# Problem Set 3 - Dynamic Programming

Problem D

I remember when first doing this question, going down a completely different (and incorrect path) with using range trees. I had broken down the question, and state d[i][1/0] into the min cost of being at location i, with turning i on (if it's a router), and with turning router i off. I then (outlined further in the comments) designated the state transitions as follows:

```
// Router
   // On - Min between:
       // inheriting off cost of on end of range
       // inheriting on cost of last router within range
   // Off - Min between:
       // INF if there is no router within range
       // inheriting on cost of last router within range
// Not Router
   // On - Min between:
       // inheriting on cost of element before it
   // Off - Min between:
       // INF if there is no router within range
       // inheriting on cost of last router within range
```

I even ended up coding up this solution, using a range tree to find the minimum cost router in the dp cache within the range of the i'th router, for some of the states, and ended up getting it wrong… I remember finding a case for which this strategy would not work, and then scrapped the idea completely.

Ultimately, I think this question just took a different frame of mind, and a complete overhaul of what I had spent the whole day figuring out to get. After a break and some more thinking, I realised I could do this all in one dimension of caching. If I just set d[i] as the minimum cost to connect the first i rooms, I realised I didn't really need any more parameters to complete the recurrence. I could just link it to previous states by either:

1. **Connecting Room i Directly:** If we're connecting the room without relying on any routers, the cost would simply be the cost of connecting all previous rooms (d[i-1]) plus the direct cost of connecting room i (which is just i). This gives the recurrence: d[i] = d[i-1] + i.
2. **OR Using a Router to Cover Room i:** Here, I needed to check if there was a router within range that could cover room i. To do this efficiently, I maintained a queue of potential router positions (r). If a router at position j could cover room i (i.e., it satisfied the range condition j + k >= i), I could calculate the cost of connecting room i using the router at j. This would just require combining the cost of connecting up to the start of the router's range with the cost of placing the router itself: d[i] = min(d[i], d[max(0, j-k-1)] + j).

I reckon this question was a pretty good example to teach me not to add extra parameters when they're not needed lol…