2 Randomized vs Deterministic Select

In this question, we consider a rivalry almost as fierce as that of the Canadian and American women's hockey teams[1]: that between Randomized Quick Select (which finds the k^{th} smallest element using a random pivot) and Deterministic Select (the algorithm we discussed in class that chooses the median of the medians of the groups of 5 elements as pivot).

Most of the algorithms we discussed since the beginning of the term were written in pseudo-code, or using a sequence of English statements (sometimes both). While converting pseudo-code to code written in an actual programming language is not usually difficult, it forces us to consider all the details that have been abstracted away by the pseudo-code description. In this question, you will implement the functions `RandomizedQuickSelect` and `DeterministicSelect`. You will then compare the average running times of the two algorithms on a sequence of random tests.

We have provided two files that contain code for a number of helper functions (one in C++, one in Java), including a main method/function.

1. [10 points] Your task will be to implement `RandomizedQuickSelect` and `DeterministicSelect` (in only **one** programming language), and then run some tests. As a reference, my `RandomizedQuickSelect` function is 31 lines long (including plenty of comments), and my `DeterministicSelect` function is 44 lines long (once again, including comments).

   IMPORTANT NOTE: the code we provided assumes that the functions `RandomizedQuickSelect` and `DeterministicSelect` return the **position** of the k^{th} smallest element in the array/vector, instead of its value.

   If you prefer to use a different programming language (Racket, Python, Haskell, Y86 Assembly Language (!), Prolog, ...), you are allowed to do so, although you will need to translate the parts of the code I provided into that language as well.

   Your `runtests` method/function should read in two parameters `size` and `attempts`, generate `attempts` random arrays of length `size`, and for each array find the median using both of the algorithms you need to implement. It should then print

   - the average number of comparisons made over all attempts, and
   - the worst-case number of comparisons made over all attempts.

   Run the program for roughly 10 different values of `size` that range from 10 to 100000, and put your results in a table.

   You should submit:

   - All of your source code.
   - The table containing your results.

   We recommend including the table in your LaTeX-generated PDF document, printing your source code to a PDF document, and then joining the two documents into a single PDF file for submission to Gradescope.

   Important: all code submitted must be your own. Code borrowed from somewhere else will not be considered academic misconduct if it is attributed properly, but neither will it be worth any marks in this case.

[1] The gold medal game is starting as I type this.