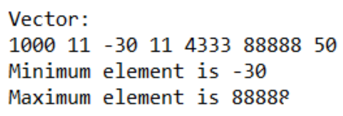
Week 2

Problem 3:

Given a vector we need to find the minimum and the maximum. In order to do that we use a voracious algorithm where we keep dividing the array in 2 and find the maximum and minimum of each sub-vector, to then compare them and take the highest maximum and the lowest minimum.

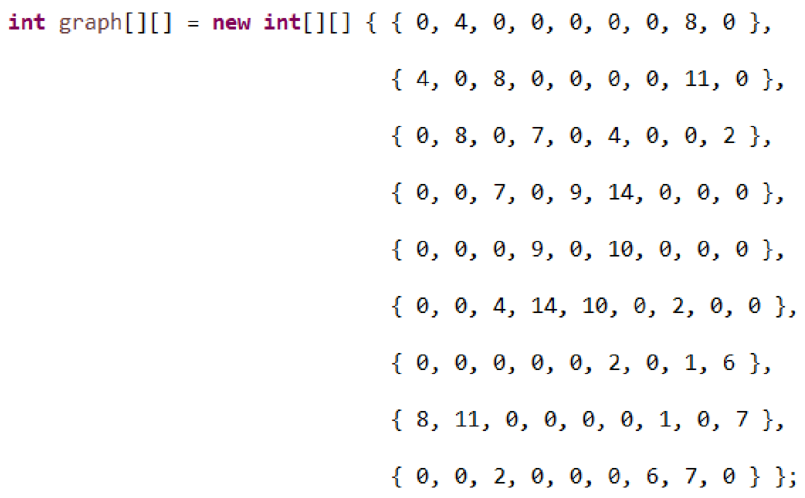


```
Vector:
1000 11 -30 11 4333 88888 50
Minimum element is -30
Maximum element is 88888
```

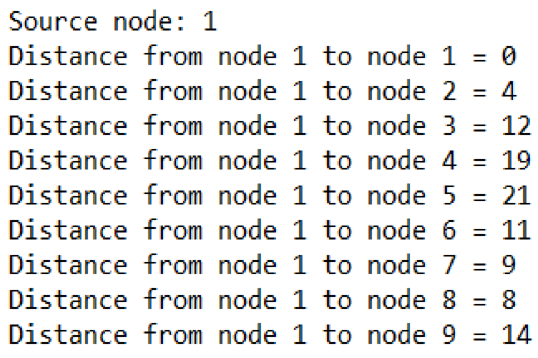
Problem 5

Given a graph, the number of nodes and the source node we use the Dijkstra algorithm to find the minimum path of between the nodes and the source node.

The distances are stored in an array. Another array is created (isPath()) which keeps track of the nodes whose minimum distance to source is calculated. We update this array with the next node not included in isPath() with the minimum distance and we update all the other distances, taking into account the last node´s distance.



```
int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                              { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                              { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                              { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                              { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                              { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                              { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                              { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                              { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
```



```
Source node: 1
Distance from node 1 to node 1 = 0
Distance from node 1 to node 2 = 4
Distance from node 1 to node 3 = 12
Distance from node 1 to node 4 = 19
Distance from node 1 to node 5 = 21
Distance from node 1 to node 6 = 11
Distance from node 1 to node 7 = 9
Distance from node 1 to node 8 = 8
Distance from node 1 to node 9 = 14
```

Problem 6

In order to perform the exercise, we used the priority queue, through which we obtained a heap. With this data structure we can optimally perform the exercise with the complexity of O(nlogn).

In order to obtain the best cost, we just needed to start welding the smallest stairs. In order to do that we took advantage of the priority queue data structure. We just needed to add the 2 smallest elements, remove them from the queue and then add the sum of both, the time invested until now, to the queue.

Finally, we would obtain the cost. The smallest element was obtained with the peek() function and to remove an element we used the remove() function. With both we achieved an O(logn) complexity. However, we also went through the priority queue, O(n) complexity. Like that we have the complexity of O(nlogn).

For example:

Stairs: 6, 7, 8, 9

First welding

6 + 7 = 13

Time = 13

Second welding

13 + 8 = 21

Time = 13 + 21
= 34

Third welding

34 + 9 = 43

Time = 13 + 21 + 43
= 77

The total time is 77.