Assignment 3

William Wallace
1216661.

Q1] Let ~~the~~ ONES-HALT = {R(M) | M halts exactly on input of form 1…1}

Notice the resemblance between ONES-HALT and HALT, where an instance of HALT is really just a pair (M, w) consisting of a TM M and an input word w.
Since w is a finite sequence of letters, it's easy to construct a new TM, M' which, on a tape consisting of only 1's, does the following: first it writes w on the tape, then it checks that w consists entirely of 1's, then it runs M.
M' will halt on only 1's if and only if M halts on w, so we have succeeded in reducing HALT to ONES-HALT, thus proving ONES-HALT to be undecidable ie. there is no algorithm to decide whether the TM will halt on that ~~input~~ form of input.

Q2] The algorithm for solving HALT IN BOUNDED SPACE requires a record of the state, read/write head position, and the symbols on each cell of the tape. We can then go about recording the combinations of each of these variables in order to know whether we can halt on input w without ever moving read/write head beyond the final character of w. Record each computation by returning "not halt" if the tape has not revisited the combination. ~~If the tape~~ We can be certain that we have entered a loop ~~#~~ or at least that entering a loop is possible, if we have revisited a combination after iterating over the tape, hence there IS an algorithm for solving HALT IN BOUNDED SPACE.

Q3) We know that a $\mathcal{E}TM$ is in P if it runs in polynomially bounded time $O(n^k)$. We also know that the number of ways the bits in a binary word of length x can be arranged is 2^x.

So for a binary word of length x, which can be any number, we can assign x to be $k+1$, that is, x will always be greater than k. This is important since polynomial time is bounded by n^k while the binary on the LHS is not bounded at all.

This looks like: $2^x \# n^k$, $x = k+1$.

Put differently, it requires 2^x steps to check whether w is accepted, but only n^k steps are possible, since $1 \cdots 1$ is 1^k, and $\sim$ is 1.

Consequently, this TM cannot have polynomially bounded running time.

Q4). Determining that a non-negative binary integer is equal to twice a perfect square is in P.

This problem requires going beyond a naive approach of using brute force to iterate over integers from 1 to $2 \times n^2$, and checking that $2n^2 \leq n$ ea until $2n^2 = n$, since it is not a polynomial time algorithm. This is because $\log n$ is the parameter, not n, and this comes with the fact that we are determining the square roots for binary numbers, not decimal numbers.

Instead, to increase efficiency, use a search and test step:

- Find a number n when added to the RNS of a current solution s, such that $(s+n) \cdot (s+n) \leq x$, where x is the value for which a root of the original perfect square is divided by 2.

When expanded, $s \cdot s$ in the expression $s \cdot s + 2sn + n \cdot n \leq x$, which is usually the remainder, can be updated incrementally & efficiently when working in binary since the value of n will have a power of 2 as the operations to compute $4 \cdot sn$ and $n \cdot n$ have increased efficiency with faster bit-shift operations.

Q5) Prove HALF SAT is NP-complete.

Let f be an instance of SAT with variables $x_1, \ldots, x_n$. For each clause C in f, f contains C and a new clause C' where C' is obtained from C by replacing all literals x_i with y_i.

For example, if $C = (x_1 \vee \neg x_2)$, then $C' = (\neg y_1 \vee y_2)$.

If we can prove that f is satisfied by some assignment if and only if f has a satisfying assignment in which exactly half the variables are true, then SAT, which is NP-complete, will have been reduced to HALF-SAT, which proves the latter is NP complete.

Assume that f has the required assignment. If this is the case, then the same assignment restricted to the variables $x_1, \ldots, x_n$ is a satisfying assignment for f' (since every clause of f is also in f').

Assume now, that f has a satisfying assignment. Extend this by letting y_i be TRUE if x_i is FALSE. (and vice-versa). This can be easily checked that this is a satisfying assignment for f and that exactly half the variables are true, so HALF-SAT is NP-complete.