

题目描述：

你有  $n$  台机器编号为  $1 \sim n$ ，每台都需要完成完成一项工作，机器经过配置后都能完成独立完成一项工作。假设第  $i$  台机器你需要花  $B_i$  分钟进行设置，然后开始运行， $J_i$  分钟后完成任务。现在，你需要选择布置工作的顺序，使得用最短的时间完成所有工作。注意，不能同时对两台进行配置，但配置完成的机器们可以同时执行他们各自的工作。

输入描述：

第一行输入代表总共有  $M$  组任务数据（ $1 < M \leq 10$ ）。

每组数第一行为一个整数指定机器的数量  $N$ （ $0 < N \leq 1000$ ）。随后的  $N$  行每行两个整数，第一个表示  $B$ （ $0 \leq B \leq 10000$ ），第二个表示  $J$ （ $0 \leq J \leq 10000$ ）。

每组数据连续输入，不会用空行分隔。各组任务单独计时。

输出描述：

对于每组任务，输出最短完成时间，且每组的结果独占一行。例如，两组任务就应该有两行输出。

示例 1

输入：

```
1
1
2 2
```

输出：

```
4
```

说明：

输入共 3 行数据，第 1 行代表只有 1 组任务；第 2 行代表本组任务只有 1 台机器；第 3 行代表本机器：配置需要 2 分钟，执行任务需要 2 分钟。输出共 1 行数据，代表执行结果为 4 分钟。

示例 2

输入：

```
2
2
1 1
2 2
3
1 1
2 2
3 3
```

输出：

```
4
7
```

说明：

第一行 2 代表输入共 2 组数据，2-4 行代表第 1 组数据，为 2 台机器的配置、执行信息（第 1 台机器：配置需要 1 分钟，执行需要 1 分钟；第 2 台机器：配置需要 2 分钟，执行需要 2 分钟）。5-8 行代表第 2 组数据，为 3 台机器的配置、执行信息（意义同上）。输出共 2 行，分别代表第 1 组任务共需要 4 分钟和第 2 组任务共需要 7 分钟（先配置 3，再配置 2，最后配置 1，执行 1 分钟，共 7 分钟）。

```
// 本题为考试单行多行输入输出规范示例，无需提交，不计分。
import java.util.*;
import javax.annotation.*;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int j = 0;
        List<Integer> data = new ArrayList<Integer>();
        while (in.hasNextInt()) { // 注意，如果输入是多个测试用例，请通过 while 循环处理多个测试用例
            int a = in.nextInt();
```

```

        data.add(a);
    }
    int n = data.get(j++);
    for (int i = 0; i < n; i++) {
        int m = data.get(j++);
        int[][] machines = new int[m][2];
        for (int k = 0; k < m; k++) {
            machines[k][0] = data.get(j++);
            machines[k][1] = data.get(j++);
        }
        sort(machines, m);
        int ans = cal(machines, m);
        System.out.println(ans);
    }
}

private static void sort(int[][] machines, int n) {
    for (int i = 0; i < n; i++) {
        int min = machines[i][1];
        int minIndex = i;
        for (int k = i + 1; k < n; k++) {
            if (machines[k][1] > min) {
                min = machines[k][1];
                minIndex = k;
            }
        }
        if (i != minIndex) {
            swap(machines, i, minIndex);
        }
    }
}

private static void swap(int[][] machines, int i, int j) {
    int[] tmp = new int[2];
    tmp[0] = machines[i][0];
    tmp[1] = machines[i][1];
    machines[i][0] = machines[j][0];
    machines[i][1] = machines[j][1];
    machines[j][0] = tmp[0];
    machines[j][1] = tmp[1];
}

public static int cal(int[][] machines, int n) {
    int last = 0;
    //int n = machines.size();
    int[] dp = new int[n];
    for (int i = 0; i < n; i++) {

```

```

        dp[i] = last + machines[i][0] + machines[i][1];
        last += machines[i][0];
    }
    int ans = 0;
    for (int i = 0; i < n; i++) {
        if (dp[i] > ans) {
            ans = dp[i];
        }
    }
    return ans;
}
}

```

```

class Tas {
    int conf;
    int run;
    //
    public int getConf() {
        return this.conf;
    }
    public int getRun() {
        return this.run;
    }
}
}

```