Sorting + Greedy contest

March 9, 2019

Artem Vasilev

Hello Muscat Programming Bootcamp 2019

A. Inspection

Find the first and second minimum in the array of numbers.

A. Inspection

Find the first and second minimum in the array of numbers.

Do two passes of selection sort, don't care about the rest.

A. Inspection

Find the first and second minimum in the array of numbers.

Do two passes of selection sort, don't care about the rest.

Don't use any quadratic sorting algorithm, you will get TLE.

A. Inspection

Find the first and second minimum in the array of numbers.

Do two passes of selection sort, don't care about the rest.

Don't use any quadratic sorting algorithm, you will get TLE.

Can you figure out how to solve this problem with only one pass?

B. Sorting

Sort the given array with a quadratic sorting algorithm.

B. Sorting

Sort the given array with a quadratic sorting algorithm.

Just implement any of the sorting algorithms: bubble, insertion or selection.

C. Inversions

Find the number of *inversions* in the given array. Inversion is a pair of indices (i, j) where $i < j$ and $a_i > a_j$.

C. Inversions

Find the number of *inversions* in the given array. Inversion is a pair of indices (i, j) where $i < j$ and $a_i > a_j$.

Just implement two nested **for**-loops with an **if** condition.

C. Inversions

Find the number of *inversions* in the given array. Inversion is a pair of indices (i, j) where $i < j$ and $a_i > a_j$.

Just implement two nested **for**-loops with an **if** condition.

Number of inversions is a measure of how well the array is sorted. Stay tuned for a surprise reveal!

D. Selection sort

Implement the selection sort. Increase counter by 1 whenever the position of the minimum element among $i \dots n-1$ is not equal to i (in this case we actually swap elements at two distinct positions).

E. Insertion sort

Implement the insertion sort from lecture, increment counter whenever you swap two elements.

E. Insertion sort

Implement the insertion sort from lecture, increment counter whenever you swap two elements.

This problem and problem C are identical! Insertion sort always does a number of swaps equal to the number of inversions!

E. Insertion sort

Implement the insertion sort from lecture, increment counter whenever you swap two elements.

This problem and problem C are identical! Insertion sort always does a number of swaps equal to the number of inversions!

Now you see that insertion sort is effective when the number of insertions is low, which means that an array is “almost sorted”.

F. Archive creation

Apply greedy algorithm.

F. Archive creation

Apply greedy algorithm.

Sort all users by data. Then iterate over all sizes in sorted order and take items, until the total sum exceeds S.

F. Archive creation

Apply greedy algorithm.

Sort all users by data. Then iterate over all sizes in sorted order and take items, until the total sum exceeds S.

Why is it optimal?

F. Archive creation

Apply greedy algorithm.

Sort all users by data. Then iterate over all sizes in sorted order and take items, until the total sum exceeds S.

Why is it optimal?

Consider any better answer. We can transform this answer so it only contains a prefix in sorted order, which is exactly what we are doing.

G. Intricate sort

Create your own comparing function that first compares last digit $a\%10$, then compares numbers themselves.

G. Intricate sort

Create your own comparing function that first compares last digit $a\%10$, then compares numbers themselves.

Or create a vector of pairs $(a\%10, a)$, sort them, output second elements.

H. Digital root

Again, you can create a vector of pairs and sort them.

H. Digital root

Again, you can create a vector of pairs and sort them.

Fun fact $d(x)$, the digital root of x is almost the same as $x \bmod 9$.

I. Union of segments

We want to merge all overlapping segments into a one large segment.

I. Union of segments

We want to merge all overlapping segments into a one large segment.

Sort all segments by their left endpoint. Iterate over all segments and keep track of the current union.

I. Union of segments

We want to merge all overlapping segments into a one large segment.

Sort all segments by their left endpoint. Iterate over all segments and keep track of the current union.

If new segment is entirely to the right, we add the old union to answer, and start a new union.

I. Union of segments

We want to merge all overlapping segments into a one large segment.

Sort all segments by their left endpoint. Iterate over all segments and keep track of the current union.

If new segment is entirely to the right, we add the old union to answer, and start a new union.

Otherwise, we extend the union to the right.

J. Minimal cover

Repeat the following steps until you reach M.

J. Minimal cover

Repeat the following steps until you reach M.

Look at all segments with left border $\leqslant 0$

J. Minimal cover

Repeat the following steps until you reach M.

Look at all segments with left border $\leqslant 0$

Take the segment with maximum r

J. Minimal cover

Repeat the following steps until you reach M.

Look at all segments with left border $\leqslant 0$

Take the segment with maximum r

Repeat, but now consider segments with left border $\leqslant r$, and so on.