetAvgRatio() {
    int weightSum = 0;
    for (int i = 0; i < KNAPCOUNT; i++){ //Weight sum is calculated.
        weightSum += WEIGHTS[i];
    }
    avgRatio = (float) VALUE / weightSum;
}

@Override
public int compareTo(Item item) { //Compares according to avgRatio of that item.
    if(avgRatio < item.avgRatio)
        return 1;
    else if (Math.abs(item.avgRatio - avgRatio) < EPSILON) //avoiding precision errors.
        return 0;
    else
        return -1;
}
```

## 4.Filling the Knapsack:

Our algorithm works with O(nm) time complexity where n is number of items, m is number of constraints. For each item in the sorted items array, we do the following:

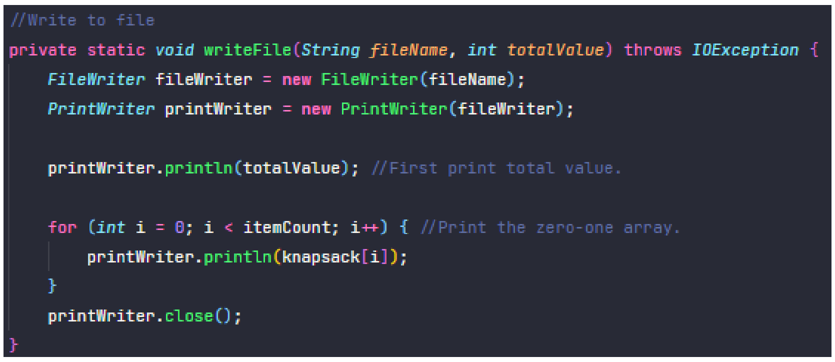
1. Check if that item is gonna exceed the boundaries of at least one constraint by adding to the weightSum value of that constraint.
2. If it exceeds, go on with the next item.
3. If not:
    a. Add that item to the knapsack (set as 1)
    b. Update the weightSum of each constraint
    c. Go on with calculating total value



```java
//Fill knapsack hypothetically, in reality function fills a zero-one array for each item in the hypothetical knapsack.
private static int fillKnapsack() {

    int totalValue = 0;
    boolean control = false; //controls the availability state
    weightSum = new int[knapCount];
    knapsack = new int[itemCount];

    // Add one by one from items array to the knapsack and each time compare the capacities
    for (int i = 0; i < itemCount; i++) {
        for (int j = 0; j < knapCount; j++) {

            // If exceeds capacity don't add that item and go on with other items
            control = capacities[j] >= (weightSum[j] + items[i].GetWeightsElement(j));

            // If not go on adding other items
            if (!control)
                break;
        }
        //Add to knapsack according to the control boolean
        if (control) {
            for (int j = 0; j < knapCount; j++) {
                weightSum[j] += items[i].GetWeightsElement(j);
            }
            totalValue += items[i].GetValue(); //sum up each value
            knapsack[items[i].GetID()] = 1; //add it to the knapsack
        }
    }

    return totalValue;
}
```

## 5.Writing to the Output File:

Writes to the output file according to the output format.



```java
//Write to file
private static void writeFile(String fileName, int totalValue) throws IOException {
    FileWriter fileWriter = new FileWriter(fileName);
    PrintWriter printWriter = new PrintWriter(fileWriter);

    printWriter.println(totalValue); //First print total value.

    for (int i = 0; i < itemCount; i++) { //Print the zero-one array.
        printWriter.println(knapsack[i]);
    }
    printWriter.close();
}
```