Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

15th December 2016

Algorithmen & Datenstrukturen Exercise Sheet P13 AS 16

Hand-in: Before Thursday, 22th December 2016 10:00 via the online judge (source code only).

Exercise P13.1 *Jungle.*

Surrounded by dense jungle and wild ravines, you nervously look at your map. A storm is approaching and you need to leave the area as soon as possible ...

On your map, the jungle is divided into a square grid $n \times n$. For each square with coordinates $0 \leq i, j < n$, your map shows the time $t_{i,j}$ it takes you to move into this square. (For simplicity, only *entering* the square costs time and the direction does not change that.) Your starting square is identified by having $t_{i_0,j_0} = 0$ and there is exactly one such square, the other squares satisfy $1 \leq t_i \leq 10\,000$. The size of the map is $1 \leq n \leq 10\,000$.

From any square, you can move in four directions to an adjacent square. When you arrive at a square adjacent to the edge of the map (at any of the four edges), you may immediately leave the jungle and your journey ends. Note that the time the entire journey takes is the sum of required times of the squares you visited, and the starting square needs time 0.

Your task is to find the *minimal time* needed to leave the jungle from the starting square. The number of squares visited is irrelevant, just the time matters. Note that there may be several fastest paths of equal time but we care only about the time of those.

Example

See below for examples of jungles with $n = 8$ and $n = 12$ with a shortest path in gray. The time needed to leave the first jungle is 10, in the second case 236.

9	1	1	9	1	1	9	1
9	1	9	9	9	9	1	9
9	1	1	9	1	1	1	9
9	1	1	9	1	1	9	9
9	1	9	1	1	0	9	9
9	1	1	1	9	1	9	9
9	1	9	1	9	9	9	1
9	9	1	9	1	9	1	42

65	81	23	82	98	41	68	48	85	96	17	22
71	10	34	65	29	39	28	21	50	66	14	86
41	71	44	79	13	78	66	56	12	13	36	29
85	50	99	87	10	94	79	31	14	20	65	91
35	98	41	27	55	23	26	18	63	41	77	50
53	21	83	47	71	10	91	38	60	41	19	68
74	93	58	70	43	24	0	92	96	38	33	93
86	26	51	63	64	44	13	91	57	39	27	63
73	39	48	27	85	50	72	72	70	67	52	35
34	14	81	95	61	80	68	65	17	96	11	31
49	72	22	96	40	99	94	38	15	37	74	50
97	58	80	35	77	72	71	41	78	27	19	38

Input The input consists of several cases. The first line of the file contains the number of cases to follow.

Each case starts with n on a separate line. The next n lines contain n numbers each. i-th line contains the integers $t_{i,j}$ for $j = 0, \ldots, n-1$, separated by spaces.

Output For every test case, write the time required to leave the jungle on a separate line.

Example

Input (for the examples above)

```
2
8
9 1 1 9 1 1 9 1
9 1 9 9 9 9 1 9
9 1 1 9 1 1 1 9
9 1 1 9 1 1 9 9
9 1 9 1 1 0 9 9
9 1 1 1 9 1 9 9
9 1 9 1 9 9 9 1
9 9 1 9 1 9 1 42
12
65 81 23 82 98 41 68 48 85 96 17 22
71 10 34 65 29 39 28 21 50 66 14 86
41 71 44 79 13 78 66 56 12 13 36 29
85 50 99 87 10 94 79 31 14 20 65 91
35 98 41 27 55 23 26 18 63 41 77 50
53 21 83 47 71 10 91 38 60 41 19 68
74 93 58 70 43 24 0 92 96 38 33 93
86 26 51 63 64 44 13 91 57 39 27 63
73 39 48 27 85 50 72 72 70 67 52 35
34 14 81 95 61 80 68 65 17 96 11 31
49 72 22 96 40 99 94 38 15 37 74 50
97 58 80 35 77 72 71 41 78 27 19 38
```

Output

```
10
236
```

Grading You will get 1 bonus point for every 100 judge points, rounded down, with maximum of 200 judge points. The program should run in time $\mathcal{O}(n^2 \log n)$ to get full points. If you want to use a heap, implement your own or use `java.util.TreeSet<>` or `java.util.PriorityQueue<>`[1].

Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=18985&problem=DA_P13.1`, enroll password is "`quicksort`".

Notes For this exercise, we provide a program template as an Eclipse project archive on the lecture website, which will load the input for you.

The archive also contains more test data for your local testing. You can use the provided `Judge.java` program to run your `Main.java` on your computer on all the provided tests – just open and run `Judge.java` in the project as you would run `Main.java`.

[1]For these library structures, you will need to create a `class` for the heap elements and either make the class `Comparable` or make a `Comparator`. Also note that `PriorityQueue` has slow removal of elements other than the queue head, and you should not modify the order of the elements in the structures without removing and reinserting them. If unsure about the programming or timing details, implement your own heap specific to your needs.