流水线题目描述：

一个工厂有 m 条流水线，来并行完成 n 个独立的作业，该工厂设置了一个调度系统，在安排作业时，总是优先执行处理时间最短的作业。

现给定流水线个数 m，需要完成的作业数 n，每个作业的处理时间分别为 t1,t2...tn。请你编程计算处理完所有作业的耗时为多少？

当 n>m 时，首先处理时间短的 m 个作业进入流水线，其他的等待，当某个作业完成时，依次从剩余作业中取处理时间最短的进入处理。

输入描述：

第一行为 2 个整数（采用空格分隔），分别表示流水线个数 m 和作业数 n；

第二行输入 n 个整数（采用空格分隔），表示每个作业的处理时长 t1,t2...tn。

0< m,n<100，0<t1,t2...tn<100。

注：保证输入都是合法的。

输出描述：

输出处理完所有作业的总时长

示例 1

输入：

```
3 5
8 4 3 2 10
```

输出：

```
13
```

说明：

1、先安排时间为 2、3、4 的 3 个作业。

2、第一条流水线先完成作业，然后调度剩余时间最短的作业 8。

3、第二条流水线完成作业，然后调度剩余时间最短的作业 10。

4、总工耗时就是第二条流水线完成作业的时间 13（3+10）。

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String[] strs = in.nextLine().split(" ");
        int m = Integer.parseInt(strs[0]);
        int n = Integer.parseInt(strs[1]);

        String[] split = in.nextLine().split(" ");
        int[] tasks = new int[split.length];
        for (int i = 0; i <split.length; i++) {
            tasks[i] = Integer.parseInt(split[i]);
        }
        Arrays.sort(tasks);
        if (n <= m) {
            System.out.println(tasks[tasks.length - 1]);
            return;
        }

        ArrayList<Integer> res = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            res.add(tasks[i]);
        }

        for (int i = m; i < tasks.length; i++ ) {
            Integer min = new ArrayList<>(new TreeSet<>(res)).get(0);
            int index = res.indexOf(min);
            res.set(index, res.get(index) + tasks[i]);
        }

        // ArrayList<Integer> r = new ArrayList<>(new TreeSet<>(res));
        // System.out.println(r.get(r.size() - 1));

        int minTotalTime = findMinTotalTime(tasks, m);
        System.out.println(minTotalTime);
        in.close();
    }

    public static int findMinTotalTime(int[] tasks, int m) {
        int n = tasks.length;
```

```
        Arrays.sort(tasks);

        int[] pipelines = new int[m];
        int minTotalTime = 0;



        for (int i = 0; i < n; i++ ){
                int minPipeline = findMinPipeline(pipelines);
                pipelines[minPipeline] += tasks[i];
        }

        minTotalTime = findMaxPipeline(pipelines);
        return minTotalTime;
}

private static int findMinPipeline(int[] pipelines) {
        int minIndex = 0;
        int minTime = pipelines[0];
        for (int i = 1; i< pipelines.length; i++) {
                if (pipelines[i] < minTime) {
                        minTime = pipelines[i];
                        minIndex = i;
                }
                        }
        return minIndex;
}

private static int findMaxPipeline(int[] pipelines) {
        int maxTime = pipelines[0];
        for( int i = 1; i< pipelines.length; i++) {
                if (pipelines[i] > maxTime) {
                        maxTime = pipelines[i];
                }
        }
        return maxTime;
}
```

```
}
```