This lab concerns the problem of finding the median. We know that we can use quick select to find the median (or any other order statistic). The problem is that whereas the average complexity of quick select

\[ C_{\text{ave}}(n) = \Theta(n) \]

which is excellent, in some situations, poor choices could be made for the pivot element, and then

\[ C_{\text{worst}}(n) = \Theta(n^2) \]

which may be unacceptable. This is even the case where we enhance the choice of the pivot point using the median of a sample of size three chosen from the array.

A useful solution to this problem would be to develop an algorithm with smaller complexity for finding an approximate median. This would be useful in any situation where a pivot point near the median is desirable for best performance, for example, with both quick select and quick sort.

1. Get the program `med_of_med.cpp` and compile and link it. Get a variety of sample datafiles which you have made in previous labs with a range of entropies. Read the introduction in the code `med_of_med.cpp` which explains the strategy and look over the procedures `median5()` and `median_of_medians()`. Note that `median5()` has constant time complexity. Test the program thoroughly to see
   i. how many basic operations (swaps, comparisons, etc.) are required by the standard quick sort and how many are required by median of medians?
   ii. how “good” are the approximate medians which are produced in terms of %-ile?

2. The claim made for the median of medians algorithm is that the approximate median will always be between the 30-th and 70-th %-iles, that is, it will always be greater than or equal to 30% of the original numbers and greater than no more than 70% of the original numbers. Is this the performance you are getting?

3. Note that the complexity of the median of medians algorithm satisfies the inequality

\[ C_{\text{median of medians}}(n) \leq C_{\text{quick select}}(n/5) + \Theta(n) \]

Why is this true?

**Extra Credit Assignment:** Adapt the median of medians algorithm to produce a new version of quick select which will be guaranteed to have a complexity

\[ C_{\text{worst}}(n) = \Theta(n) \]

and implement your program in C or C++. I want both the program and a short writeup explaining why the worst case will be no worse than linear time. You may incorporate any code that I have provided in sample programs in your program.