CPSC 320 2017W2: Tutorial Quiz 5 Sample Solutions

March 23, 2018

1 When You Have Eliminated the Uncruftable

In the hit game MyCruft (homes edition), your character has a limited supply S of k types of resources (resources that might be things like pipes, revolvers, caps, clues, and suspects but which we just call $r_1, r_2, \ldots, r_k$). The supply of each resource r_i is a non-negative integer $S[r_i]$.

Your character is also capable of "crufting" resources according to a list of c crufting "formulas". A crufting formula has a non-empty input list of resources (that may contain duplicates if more than one of a particular type of resource is required) and a non-empty output list of resources of the same format. To *apply* a crufting rule, your character uses (removes from your supply) exactly the input list of resources and produces the exact output list of resources.

Note that the same resource may appear in both the input list and the output list. However, **each crufting rule's input list is longer than its output list.**

For example, a crufting rule with input list "pipe, pipe, cap, cap, cap" and output list "clue, cap, cap" would consume two of the resource named "pipe" and three of the resource named "cap" and produce one of the resource named "clue" and two of the resource named "cap". This rule cannot be run if only two or fewer caps are available, even though it only consumes a net of one cap. (With resources named $r_1, r_2, r_3, \ldots$ for "pipe, cap, clue, ..." that rule would be: $r_1, r_1, r_2, r_2, r_2 \to r_3, r_2, r_2$.)

Your goal is to determine the maximum number of resource r_k that it is possible to accumulate via application of crufting rules, starting from S.

1. What is the solution to an instance where there are $k = 4$ resources r_1, r_2, r_3, r_4; the two crufting rules are $r_1, r_1 \to r_2$ and $r_1, r_1, r_2, r_3 \to r_4, r_4, r_3$; and the initial resources S (listed as $[S[r_1], S[r_2], S[r_3], S[r_4]]$) are $[6, 1, 3, 1]$?

   Maximum achievable r_k is ☐.

2. If we were designing a brute force, recursive algorithm to solve this problem, we would want to identify base cases (i.e., "trivial" cases). Fill in the circle next to the **most important** factor that describes a base case instance of this problem.

   ◯ There are no remaining supplies besides r_k
   ◯ The available supplies exactly match the input of a single crufting rule.
   ◯ There are insufficient resources to apply any crufting rule.
   ◯ $k = 0$
   ◯ $c = 0$
   ◯ $n = 0$

   UNGRADED: Design the outline of a brute force, recursive algorithm to solve this problem!

3. With resources $r_1, \ldots, r_k$, what is the smallest possible value for k and an extremely simple crufting rule that would illustrate what could go wrong if this restriction from above were *not* included: "each crufting rule's input list is longer than its output list."

   k is ☐. The crufting rule is ☐ $\rightarrow$ ☐.

4. In a recursive case, a brute force, recursive algorithm will generally find a "first choice" that divides the problem into subproblems. What is the critical "first choice" in a brute force, recursive algorithm to solve this problem? Fill in the circle next to the **best** answer.

   ◯ should we apply crufting rule c now or never apply it again?
   ◯ which crufting rule should we apply next?
   ◯ how many of r_1 should we use up next?
   ◯ should we use up r_1 next or not?
   ◯ which resource should we use next?

5. We can use the insights above to design a brute force, recursive algorithm to solve this problem. What is the maximum number of recursive calls that a **given** call to this algorithm will make (we are **not** asking about the depth of the recursion tree; we want the largest possible number of children of a node) ?

   The maximum number of calls is: ☐.

1.1 Quiz Solution

1. Achievable r_k is [5]. (Use crufting rule 1. Then use crufting rule 2 twice.)
2. The **most important** factor that describes a base case instance of this problem is: **There are insufficient resources to apply any crufting rule.**
3. k is 1. The crufting rule is $r_1 \rightarrow r_1, r_1$. (If we allow this, then there's no bound on the number of r_k resources we can create.)
4. The critical "first choice" in a brute force, recursive algorithm to solve this problem is **which crufting rule should we apply next?**
5. The maximum number of calls is: c.

2 Clique and Claque

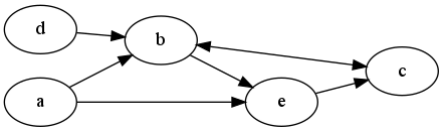
In graph theory, CLIQUE is the problem of finding the largest "clique" (complete subgraph) in a simple graph. So, given an unweighted, undirected graph, find the largest subset of the vertices in the graph such that each pair of vertices in the subset have edges between them.

In CPSC 320, a "claque" is a set of $k > 0$ vertices $\{v_1, \ldots, v_k\}$ in a directed graph such that $\{v_2, \ldots, v_k\}$ have no edges between them but there is an edge (v_i, v_1) for each $v_i \in \{v_2, \ldots, v_k\}$. CLAQUE is the problem of finding the largest claque in a directed, unweighted graph (with no self-edges like (v, v)).

To avoid confusion, we'll write "clique-with-an-I" and "claque-with-an-A" below.

Finally, let the complement of a graph $G = (V, E)$ be $G^c = (V, E')$, where $(u, v) \in E' \leftrightarrow (u, v) \notin E$. In other words, an edge leads from one vertex to another in G^c exactly when **no** edge leads from the first to the second in G. We define this for directed graphs, but it has an exact parallel for undirected graphs where any two vertices are connected in G^c exactly when they're not connected in G.

1. For this (and the next two problems), consider the following graph:



   List the vertices in the largest claque-with-an-A in this graph.

   $v_1 =$ [ ].

   List $\{v_2, \ldots, v_k\}$ in sorted order, separated by commas: [ ].

2. Draw the complement of the graph above. **(Ungraded but useful practice)**
3. Now, imagine that we "erase" the direction of each edge. (So, where there is a directed edge (a, b), there is now an undirected edge $\{a, b\}$. Where there were bidirectional edges (b, c) and (c, b), there is now one undirected edge $\{b, c\}$.) List the vertices in the largest clique-with-an-I in the resulting undirected graph in alphabetical order, separated by commas.

   Vertices: [ ]

4. Complete the following (correct) reduction from CLIQUE to CLAQUE:

To transform an instance of CLIQUE to an instance of CLAQUE:

- Let G^c be the complement of the CLIQUE graph G.
- For each vertex $v \in V$ in G^c, produce [ ] in a new graph G' (which is the CLAQUE instance).
- For each edge $\{u, v\}$ in G^c, produce [ ] and [ ] in G'.
- Add a vertex x to G'. For each [ ] produce [ ].

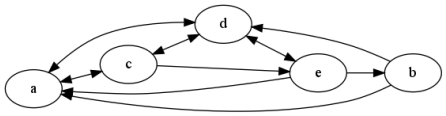
To transform the solution of the CLAQUE instance to a solution to the CLIQUE instance:

Given $\{v_1, v_2, \ldots, v_k\}$ in the CLAQUE solution, let the CLIQUE solution be [ ].

2.1 Quiz solutions

1. $v_1 = b$ and $\{v_2, \ldots, v_k\} = \{a, c, d\}$.

   None of a, c, and d have edges among them but all have edges to b.

2. 

3. $\{a, b, e\}$ or $\{b, c, e\}$.

4. **To transform an instance of CLIQUE to an instance of CLAQUE:**

   - Let G^c be the complement of the CLIQUE graph G.
   - For each vertex $v \in V$ in G^c, produce a vertex v in a new graph G' (which is the CLAQUE instance).
   - For each edge $\{u, v\}$ in G^c, produce (u, v) and (v, u) in G'.
   - Add a vertex x to G'. For each vertex v in G^c, produce an edge (v, x) in G'.

   To transform the solution of the CLAQUE instance to a solution to the CLIQUE instance:

   - Given $\{v_1, v_2, \ldots, v_k\}$ in the CLAQUE solution, let the CLIQUE solution be $v_2, \ldots, v_k$.

3 Gossiping lazily, but surely

UBC students love to gossip. However they are also extremely busy, and so while they want to make sure every other UBC student hears about an interesting piece of gossip that **any** of them has heard, they don't want to spend too much time passing the information around. That is, they want to limit the number of other students they will chat with about the information to be **at most** k, while still ensuring the information is distributed to everybody.

You can think of the set of UBC students as a connected, undirected graph U, where each student is a vertex, and an edge joins two students if they know each other's phone numbers. One student/vertex learns a piece of gossip first, and we want to spread it to everyone.

1. Fill in the blanks below to accurately describe the smallest possible subset of the edges of U that achieves both conditions listed above:

   - everybody hears the gossip no matter who learned about the information first
   - no one needs to have more than k phone conversations about a single piece of information.

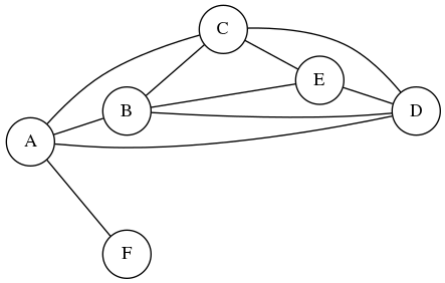
   Together with the vertices, that subset forms a ☐ with a maximum ☐ of ☐.

   It will help to draw and solve several small and trivial examples!

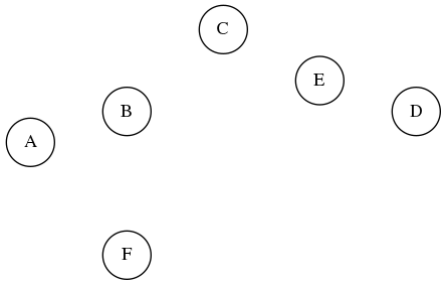
2. Consider the following algorithm whose goal is to find a spanning tree of a graph where every vertex has degree ≤ 3:

```
Sort the edges of G by increasing max degree of their endpoints;
    that is, we compare two edges by comparing the larger-degree
    endpoint of one edge against the larger-degree endpoint of
    the other edge; ties are broken arbitrarily
Initialize each node as its own category in T.
Initialize the category count to |V|.
While we have more than 1 category:
    Remove an edge (u,v) from the list of edges.
    If u and v are not in the same category:
        If degree(u) in T ≤ 3 and degree(v) in T ≤ 3:
            Add the edge (u, v) to T
            Merge u's and v's categories
            Reduce the category count by 1.
```

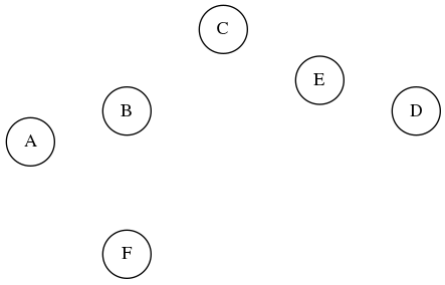
Depending on tie-breaking behaviour, the algorithm can fail to find a tree that meets the degree requirement on the following instance, even though such a tree exists:



Draw a correct tree for this instance here:

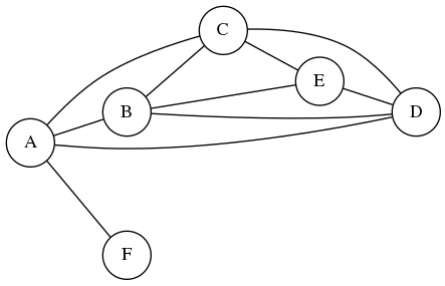


Draw an incorrect answer that the algorithm could produce (depending on tie-breaking behaviour) here:

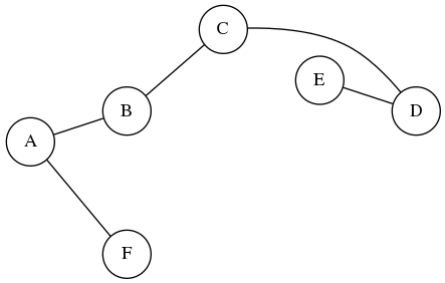


3.1 Quiz solutions

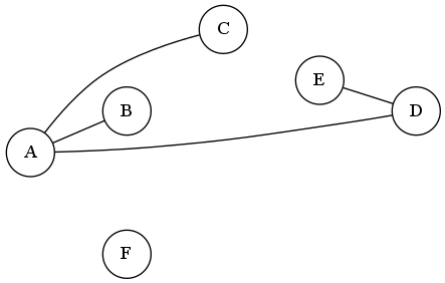
1. Together with the vertices, that subset forms a spanning tree with a maximum degree of k.
2. The instance is:



One correct answer is:



and the answer incorrectly returned by the algorithm is:



because the algorithm might add the edges $\{D, E\}$, $\{A, B\}$, $\{A, C\}$, and $\{A, D\}$ first, and then is unable to add the edge $\{A, F\}$ which is the only way to connect vertex F to the rest of the graph. (Depending on the specific implementation the algorithm might even crash rather than return anything since it will try to remove an additional edge after all edges of the list have already been discarded.)

4 Stable Weddings

Wedding theorists posit that seating arrangements at wedding parties are the most difficult algorithmic problem known to humanity. Let's explore that assertion.

In the "Stable Wedding" problem (SWP), you are given a set of n guests, a list of "disallowed" subsets of the set of guests (each with at least two guests), and a list of positive integer table sizes. You cannot seat **all** the guests in a disallowed subset together at the same table (but you could seat, e.g., all but one of them). You want to find an assignment of guests to tables (that may leave some guests unassigned) that

maximizes the total number of guests at the tables, breaking ties by minimum number of tables used. (If two solutions both seat 10 people but one uses 3 tables and the other 4, the former solution is better. If one solution seats 11 people and the other seats only 10, the former solution is better, regardless of the number of tables used.)

1. Solve the following SWP instance. In your solution, list the guests at each table in ascending order, separated by commas. (E.g., if guests 1, 5, and 3 were at a table, write $1,3,5$ in its blank.)

   The guests are $\{1,2,3,4,5\}$. The disallowed subsets are: $\{1,2\}$, $\{1,3,5\}$, and $\{2,4\}$. There are three tables. Table 1 has 4 seats. Table 2 has 2 seats. Table 3 has 1 seat.

   The optimal solution to the problem is:

   (a) Guests at table 1: ☐ (c) Guests at table 3: ☐

   (b) Guests at table 2: ☐

2. In the Independent Set (IS) optimization problem, we are given a simple (undirected, unweighted) graph and want to find the largest subset of vertices V' such that there is no pair of vertices $u, v \in V'$ that have an edge between them.

   Complete the following reduction from IS to SWP.

   To transform an instance of IS to an instance of SWP: Let the guests in SWP be the ☐ from IS. For each edge $\{u, v\}$ in IS, produce a ☐ in SWP. The table list in SWP should be: ☐.

   To transform the solution of the SWP instance to a solution to the IS instance: Produce ☐.

3. A similar problem called SWP2 is the same as SWP except that you are not allowed to seat **any** of the guests in a disallowed subset at the same table. (They all hate each other.) Fill in the single blank in the following reduction from SWP2 to SWP to make a clear, correct reduction:

   To transform an instance of SWP2 to an instance of SWP: Copy across the guests and tables. For each disallowed subset S_i in the SWP2 instance, copy each [ ] within S_i into the SWP instance as a disallowed subset.

   To transform the solution of the SWP instance to a solution to the SWP2 instance: Simply copy the solution across.

4. Consider the following claim: "If there are at least 2^n seats total at the tables, then it must be possible to seat all the guests."

   Is this claim true or false? Fill in the circle next to the **best** answer.

   ◯ True ◯ False

5. An *inoffensive* guest is one that participates in **no** disallowed subsets.

   Consider the following claim: "For any SWP instance with at least one inoffensive guest and a unique largest table, an optimal solution must exist that seats an inoffensive guest at that largest table."

   Is this claim true or false? Fill in the circle next to the **best** answer.

   ◯ True ◯ False

4.1 Quiz Solution

1. The optimal solution to the problem is:

   (a) Table 1: Guests 2, 3, 5
   (b) Table 2: Guests 1, 4

   Alternatively:

   (a) Table 1: Guests 1, 4, 5
   (b) Table 2: Guests 2, 3

   Or, finally:

   (a) Table 1: Guests 1, 3, 4
   (b) Table 2: Guests 2, 5

2. **To transform an instance of IS to an instance of SWP:** Let the guests in SWP be the vertices from IS. For each edge $\{u, v\}$ in IS, produce a disallowed subset (u, v) in SWP.

   The table list in SWP should be: one table with n seats.

   To transform the solution of the SWP instance to a solution to the IS instance: Produce the guests seated at the one table.

3. **To transform an instance of SWP2 to an instance of SWP:** Copy across the guests and tables.

   For each disallowed subset S_i in the SWP2 instance, copy each pair of guests within S_i into the SWP instance as a disallowed subset.

   To transform the solution of the SWP instance to a solution to the SWP2 instance: Simply copy the solution across.

4. The claim "If there are at least 2^n seats total at the tables, then it must be possible to seat all the guests." is **False**. Consider having a single table with 2^n seats and at least one disallowed subset. We still wouldn't be able to seat every guest.

5. The claim "For any SWP instance with at least one inoffensive guest and a unique largest table, an optimal solution must exist that seats an inoffensive guest at that largest table." is **False**.

   Here's a counterexample:

   Instance:
   - Guests: a, b, c, d, e
   - Disallowed subsets:
     - b, c
     - b, d
     - b, e
   - Tables: 3, 2

   Solution: $\{c, d, e\}, \{a, b\}$

   But, a **cannot** be seated at the first table, or someone else will have to be with b (which no one is willing to do).

5 High-stress, low-stress, no stress

You are managing a consulting team of expert computer hackers, and each week you have to choose a job for them to undertake. Now, as you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g., setting up a Web site for a class at the local elementary school) and those that are *high-stress* (e.g., protecting the nation's most valuable secrets, or helping a desperate group of UBC students finish an assignment on dynamic programming). The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job for your team in week i, then you get a revenue of $l_i \geq 0$ dollars; if you select a high-stress job, you get a revenue of $h_i \geq 0$ dollars. The catch, however, is that in order for the team to take on a high-stress job in week i, it is required that they do no job (of either type) in week $i-1$; they need a full week of preparation to get ready for the crushing stress level. On the other hand, it is okay for them to take a low-stress job in week i even if they have done a job (of either type) in week $i-1$.

So, given a sequence of n weeks, a *plan* is specified by a choice of "low-stress", "high-stress", or "none" for each of the n weeks, with the property if that "high-stress" is chosen for week $i > 1$ then "none" has to be chosen for week $i-1$ (choosing a high-stress job in week 1 is acceptable). The *value* of the plan is determined in the natural way: for each i, you add l_i to the value if you choose "low-stress" in week i, and you add h_i to the value if you choose "high-stress" in week i (you add 0 if you choose "none" in week i).

The problem: Given sets of values $l_1, \ldots, l_n$ and $h_1, \ldots, h_n$, find a plan of maximum value (such a plan will be called *optimal*).

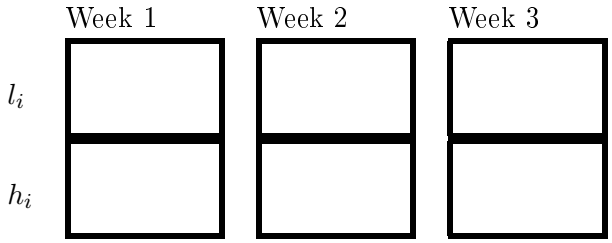
1. [For the group stage only] Give one trivial instance and at least two small instances of this problem. For each instance, indicate the optimal solution.

2. [For the group stage only] Pick a simple, plausible, but suboptimal greedy approach and then generate a counterexample to its optimality.

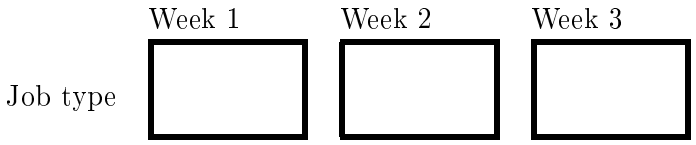
3. Show that the following greedy algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer:

```
i = 1
while i ≤ n:
  if i < n and h_{i+1} > l_i + l_{i+1}:
    output "choose no job in week i"
    output "choose a high-stress job in week i+1"
    i = i + 2
  else:
    output "choose a low-stress job in week i"
    i = i + 1
```

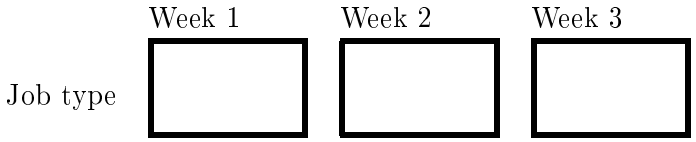
Write the instance here:



Write the correct answer for your instance here:



and the answer incorrectly returned by the algorithm here:



4. Let *nojob*[i], *low-stress-job*[i] and *high-stress-job*[i] denote the values of the best plans for the first i weeks that have no job, a low-stress job, or a high-stress job during week i. Write a recurrence relation for *low-stress-job*[i]. If you want, you can abbreviate *nojob*[i], *low-stress-job*[i] and *high-stress-job*[i] to $nj[i]$, $ls[i]$ and $hs[i]$ respectively.

5.1 Quiz solutions

1. Trivial instance: when $n = 0$, the plan value is 0.

   Small instances:

   - $n = 2$, $l = [3, 4]$ and $h = [4, 6]$: the optimal solution is to choose a high stress job in week 1 (value 4) and a low-stress job in week 2 (value 4) for a total plan value of 8.
   - $n = 2$, $l = [3, 4]$ and $h = [4, 10]$: the optimal solution is to choose no job in week 1 (value 0) and a high-stress job in week 1 (value 10) for a total plan value of 10.

2. See the algorithm from question 3.

3. A counterexample instance is:

	Week 1	Week 2	Week 3
l_i	10	10	5
h_i	5	25	30

   The correct answer for the instance is:

	Week 1	Week 2	Week 3
Job type	low stress	no job	high stress

   which has a total value of 40, and the answer incorrectly returned by the algorithm is:

	Week 1	Week 2	Week 3
Job type	no job	high stress	low stress

   which has a total value of 30.

   Note that you could craft a very different sort of counterexample that instead relies on choosing a high stress job in week 1, since the greedy algorithm never considers that option. (A reasonable extension of our first small instance to have $n = 3$ would work.)

4. If we took a low stress job at week i, we might have taken any kind of job (or none) at week $i-1$. (It would also be reasonable to leave the $nj[i-1]$ term off, since we never get paid a negative amount!)

$$ls[i] = \begin{cases} 0 & \text{when } i = 0 \\ l[i] + \max\{nj[i-1], ls[i-1], hs[i-1]\} & \text{otherwise} \end{cases}$$

6 Astronomy evenings

On most clear days, a group of your friends in the Astronomy department gets together to plan out the astronomical events they are going to try observing that night. We will make the following assumptions about the events.

- There are n events, which for simplicity we will assume occur in sequence separated by exactly one minute each. Thus event j occurs at minute j; if they fail to observe this event at exactly minute j, then your friends miss out on it.
- The sky is mapped according to a one-dimensional coordinate system, measured in "points" along an axis with the initial telescope position as 0; event j will be taking place at point p_j, for some integer value p_j. The telescope starts at point 0 at minute 0.
- The last event n is much more important than the others; so it is required that they observe event n.

The Astronomy department operates a large telescope that can be used for viewing these events. Because it is such a complex instrument, it can only move at a rate of one point per minute. Thus they do not expect to be able to observe all n events; they just want to observe as many as possible, limited by the operation of the telescope and the requirement that event n must be observed.

We say that a subset S of the events is *viewable* if it is possible to observe each event $j \in S$ at its appointed time j, and the telescope has adequate time (moving at its maximum of one point per minute) to move between consecutive events in S.

The problem: given the points of each of the n events, find a viewable subset of maximum size, subject to the requirement that it should contain event n. Such a solution will be called *optimal*.

1. [For the group stage only] Give one trivial instance and at least two small instances of this problem. For each instance, indicate the optimal solution.

2. [For the group stage only] Pick a simple, plausible, but suboptimal greedy approach and then generate a counterexample to its optimality.

3. Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
Mark all events j with |p_n − p_j| > n − j as illegal.
Mark all other events as legal.
Initialize current position to point 0 at minute 0.
While not at end of event sequence:
  Find earliest legal event j reachable from current position.
  Add j to S.
  Update current position to point p_j at minute j.
Return S
```

Write the instance here:

Event	1	2	3	4
Point				

The optimal solution is

and the answer incorrectly returned by the algorithm is:

4. Suppose that at time j you are viewing an event at position p_j. Write a necessary and sufficient condition for being able to view an event at time k, where $k > j$, and no events are viewed in-between times j and k.

6.1 Quiz solutions

1. Trivial instance: $n = 1$, and $p_1 = 0$. The solution is to stay at position 0.

   Small instances:

   - $n = 2$, $p_1 = -1$, $p_2 = 2$: the solution is to ignore event 1, and move the telescope from point 0 to point 2 in time for event 2.
   - $n = 3$, $p_1 = 1$, $p_2 = 2$, $p_3 = 1$: the solution is to move the telescope from point 0 to point 1 in time for event 1, then from point 1 to point 2 in time for event 2, and finally back to point 1 in time for event 3.

2. See question 3.

3. A counterexample instance is:

Event	1	2	3	4
Point	-1	2	3	2

   The optimal solution is to move from point 0 to point 2 in time for event 2, then to point 3 in time for event 3, and finally back to point 2 in time for event 4.

   The answer incorrectly returned by the algorithm is to move to point -1 for event 1, and then back to point 2 for event 4, missing events 2 and 3.

4. You can view the event at position p_k if and only if $|p_k - p_j| \leq k - j$.