You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. Adhere to the Code of Academic Integrity. For a group, “you” below refers to “your group.” You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student’s code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on your own, seek help from the course staff.

Objectives

Completing this project will solidify your understanding of object-oriented programming and give you practice on developing and testing code incrementally—one class (or even one method) at a time. In Part B you will work with recursion.

Part A (question 1) appears in a separate document.

2 The Median

MATLAB has a median function that computes the median, also known as the 50th-percentile value. For example

```
>> x = [20 50 10 70 30];
>> med = median(x);
med =
    30
```

The median is the “middle value” when the data is sorted, so when the length of x is odd, \( \text{med} = \text{median}(x) \) is equivalent to

\[
y = \text{sort}(x); \quad m = \text{ceil}(\text{length}(x)/2); \quad \text{med} = y(m);
\]

If the length of x is even, then the median is the average of the two values in the middle:

\[
y = \text{sort}(x); \quad m = \text{length}(x)/2; \quad \text{med} = ( y(m) + y(m+1) )/2;
\]

However, sorting is a time-intensive procedure. Can we compute the median in a way that is more time-efficient than the “sort first” approach above? Our intuition tells us that we would be wasting our time by sorting the entire array: we just need the middle value and really do not care how the other values are ordered.

2.1 Divide-and-conquer

The key to a fast median-finder is the development of a recursive function using a divide-and-conquer approach. You will implement the following function:

```
function s = selectR(x,k)
% s is the kth smallest value in vector x
% selected by using a RANDOM splitter, using RECURSION.
% x is a vector of real values.
% k is an integer in [1..n] where n is the length of x.
```

Note the relevance of \texttt{selectR} to the median problem. For example, if \texttt{length(x)} is odd, then

\[
s = \text{selectR}(x, \text{ceil}(\text{length}(x)/2))
\]

assigns to \( s \) the median of \( x \). (The even-length case requires a follow-up calculation which you will deal with later.)

For simplicity, we start by assuming that the values in vector \( x \) are distinct, i.e., all the values in \( x \) are different. We will remove this assumption later. Let \( i \) be an integer whose value satisfies \( 1 \leq i \leq n \) where \( n \) is the length of vector \( x \). Index \( i \) defines a splitter: we can separate out the values smaller than the splitter from the values greater than the splitter. Let’s use a length 15 example:
Let $i=13$, meaning that $x(13)$ is the splitter. (The 13th component is marked above with an arrow; its value is 57.) We collect the values smaller than the splitter into a vector littleVals and the values greater than the splitter into bigVals:

$\text{littleVals} = \begin{bmatrix} 22 & 37 & 10 & 34 & 30 & 16 & 41 & 36 & 50 \end{bmatrix}$ length 9 vector

$\text{splitter} = 57$ scalar

$\text{bigVals} = \begin{bmatrix} 89 & 66 & 75 & 83 & 97 \end{bmatrix}$ length 5 vector

Our function selectR wants the $k$-th smallest value in $x$ where $1 \leq k \leq n$. It follows that

$$\begin{align*}
\begin{cases}
  k \leq 9 & \text{then } s = \text{selectR}(\text{littleVals}, k) \\
  k = 10 & \text{then } s = x(i) \\
  k \geq 11 & \text{then } s = \text{selectR}(\text{bigVals}, k-10)
\end{cases}
\end{align*}$$

assigns to $s$ the $k$-th smallest value in $x$.

In other words, once we have littleVals and bigVals, the selectR($x$, $k$) calculation can be reduced to a “shorter” selectR computation. This is a classical divide-and-conquer situation. Either we “luck out” and stumble upon the $k$-th smallest value ($s = x(i)$) or we apply selectR to a shorter vector (either littleVals or bigVals). Thus, we can structure the body of the function selectR as follows:

Consider an index $i$ whose value satisfies $1 \leq i \leq n$ where $n$ is the length of $x$. (Details to follow.)

Compute the vectors littleVals and bigVals based on the splitter $x(i)$.

Based on length(littleVals), figure out which of these situations applies:

1. $x$’s $k$-th smallest value is the splitter. Assign $x(i)$ to $s$ and return—no recursion.
2. $x$’s $k$-th smallest value is in littleVals. Recursively call selectR to find it.
3. $x$’s $k$-th smallest value is in bigVals. Recursively call selectR to find it.

Notice that the input vectors for the recursive calls are shorter than $x$. It follows that the recursion must end. Either we luckily land on the $k$-th value if length($x$) > 1 or we have to land on the $k$-th value because length($x$) is 1.

Before we discuss how to choose index $i$ for the splitter, we will remove the assumption that $x$ has distinct values. Here is what we will do: put into bigVals the components that are greater than or equal to the splitter but exclude the splitter itself. Let’s modify the above example so that there are duplicates in $x$ and one of the duplicates happens to be chosen as the splitter:

$\downarrow$

Let $i=13$ again, i.e., the 13th component is the splitter. This time there’s another component (the 7th) that has the same value (57) as the splitter. Then we have

$\text{littleVals} = \begin{bmatrix} 22 & 37 & 10 & 34 & 30 & 16 & 41 & 36 & 50 \end{bmatrix}$ length 9 vector

$\text{splitter} = 57$ scalar

$\text{bigVals} = \begin{bmatrix} 89 & 66 & 57 & 83 & 97 \end{bmatrix}$ length 5 vector

The 13th component of $x$ is the splitter—it must be a scalar. The 7th component of $x$, which happens to have the same value as the splitter, is in bigVals. Be careful in writing your code: the splitter—the $i$th component of $x$—should not be in bigVals while any other components that have the same value as the splitter should be in bigVals.

Finally we turn to choosing index $i$ for the splitter. Clearly we would like the value of the splitter to be near the median value. This would ensure that the input vector for the recursive call is roughly half the length of $x$. Rapid progress to the $n = 1$ base case would ensue. Unfortunately, the efficiency of the overall search-for-the-median process would be degraded if too much effort is spent looking for a “good” splitter. Therefore we opt for what appears to be a careless but super-cheap (effort-wise) strategy:
Let $i$ be a randomly generated integer, equally likely to be any index of the vector $x$.

It is not at all clear that this is a wise choice. Only through implementation and experimentation can we find out.

After you implement `selectR` as specified above, make effective use of it in order to implement `myMedian`:

```matlab
function med = myMedian(x)
    % med is the median of real-valued vector x.
    % Make effective use of selectR. Do not use built-in functions median or sort.
```

### 2.2 Random splitter vs. middle splitter

Implement the following function:

```matlab
function s = selectM(x,k)
    % s is the kth smallest value in vector x
    % selected by using a MIDDLE splitter, using RECURSION.
    % If x is even-length, define "middle" as length(x)/2.
    % x is a vector of real values.
    % k is an integer in [1..n] where n is the length of x.
```

Just a few changes of your function `selectR` will give you `selectM` since all that is different is that the index $i$ that defines the splitter is now the middle index of $x$. Be sure to change the recursive calls to use `selectM` as well and not `selectR`.

Next we do some experiments to see if one splitter is better than other. In Lectures 26 and 27 we compare algorithms by counting the number of operations required. Here we use a different approach; we will *time* the execution of our functions using the built-in timing functions `tic` and `toc`.

Download the script `compareSelect` from the course website and execute it to see how the random splitter and middle splitter compare. Read `compareSelect` to make sure that you understand what it does—you should be able to write all that code (but we give it to you in order to keep the length of the project reasonable).

The given code uses as test cases only three vector lengths: $10 + 1$, $10^2 + 1$, $10^3 + 1$. Modify the code to include also the length $10^4 + 1$. (Are you daring . . . and patient? You can try even longer vectors if you want.)

Observe the use of the `plot` function to draw the graphs. MATLAB has functions for graphing on logscale axes:

- `semilogx` for using the log scale on the x-axis
- `semilogy` for using the log scale on the y-axis
- `loglog` for using the log scale on both axes

Try them out! All you need to do is to replace `plot` with the above functions one at a time; the arguments remain the same. Keep the function that you think is best for displaying the results.

At the end of the file add a comment to answer the following questions:

1. Is one splitter choice better than the other? Why do think that is the case?
2. Which function do you think is best for displaying the results? Why?

There is no need write an essay. Two or three sentences suffice for each question.

We end this project by reminding you of an important theme in the course: we compute because we want to gain *insight* into a problem, not because we want just one answer. When we write code we should think about facilitating *analyses*. Take `compareSelect` for example. Writing the code cleanly, using appropriate data structures (e.g., cell vs. simple arrays), using named parameters, and including concise comments facilitate our experimentation. Visual representation of the computational experiments—make good use of the graphics functionalities in MATLAB—further supports computational exploration. Remember these ideas and tools!

Submit your files `selectR.m`, `myMedian.m`, `selectM.m`, and your modified file `compareSelect.m` on CMS.