Assembly:
My workflow was as follows. First, I considered how to best approach decontamination of the input reads. Since I noticed the question dictated that we discard reads with contamination detected, I knew firstly that extension would not play a role. So, the real question was whether scanning the ends of a read for contamination should be based on exact matches with vector kmers or a scoring matrix and threshold value. I ended up choosing exact matches because I reasoned that exact matches decreases the number of reads which end up being marked as contaminated, which compensates for the large amount of data loss per contaminated read since we discard the entire read upon each contamination. This contamination function takes in a parameter k, which I ended up using 10 as my default value for because it optimizes the length of my final sequenced assembly compared to other values I tried (4, 5, 15, 20, 40, etc). Specifically, out of a total number of 1274 input reads, contamination using k=10 resulted in 1224 input reads for the next stage of assembly, which is a pretty reasonable loss of data compared to some of the other values of k.

Next, for correction, the main concern was whether to include the kmers of the vector as part of the frequency counting and/or replacement process based on just the input reads. I ultimately decided against using the kmers of the vector in this counting process because I felt that kmers in the vector were not relevant to the input reads, and that I would rather like replacements to be from actual sequences in the reads rather than just biological sequences which make sense but are not present in the specific sequence we are considering. This function has parameter k, t, d, which I ended up optimizing to be k=15, t=2, d=2. This k and t was chosen based off the optimal k we discovered in pr2, and d was chosen using trial and error by testing around values (3, 4, 5, 6, 10, 20). This ultimately resulted in a final sequence length of 125 (AAAACCCGCAATCCTGCTAACAATGCTGCAATCGTGCTACAACTTCCTCAAGGAACAACATTTGATGAGAGGCGGCAGTCAAGCCTCTTCTCGTTCCTCATCACGTAGTCGCAACAGTTCAAGAT). A strength of my approach was that it is very fast, and running the full assembly workflow using the given input reads takes around 20 seconds. A downside of my approach is that it may not be optimized for accuracy since my correction code was designed in pr2 to prioritize computational time over accuracy, since it achieved near-perfection (but not perfection) corrections in a much faster time than algorithms focusing on perfect correction. This makes it hard to gain a good picture of the actual sequence we are trying to assemble.