

## Task Proposal: Library

Daniel Graf

grafdan@ethz.ch

*Remark:* I submitted this task already to IOI 2016. Since the feedback encouraged a resubmission, here it is again. I only slightly updated the story and the background information.

## 1 Task Description

The city of Tehran is home to the National Library of Iran. With over 90'000 square meters of space, it is one of the largest library campuses in the Middle East<sup>1</sup>.

So far, all the books were ordered by language and then sorted alphabetically within each language. Mina, who is managing the library and is fluent in many different languages, decided that she preferred it if the books were sorted alphabetically across all languages. She already computed how the books would need to be reordered and now she wants to sort the books as quickly as possible.

The library consists of a single bookshelf that is a line of  $N$  equally spaced slots. The slots are numbered from left to right from 0 to  $N - 1$ . Each slot can hold a single book, so there are  $N$  books in total. Mina initially stands in front of slot  $S$  and can never carry more than one book at a time. Moving from one slot to the next slot on the left or on to the right takes Mina one second. Mina can not move past the leftmost or rightmost slot.

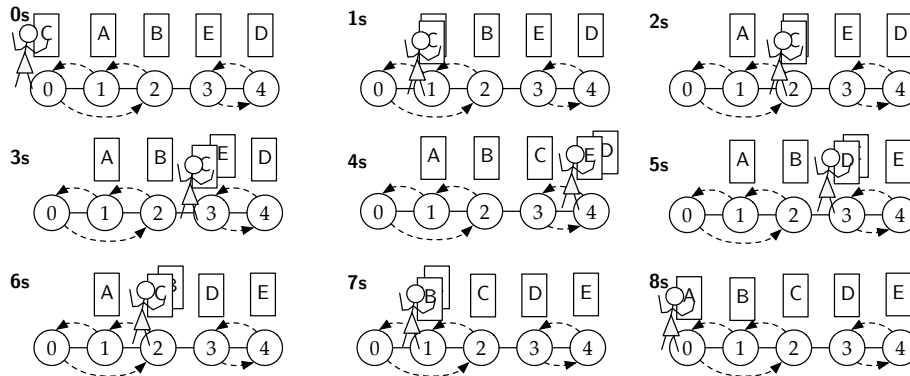
When standing in front of a slot, she can pick up the book in it (assuming the slot is not empty). If she was already carrying a book, she can swap it with the book in the slot. If there is no book in the slot, she can put down the book she was carrying. All of this happens instantaneously. And of course she can also do nothing and continue carrying the same book or continue empty-handedly. The only thing that takes time is movement. It is fine, if Mina picks up the same book multiple times during this sorting process. In the end, Mina wants to be back at slot  $S$ .

Your task is given the starting sequence of the books and the slot Mina stands in front of to find the smallest number of seconds Mina needs to sort all the books and then return to her initial position.

### 1.1 Example

In this example, we have  $N = 5$  and Mina starts at slot  $S = 0$ . The books are called A, B, C, D, E and are initially ordered CABED. One of the optimal solutions is shown in the picture. Mina first takes book C and moves it to slot 3 where she swaps it for book E. She can then bring book E to slot 4, book D to slot 3, book C to slot 2, book B to slot 1 and finally book A to slot 0. This trip takes her 8 seconds and there is no way to do it any faster.

<sup>1</sup>according to [https://en.wikipedia.org/wiki/National\\_Library\\_of\\_Iran](https://en.wikipedia.org/wiki/National_Library_of_Iran)



## 1.2 Task

You are given  $N$ ,  $S$  and the order in which the books are initially placed. Compute the smallest number of seconds Mina needs to sort all the books and return to slot  $S$ . You need to implement the function `booksort`:

- `booksort(N,S,order)` – This function will be called by the grader exactly once.
  - $N$ : the number of slots and number of books
  - $S$ : the slot where Mina starts from
  - `order`: an array of length  $N$ . `order[0], …, order[N-1]` give the initial order of the books, where the books are numbered from 0 to  $N - 1$  in alphabetic order. This means that the book initially placed at slot  $i$  should end up at slot `order[i]`.
  - The function should return the smallest number of seconds in which Mina can complete her task.

## 1.3 Subtasks

subtask	points	$N$	$S$
1	10	$1 \leq N \leq 4$	$S = 0$
2	10	$1 \leq N \leq 7$	$S = 0$
3	15	$1 \leq N \leq 1'000$	$S = 0$
4	15	$1 \leq N \leq 1'000'000$	$S = 0$
5	20	$1 \leq N \leq 1'000$	$0 \leq S < N$
6	30	$1 \leq N \leq 100'000$	$0 \leq S < N$

### 1.3.1 Sample grader

The sample grader reads the input in the following format:

- line 1:  $N$   $S$
- line 2: `order[0] … order[N-1]`

The sample grader prints the return value of `booksort`.

## 2 Description of Desired Solutions

### 2.1 Easy subtasks (subtasks 1 and 2)

The first subtask, where there are at most 4 books in the library, can easily be solved by hand. There are only  $1! + 2! + 3! + 4! = 33$  possible inputs and so they can all be precomputed by hand simulating the process with pen and paper.

The second subtask allows a brute force state exploration. We can take the triple (state of the shelf, Minas position, Minas current book) as the state in a breadth first search. This state space is roughly of size  $(N + 1)! \cdot N$  and so for  $N \leq 7$  we can still find the shortest path to the sorted state fast enough. See `library_20.cpp` for an implementation.

### 2.2 Medium subtask (start/end on the very left)

To compute the number of steps (=seconds) needed without actually simulating the entire sorting process, we need a couple of observations:

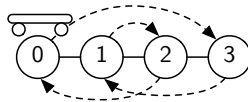
**Balancing property** Across every edge of the underlying path graph, Mina will walk equally often from left to right as she walks right to left. A similar balance also applies to the books. For every edge of the path, there are equally many books that need to be moved from left to right as there are from right to left.

**Cycles of the permutation** The array `order` specifies a permutation. We will denote that permutation by  $\pi$  from here on. We will see, that the answer only depends on how  $\pi$  partitions the set of slots  $\{0, \dots, n - 1\}$  into disjoint cycles. A book that is placed correctly from the beginning, we call *trivial*. Their corresponding trivial cycles (cycles of length one) can almost be ignored, as we will always just move past them.

**Single cycle** If  $\pi$  consists of a single cycle then the answer is easy to find: Mina can grab the book at  $S$ , bring it to  $\pi(S)$ , take the book from there to  $\pi(\pi(S))$  and so on until she returns to  $S$ . [\[Here is a video illustrating this.\]](#) As she brings one book on slot closer to its target position in every step, her walk surely is optimal. We can compute this number of steps by

$$d(\pi) = \sum_{i=0}^{n-1} |i - \pi(i)|$$

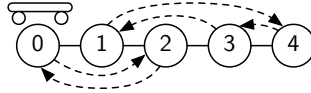
which is exactly the sum of the distances between initial and target position of every book.



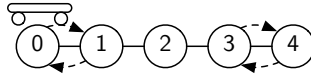
**Lower bound  $d(\pi)$**  The sum of distances  $d(\pi)$  is a lower bound on the answer even if  $\pi$  consists of multiple cycles. We distinguish two kinds of steps for Mina: A step is called *essential* if Mina brings one book one step closer to its target position than this book ever was before. Otherwise the step is called *non-essential*. Every way of sorting the books consists of exactly  $d(\pi)$  many essential steps. The number of non-essential steps needed depends on how the cycles of  $\pi$  overlap.

**Two cycles** Every cycle of  $\pi$  covers some interval  $I$  of the bookshelf which extends from the leftmost book to the rightmost book that are part of this cycle. We have  $I \subseteq [0, n - 1]$ , where we use  $[i, j]$  as a shorthand for  $\{i, i + 1, \dots, j - 1, j\}$ . Let  $\pi$  consist of exactly two non-trivial cycles  $C_1$  and  $C_2$  with their respective intervals  $I_1, I_2$  and let  $S = 0$  with  $S \in C_1$ . Then the answer only depends on whether the cycles overlap ( $I_1 \cap I_2 \neq \emptyset$ ) or not. If they overlap, the answer is  $d(\pi)$  otherwise it is  $d(\pi) + 2$ .

Why? If they overlap Mina can sort along  $C_1$  until she encounters the first book belonging to  $C_2$ . She then leaves the book she was carrying at that slot to fully sort  $C_2$  and return to the same slot. She can then pick up that book again and finish sorting  $C_1$  without ever spending a non-essential step.



If the two cycles don't overlap (so  $C_1$  is entirely to the left of  $C_2$ ), Mina can do something similar. She starts sorting  $C_1$  until she encounters the rightmost slot belonging to  $C_1$ . She then takes the book from there and non-essentially walks with one step to the right to the leftmost slot of  $C_2$ . There, she sorts  $C_2$  and picks up the same book again to non-essentially walk back to  $C_1$ . Finally, she finishes sorting  $C_1$  and returns to  $S$ . This is optimal since the only two non-essential steps of Mina's walk are spent across an edge that no book needs to cross but has to be crossed by Mina eventually as there are non-trivial books on both sides. [\[Here is a video illustrating this.\]](#)



**Multiple cycles** The two cases from before generalize to the case where  $\pi$  consists of many cycles. Any two overlapping cycles can be interleaved without non-essential steps and Mina has to spend two non-essential steps across every edge that no book needs to cross but on both sides there are either some non-trivial books or  $S$ . [\[Here is a video illustrating this.\]](#) More formally, let  $E'$  be the subset of the edges of the path with the following property:

$$\begin{aligned}
 e = (i, i + 1) \in E' &\Leftrightarrow \text{no book has to cross } e \text{ and} \\
 &\quad \text{some book to the left of } e \text{ is non-trivial or at } S \text{ and} \\
 &\quad \text{some book to the right of } e \text{ is non-trivial or at } S \text{ and} \\
 &\Leftrightarrow ((\nexists j \in [0, i] : \pi(j) \geq i + 1) \wedge (\nexists j \in [i + 1, n - 1] : \pi(j) \leq i)) \text{ and} \\
 &\quad (\exists l \in [0, i] \text{ s.t. } (l \neq \pi(l) \vee l = S)) \text{ and} \\
 &\quad (\exists r \in [i + 1, n - 1] \text{ s.t. } (r \neq \pi(r) \vee r = S))
 \end{aligned}$$

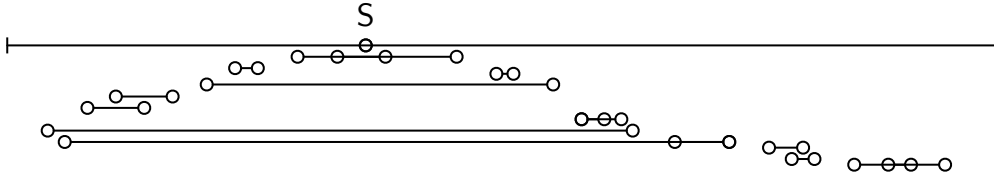
If  $S = 0$ , these are all the non-essential steps needed, so the answer is  $d(\pi) + 2 \cdot |E'|$ .

**Implementation** With these observations, it is easy to compute both  $d(\pi)$  and  $|E'|$ . If it is done in quadratic time (e.g. by just checking the above conditions for  $E'$  by looping over all indices for every edge), this will solve subtask 3 (see `library_35.cpp` for an implementation). However, it is not hard to compute  $E'$  in linear time (e.g. in a scanline fashion from left to right), which will then score for the first four subtasks (see `library_50.cpp` for an implementation).

### 2.3 Harder subtasks (with $S \neq 0$ )

If Mina does start somewhere in the middle of the shelf we might need some additional non-essential steps across edges not in  $E'$ . It is not obvious whether Mina should first go to the left

or to the right as there might be non-trivial books on both sides.



We could try out both options and define the following subproblem:

*How many non-essential steps do we need, if we already know how to connect all the cycles in the interval  $[l, r]$  with  $S \in [l, r]$ ?*

This gives rise to a *dynamic programming* formulation with a state of quadratic size (all intervals containing  $S$ ).

For any given interval, we define the function  $\text{extend}(l, r)$  with  $[l', r'] = \text{extend}(l, r)$  being the largest part of the shelf that we can sort without spending any additional non-essential steps. So  $\text{extend}$  has to repeatedly add all cycles  $C$  whose interval  $I$  partially or fully overlap with  $[l, r]$  and then continue with  $[l, r] := [l, r] \cup I$  until no more cycles can extend the interval.

Once there is no other overlapping cycle (so  $[l, r] = \text{extend}(l, r)$ ), we are either done or we know that we have to spend some non-essential steps. Let  $\text{combine}(l, r)$  be the function that computes the cost of connecting all cycles to the interval  $[l, r]$ . We can recursively compute it using  $\text{combine}(l, r) = 2 + \min(\text{combine}(l - 1, r), \text{combine}(l, r + 1))$ . We need to take care of the border cases (when  $l - 1$  or  $r + 1$  are outside the shelf) and initialize it with  $\text{combine}(l', r') = 0$  for  $[l', r']$  being the smallest interval that contains all non-trivial books and  $S$ .

By implementing  $\text{extend}$  carefully, we can achieve an amortized constant time complexity across all calls, so that the dynamic program runs in quadratic time overall. The code in the file `library_70.cpp` implements this using memoization. Note that for  $S = 0$ , the set of states is only of linear size, so this solution also passes subtask 4.

To solve the problem in linear time, we note that we can decide whether to go left or right somewhat locally without exploring quadratically many states. If  $[l, r]$  is some extended interval ( $[l, r] = \text{extend}(l, r)$ ), we look for two special cycles  $C_l$  and  $C_r$ .  $C_l$  is the first  $S$ -containing<sup>2</sup> cycle that we encounter, when walking from  $l$  to the left. Similarly,  $C_r$  is the first  $S$ -containing cycle when walking from  $r$  to the right.

Let  $c_l$  be the cost of reaching  $C_l$  from  $l$  (and define  $c_r$ ). Note that  $c_l$  is not just twice the distance between  $l$  and the closest book of  $C_l$  as there might be some small cycles along the way that help us save some non-essential steps. But we can compute  $c_l$  quickly by solving the ( $S = 0$ )-problem between  $l$  and the first box of  $C_l$ .

Observe that if  $C_l$  does not exist,  $C_r$  does also not exist (as  $[l, r]$  is maximally extended, the book of  $C_l$  to right of  $S$  also has to be to the right of  $r$  and vice versa). Also note that once we reach either  $C_l$  or  $C_r$ , we also reach  $C_l \cup C_r$  and therefore get  $\text{extend}(C_l \cup C_r)$  without any further cost. This means that we can greedily decide for the cheaper of the two sides (of cost  $\min(c_l, c_r)$ ) and then continue with the interval  $\text{extend}(C_l \cup C_r)$  regardless of whether we decided to go left or right.

Finally, one has to take care of the *border* region, everything outside of the outermost  $S$ -containing cycles (so once  $C_l$  and/or  $C_r$  no longer exist). But this is easy, as this is just another ( $S = 0$ )-case on each side.

<sup>2</sup>A cycle  $C$  with interval  $I$  is  $S$ -containing if and only if  $S \in I$ .

We can answer all the `extend(l, r)` calls and compute all the  $c_l$  and  $c_r$  costs using only one overall sweep over the shelf (if we precompute the cycle of each shelf and the interval of each cycle, which we can also do in  $\mathcal{O}(n)$ ). Therefore, we can find determine the answer  $d(\pi) + \text{combine}(S, S)$  in linear time. The code in the file `library_100.cpp` implements this optimal solution.

## 2.4 Almost correct solutions

One thing a contestant might overlook is that the answer can be of the order  $\Theta(n^2)$  and therefore does not necessarily fit into a 32-bit integer. This causes an overflow in subtasks 4 and 6.

Other submissions might consider only near optimal sorting walks. Some might visit also all the trivial books at the ends of the paths, even though there is nothing to sort there. Others might just traverse the path once without carrying any book and then sort every cycle individually along the way without interleaving anything and hence spending  $2n - 2$  many non-essential steps. Both of these would result in non-optimal answers.

## 2.5 Overview

To summarize, here are the solution techniques listed per subtask:

subtask	points	technique
1	10	pen and paper
2	10	state exploration, BFS
3	15	ad hoc
4	15	ad hoc
5	20	dynamic programming
6	30	ad hoc / greedy

### 3 Testdata Generation

I can see two challenges when creating test sets for this task:

- The worst case running time of  $\mathcal{O}(n^2)$  of the dynamic programming solution for subtask 5 is only achieved for certain permutations with a clever nesting of the cycles.
- Some off-by-one-bugs might not immediately result in a wrong answer if in the particular test cases they just happen to occur along edges that do not need any non-essential steps.

The directory `testdata` contains a rudimentary test data generator for this tasks. It creates inputs of three different types:

- Random permutations (w.h.p. contains one very large cycle)
- Concatenations of constant-size random permutations (breaks many  $\mathcal{O}(n^2)$  approaches)
- $\approx \sqrt{n}$  many random permutations of size  $\approx \sqrt{n}$  nested in between the elements of another random permutation of size  $\approx \sqrt{n}$  (to get at least two layers of nested  $S$ -containing cycles)

In my preliminary tests, this was already enough to clearly distinguish my implementations for all six subtasks w.r.t to running time and correctness. But of course, testing against potentially good heuristics would need to be done before giving this task at a contest.

Take a look at the `README` in `testdata` directory to see how to compile, run and verify the test sets. In a nutshell, this is how it should work:

```
$ ./build.sh
$ ./generate.sh
$ ./verify_solutions.sh
[ Verify solutions with timelimit 2s ]
[ lib100 ]
-- verify solution tests/subtask1.in: [Correct]
-- verify solution tests/subtask2.in: [Correct]
-- verify solution tests/subtask3.in: [Correct]
-- verify solution tests/subtask4.in: [Correct]
-- verify solution tests/subtask5.in: [Correct]
-- verify solution tests/subtask6.in: [Correct]
[ lib70 ]
-- verify solution tests/subtask1.in: [Correct]
-- verify solution tests/subtask2.in: [Correct]
-- verify solution tests/subtask3.in: [Correct]
-- verify solution tests/subtask4.in: [Correct]
-- verify solution tests/subtask5.in: [Correct]
-- verify solution tests/subtask6.in: [TLE]
[ lib50 ]
-- verify solution tests/subtask1.in: [Correct]
-- verify solution tests/subtask2.in: [Correct]
-- verify solution tests/subtask3.in: [Correct]
-- verify solution tests/subtask4.in: [Correct]
-- verify solution tests/subtask5.in: [WA]
-- verify solution tests/subtask6.in: [WA]
[ lib35 ]
-- verify solution tests/subtask1.in: [Correct]
-- verify solution tests/subtask2.in: [Correct]
-- verify solution tests/subtask3.in: [Correct]
-- verify solution tests/subtask4.in: [TLE]
-- verify solution tests/subtask5.in: [WA]
-- verify solution tests/subtask6.in: [TLE]
[ lib20 ]
-- verify solution tests/subtask1.in: [Correct]
-- verify solution tests/subtask2.in: [Correct]
-- verify solution tests/subtask3.in: [TLE]
-- verify solution tests/subtask4.in: [TLE]
-- verify solution tests/subtask5.in: [TLE]
-- verify solution tests/subtask6.in: [TLE]
```

## 4 Background Information

### 4.1 Author Information

- Author: Daniel Graf
- E-Mail: [d.graf.vilters@gmail.com](mailto:d.graf.vilters@gmail.com), [daniel.graf@inf.ethz.ch](mailto:daniel.graf@inf.ethz.ch) or [daniel@soi.ch](mailto:daniel@soi.ch)
- Affiliation: PhD student at Department of Computer Science at ETH Zürich
- Olympiad-Role: president of the Swiss Olympiad in Informatics (2011-2016), IOI participant for Switzerland 2009 (bronze medal), IOI team leader for Switzerland 2012, 2014, 2015, 2016, ACM ICPC participant at SWERC 2011, 2013, 2014 and World Finals 2014 and 2015

### 4.2 Task Background

This task came up as part of my Master thesis during summer 2015 where I looked at scheduling and sorting algorithms for robotic warehouses, specifically a robot system for automatic bicycle parking called [BikeLoft](#).

The story is basically that a robot manages a set of boxes that can each hold a single bicycle and can be stored along a path of  $n$  slots. At one end of the path there is door where customers can bring and pick up there bicycles. [\[Here is a video illustrating this.\]](#) [\[And here is video documentary about the actual prototype of the system.\]](#)

Algorithmic scheduling questions arose along the lines of: "Where should the robot put a certain box if he knows that this customer arrives in five minutes?", "How can the robot keep enough empty boxes near the door to be ready quickly when a new customer wants to store his bike?". The sorting aspect came from the assumption that for typical bike parkings (e.g. at a train station) most people bring their bike in the morning and reclaim it in the evening. Hence, you might be able to compute the optimal reordering that you should do over lunch to be ready for the evening. How long does the robot need to perform this optimal reordering over lunch? [\[Here another video illustrating this bicycle sorting.\]](#)

I then studied and solved this sorting problem of labelled tokens on a graph for the case where the underlying graph is a path, a cycle or a tree and I showed that it is NP-hard for general planar graphs.

Here are some links to what I did as part of my thesis:

- [Master thesis writeup](#)
- [Master thesis presentation slides](#)
- [ESA 2015 publication about sorting on paths and trees \[3\]](#)
- [ESA 2015 presentation slides](#)
- [Implementation and visualization of the solution on paths and trees](#)

In all of them, I only look at the case of  $S = 0$  for paths (as this corresponds to the motivation of having a door on one end where the robot needs to be). The sections about sorting on trees include the case of paths with  $S \neq 0$ , but my algorithm there runs in  $\Theta(n^2)$  and relies on Edmond's algorithm for optimum branchings (which is outside the IOI syllabus). The case corresponding to the hardest subtasks (on a path with  $S \neq 0$  in linear time), I have not published anywhere yet, but it is part of a journal paper that is currently under review.



As far as I am aware, this task has never appeared at a programming competition. The closest related problem that I could find is the task **Boxes/Kuti je** from the *Croatian National Competition in Informatics in 2006* which is available in Croatian [\[here\]](#) and I have discussed in Section 3.4.3 of my thesis.

The task **Baggage** from ICPC World Final 2014 has a similar setting, but the details (sorting  $2n$  black-and-white-alternating marbles while always moving two neighboring marbles at once) make the solution totally different. I refer to Section 3.4.4 of my thesis for detailed discussion.

The task **Boxes** from IOI 2015 is quite similar in the way the problem is phrased (with the graph being a cycle instead of a path and with the server having a capacity bigger than one). But again, the actual problem and solution are substantially different in my opinion.

While I have talked about the case of  $S = 0$  on a few occasions (my Master thesis defense, at ESA15, some people from my research group at ETH<sup>3</sup>), I took special care not to mention the  $S \neq 0$  case to anyone but two people (my Professor Dr. Peter Widmayer (chair of IOI 1992) and my room mate Sandro Feuz (IOI-contestant 2007)). I also never mentioned the specifics of my Master thesis topic at any point in discussions with any of the contestants of the Swiss olympiad.

Very similar results were known already since the 1980s. This paper [\[1\]](#) [\[Link\]](#) by Atallah and Kosaraju solves the problem on paths and cycles in linear time (but the implementation would be more intricate as the path is just a special case of an elaborate solution on the cycle). Another series of papers by Frederickson and Guan, most importantly in [\[2\]](#) [\[Link\]](#), solves the problem on trees in time  $\mathcal{O}(n \log n)$ . However, I could not find anything on anyone ever implementing any of these solutions so far.

The feedback to the submission of this task to IOI 2016 mentioned that the tree version of this problem was recently used in an opencup.ru contest. I could not find this task online.

## References

- [1] Mikhail J Atallah and S Rao Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5):849–869, 1988.
- [2] Greg N Frederickson and DJ Guan. Ensemble motion planning in trees. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 66–71. IEEE, 1989.
- [3] Daniel Graf. How to sort by walking on a tree. In *Algorithms–ESA 2015*, pages 643–655. Springer, 2015.

---

<sup>3</sup>None of these intersect with the set of potential IOI-contestants to the best of my knowledge.

## 5 Sample Implementations

### 5.1 Easy subtasks (subtasks 1 and 2): State exploration with BFS

```
1 // Library solution for S=0 in O((N+2)!)
2 // Score: 20
3 // Use BFS over the entire state space
4
5 #include "library.h"
6 #include <vector>
7 #include <queue>
8 #include <map>
9 #include <iostream>
10
11 using namespace std;
12
13 struct state {
14     int p; // Minas position
15     int b; // Minas current book
16     vector<int> s; // shelf
17     bool operator < (const state &o) const {
18         if(p != o.p) return p<o.p;
19         if(b != o.b) return b<o.b;
20         return s<o.s;
21     }
22     bool operator == (const state &o) const {
23         return !(*this < o) && !(o < *this);
24     }
25     void print() const {
26         cout << "state: p=" << p << " b=" << b << " s=[";
27         for(int i=0; i<s.size(); i++) {
28             if(i>0) cout << ",";
29             cout << s[i];
30         }
31         cout << "]";
32     }
33 };
34
35 long long int booksort(int N, int S, vector<int> order) {
36     // BFS over the state space
37     map<state, int> M; // Distance map for discovered states
38     queue<state> Q; // BFS queue
39
40     vector<int> initial(N);
41     for(int i=0; i<N; i++) initial[i]=order[i];
42     state start = {0, -1, initial};
43
44     vector<int> sorted(N);
45     for(int i=0; i<N; i++) sorted[i]=i;
46     state target = {0, -1, sorted};
47
48     if(start==target) return 0;
49
50     M[start] = 0;
51     Q.push(start);
52
53     while(!Q.empty()) {
54         state s = Q.front(); Q.pop();
55         vector<state> succ;
56         // generate all possible successor states
```

```

57 state ns = s;
58 for(ns.p = s.p-1; ns.p <= s.p+1; ns.p++) { // next position +/- 1
59     if(!(0<=ns.p && ns.p<N)) continue; // don't move out of the shelf
60
61     swap(ns.b, ns.s[s.p]);
62     succ.push_back(ns); // swap only before the step
63     swap(ns.b, ns.s[s.p]);
64
65     swap(ns.b, ns.s[ns.p]);
66     succ.push_back(ns); // swap only after the step
67     swap(ns.b, ns.s[ns.p]);
68
69     swap(ns.b, ns.s[s.p]); swap(ns.b, ns.s[ns.p]);
70     succ.push_back(ns); // swap before and after the step
71     swap(ns.b, ns.s[ns.p]); swap(ns.b, ns.s[s.p]);
72 }
73 // explore all new successor states
74 for(auto & ns : succ) {
75     if(M.find(ns) == M.end()) {
76         M[ns] = M[s] + 1;
77         Q.push(ns);
78     }
79     if(ns == target) {
80         return M[s] + 1;
81     }
82 }
83 }
84 return -1;
85 }

```

## 5.2 Medium subtasks (subtasks 3 und 4): $S = 0$

```

1 // Library solution for S=0 in  $O(N^2)$ 
2 // Score: 35
3 #include "library.h"
4 #include <cstdlib>
5 #include <vector>
6
7 using namespace std;
8
9 long long int booksort(int N, int S, vector<int> order) {
10     long long int result = 0;
11     vector<bool> covered(N-1, false);
12     for(int i=0; i<N; i++) {
13         result += abs(i-order[i]); // Compute d(pi)
14         // Non-optimal linear sweep over the arc [i,order[i]]
15         for(int j=min(i,order[i]); j<max(i,order[i]); j++) {
16             covered[j] = true;
17         }
18     }
19     int counter = 0;
20     for(int i=0; i<N-1; i++) {
21         if(!covered[i]) {
22             // [i,i+1] is in E' if some uncovered edge follows
23             counter++;
24         } else {
25             result += 2*counter;
26             counter = 0;
27         }
28     }
29     return result;

```

```

30 }

1 // Library solution for S=0 in O(N)
2 // Score: 50
3 #include "library.h"
4 #include <cstdlib>
5 #include <vector>
6
7 using namespace std;
8
9 long long int booksort(int N, int S, vector<int> order) {
10     long long int result = 0;
11     int right = 0;
12     vector<bool> covered(N-1, false);
13     for(int i=0; i<N; i++) {
14         result += abs(i-order[i]); // Compute d(pi)
15         right = max(right, order[i]);
16         if(i<N-1 && right>i) covered[i] = true;
17     }
18     int counter = 0;
19     for(int i=0; i<N-1; i++) {
20         if(!covered[i]) {
21             // [i,i+1] is in E' if some uncovered edge follows
22             counter++;
23         } else {
24             result += 2*counter;
25             counter = 0;
26         }
27     }
28     return result;
29 }

```

### 5.3 Hard subtasks (subtasks 5 und 6): $S \neq 0$

```

1 // Library solution for all S in O(N^2)
2 // Score: 70
3 #include "library.h"
4 #include <cstdlib>
5 #include <vector>
6 #include <map>
7 #include <iostream>
8
9 const int INFTY = 10000000;
10
11 using namespace std;
12 using VI = vector<int>;
13
14 // Compute extend(l,r), initially assuming that the only
15 // cycles not checked yet are C[l] and C[r].
16 void extend(int &l, int &r, VI &C, VI &L, VI &R) {
17     // [ll, rr] is always the current extension,
18     // while only [l,r] was already checked for
19     // further overlapping cycles.
20     int ll = l, rr = r;
21     ll = min(ll, min(L[C[l]], L[C[r]]));
22     rr = max(rr, max(R[C[l]], R[C[r]]));
23
24     while(ll<l || r<rr) {
25         if(ll<l) {
26             l--;
27             ll = min(ll, L[C[l]]);

```

```

28     rr = max(rr,R[C[l]]);
29 } else {
30     r++;
31     ll = min(ll,L[C[r]]);
32     rr = max(rr,R[C[r]]);
33 }
34 }
35 }
36
37 // Compute the remaining cost of non-essentially connecting all the cycles
38 // if we already connected from S to [l,r].
39 long long int connect(int l, int r, VI &C, VI &L, VI &R, map<pair<int,int>,int> &Memo) {
40     extend(l,r,C,L,R);
41     // Memoization lookup: Did we compute it already?
42     if(Memo.find({l,r}) != Memo.end()) {
43         return Memo[{l,r}];
44     }
45     // If we have to do something, try going one step to the left or to the right
46     // and then take the cheaper of the two.
47     int nl, nr; // New interval [nl, nr]
48     long long int res = INFY;
49     if(l>0) { // Extend to the left.
50         nl = l-1; nr = r;
51         extend(nl, nr, C, L, R);
52         res = 2+connect(nl, nr, C, L, R, Memo);
53     }
54     if(r<C.size()-1) {
55         nl = l; nr = r+1;
56         extend(nl, nr, C, L, R);
57         res = min(res,2+connect(nl, nr, C, L, R, Memo));
58     }
59     Memo[{l,r}] = res;
60     return res;
61 }
62
63 long long int booksort(int N, int S, vector<int> order) {
64     long long int dP = 0;
65
66     // For every slot i, determine its cycle C[i].
67     // For every cycle c, determine its interval [L[c], R[c]].
68     vector<int> C(N, -1), L(N), R(N);
69     int l = S, r = S; // Compute the range that Mina needs to visit.
70     int c = 0; // Number of cycles of Pi.
71     for(int i=0; i<N; i++) {
72         dP += abs(i-order[i]); // Compute d(pi).
73         if(C[i] == -1) { // New cycle detected.
74             L[c] = i; R[c] = i; // Initialize its leftmost and rightmost slot
75             int j = i;
76             do { // Loop over the cycle.
77                 C[j] = c;
78                 R[c] = max(R[c], j);
79                 j = order[j];
80             } while (i != j);
81             if(i != order[i]) {
82                 // If the cycle is non-trivial, it needs to be part of the
83                 // range that Mina has to visit.
84                 l = min(l,L[c]);
85                 r = max(r,R[c]);
86             }
87             c++; // Finished processing the cycle containing slot i.
88         }

```

```

89 }
90
91 // Use dynamic programming to compute the cost of connecting all cycles.
92 map<pair<int,int>, int> Memo;
93 Memo[{l,r}] = 0; // If we reach the target we are done.
94 return dP+connect(S, S, C, L, R, Memo);
95 }

1 // Library solution for all S in O(N)
2 // Score: 100
3 #include "library.h"
4 #include <cstdlib>
5 #include <vector>
6 #include <map>
7 #include <iostream>
8 #include <cassert>
9
10 using namespace std;
11 using VI = vector<int>;
12
13 // Compute extend(l,r), initially assuming that the only
14 // cycles not checked yet are C[l] and C[r].
15 void extend(int &l, int &r, VI &C, VI &L, VI &R) {
16 // [ll, rr] is always the current extension,
17 // while only [l,r] was already checked for
18 // further overlapping cycles.
19 int ll = l, rr = r;
20 ll = min(ll, min(L[C[l]], L[C[r]]));
21 rr = max(rr, max(R[C[l]], R[C[r]]));
22
23 while(ll < l || r < rr) {
24     if(ll < l) {
25         l--;
26         ll = min(ll, L[C[l]]);
27         rr = max(rr, R[C[l]]);
28     } else {
29         r++;
30         ll = min(ll, L[C[r]]);
31         rr = max(rr, R[C[r]]);
32     }
33 }
34 }
35
36 // Compute the remaining cost of non-essentially connecting all the cycles
37 // if we already connected from S to [l,r] but need to go until we covered
38 // all of [target_l, target_r].
39 int connect(int l, int r, int target_l, int target_r, VI &C, VI &L, VI &R) {
40     int cost = 0;
41
42     // Repeat as long as [l,r] != [target_l, target_r]
43     do {
44         extend(l,r,C,L,R);
45         // Compute whether there is a next S-including cycle C_l to the left of l
46         // and its reaching cost c_l.
47         bool next_l = false; // Does C_l exist?
48         int cost_l = 0;
49         int l_l=l, r_l=r; // Temporary interval [l_l, r_l].
50         while(true) {
51             if(l_l <= target_l) break;
52             l_l--;
53             cost_l += 2;

```

```

54     extend(l_l , r_l , C, L, R);
55     if(r_l > r) { // Detect extension on the other side.
56         next_l = true;
57         break;
58     }
59 }
60 // Compute whether there is a next S-including cycle C_r to the right of r
61 // and its reaching cost c_r.
62 bool next_r = false; // Does C_r exist?
63 int cost_r = 0;
64 int l_r = l, r_r = r; // Temporary interval [l_r, r_r].
65 while(true) {
66     if(r_r >= target_r) break;
67     r_r++;
68     cost_r += 2;
69     extend(l_r , r_r , C, L, R);
70     if(l_r < l) { // Detect extension on the other side.
71         next_r = true;
72         break;
73     }
74 }
75 // Either there was an S-including cycle on both sides or on none.
76 assert(!(next_l ^ next_r));
77 if(next_l && next_r) { // We can extend on both sides.
78     cost += min(cost_l, cost_r); // Take the cheaper of both options.
79 } else {
80     // If there are no more S-including cycles, then we have to
81     // walk the necessary non-essential steps on both sides.
82     cost += cost_l + cost_r;
83 }
84 // New interval [l,r] = extend(C_l \cup C_r).
85 l = min(l_l, l_r);
86 r = max(r_l, r_r);
87 } while(target_l < l || r < target_r); // As long as Mina needs to explore more.
88 return cost;
89 }
90
91 long long int booksort(int N, int S, vector<int> order) {
92     long long int dP = 0;
93
94     // For every slot i, determine its cycle C[i].
95     // For every cycle c, determine its interval [L[c], R[c]].
96     vector<int> C(N, -1), L(N), R(N);
97     int l = S, r = S; // Compute the range that Mina needs to visit.
98     int c = 0; // Number of cycles of Pi.
99     for(int i=0; i<N; i++) {
100         dP += abs(i-order[i]); // Compute d(pi).
101         if(C[i] == -1) { // New cycle detected.
102             L[c] = i; R[c] = i; // Initialize its leftmost and rightmost slot
103             int j = i;
104             do { // Loop over the cycle.
105                 C[j] = c;
106                 R[c] = max(R[c], j);
107                 j = order[j];
108             } while (i != j);
109             if(i != order[i]) {
110                 // If the cycle is non-trivial, it needs to be part of the
111                 // range that Mina has to visit.
112                 l = min(l, L[c]);
113                 r = max(r, R[c]);
114             }

```

```
115     c++; // Finished processing the cycle containing slot i.
116   }
117 }
118 // Add up the number essential and non-essential steps needed.
119 return dP+connect(S, S, l, r, C, L, R);
120 }
```