**Writeup**: For this programming task, test your implementation by invoking it for bit vectors of various sizes, and plotting the bit-vector size (say N) versus the time requried to do some fixed number of rank operations. Also, plot the bit-vector size (say N) versus the result of calling the overhead() function. Does your implementation match the expected theoretical bounds?

**Writeup**: For this programming task, test your implementation by invoking it for bit vectors of various sizes, and plotting the bit-vector size (say N) versus the time requried to do some fixed number of select operations. Also, plot the bit-vector size (say N) versus the result of calling the overhead() function. Does your implementation match the expected theoretical bounds? *If you feel ambitious*, you can additionally implement a constant-time bit-vector select, though this is not required.

**Writeup**: For this programming task, test your implementation by generating sparse arrays of a few different lengths (e.g. 1000, 10000, 100000, 1000000) and having various sparsity (e.g. 1%, 5%, 10%). How does the speed of the different functions vary as a factor of the overall size? How about as a function of the overall sparsity? Finally, try and estimate how the size of your sparse array in memory compares to what the size would be if all of the 0 elements were instead explicitly stored as “empty” elements (e.g. as empty strings). How much space do you save? How do your savings depend on sparsity?