CS5230.
Tutorial 1: Answer sketches

Q1. Various ways to implement sorting. Easiest mechanism would be to represent input numbers in binary, with a special symbol $*$ separating them. Then, one can do bubble sort.

This process takes about $O(k^2 * n)$ time, where n is the length of the whole input (note that this is sum of the length of all the numbers plus the number of numbers) and k is the number of numbers.

Q2 (sktech). Done in class, by representing each (one sided infinite) tape using two tracks: one to give the content of the tape, and the other to give the head location.

One can simulate each step by (i) first finding the location/content of each tape by searching for the head locations, then (ii) determining what needs to be written at these positions/head movement/new state, and then (iii) implementing these by updating the content of the tapes (at the head locations) and then moving the head symbols to left/right for each of the heads and changing the state.

Simulating each step in the above process takes time about $O(s)$, where s is the furthest any head is from the starting position. As s is bounded by t the number of steps of the original TM, the total time needed is bounded by $O(t^2)$.

Q3 (a) No. For example, finding if there are odd number of 1's in the input can be decided in $T(n)$ time, but not in $T(n/2)$ time.

(b) Yes, using the space compression theorem done in class.

Q4 (sketch). In class we showed that for any recursive function h there are computable functions which are not $T(h(n))$ computable. As 2^n and $2^{2^{2^n}}$ are recursive functions, we have the result.

For natural examples, decidability of truths of given statement in Presburger arithmetic takes doubly exponential time.

Q5 (sketch).

Easier way to do this is to use Queston 6. However, below is sketch of idea to do it directly.

(a.i) For n^2, construct a machine which goes through the input tape exactly n times. In the first round, copy the input to the second tape. Then, use the length of written part of second tape as a counter, to go through the input $n-1$ more times.

(a.ii) For 2^n do as follows (using 3 tapes):

First move right one cell on the input tape. Then do $n-1$ rounds as follows (where $n-1$ rounds are counted using the remaining $n-1$ letters of the input tape).

In first round write 1 on the 2nd tape. Then execute the following loop for a total of $n-2$ rounds (note that 2 rounds are executed in each iteration of the loop).

Loop:

In next round double the number of 1s in 2nd tape and write this on the third tape (while erasing the content of the 2nd tape).

In next round double the number of 1s in 3rd tape and write this on the second tape (while erasing the content of the 3rd tape).

End Loop

Total time used by above process is $1 + 1 + 2 + 4 + 2^3 + \ldots + 2^{n-1} = 2^n$.

(a.iii) For $n\lceil \log n \rceil$, the best way to do it is using Q6. Alternatively, we could first copy the input on second tape, and then calculate the length of the input in binary using the second and third tapes. The length of this binary number is the number of times we need to go through the input tape. Detailed omitted.

(b) For n, just copy the input tape into second tape. For $\lceil \log n \rceil$, count the number of cells used in the input tape (in binary, using the work tape). For $n\lceil \log n \rceil$, first write n and then $\lceil \log n \rceil$, in binary, in two work tapes. Then compute their product, and convert to unary in a new work tape.

Q6 (sketch): Assume $T(n)$ is super linear (that is $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$).

Suppose M computes $T(n)$ in binary within time $T(n)$. Using linear speed up theorem, construct a Turing Machine M' which computes $T(n)$ in base 2^m, within time $T(n)/10$, for some large enough m ($m = 1000$ should work).

Here is the machine which witnesses that $T(n)$ is fully time constructible. On any input of length n, simulate M'.

(a) Let the number $T(n)$ computed above be called b.

This process takes time at most $T(n)/10$

(b) Let $q = \lfloor \frac{b}{2^m} \rfloor$ and $r = b \bmod 2^m$. Note that one can compute q by just dropping the least significant "bit" of b, and r is the least significant "bit" of b.

This process takes time at most $O(\log T(n))$.

(c) Convert q into unary (in a separate tape). Note that this takes time bounded by $c * q$, for some constant c ($c = 4$ is enough).

This process takes time at most $\frac{4T(n)}{2^m}$.

(d) While the above process ((a) to (c)) is ongoing, record time taken by the above computation, in a separate tape. This is done in unary like system, where each position records a value upto $2^m - 1$. Thus, the time taken is recorded in s cells, where $s - 1$ cells have value $2^m - 1$, and the s-th cell has value between 0 and $2^m - 1$. Let the value in the s-th cell be t_s. Thus, total time taken upto now is $(s - 1) * (2^m - 1) + t_s$.

(e) Note that the total time that we need to spend is $q * 2^m + r$. Thus, the remaining time needed to be spent is $q * 2^m + r - (s - 1) * (2^m - 1) - t_s$. Note here that $q \geq s$, for large enough n. Thus, the time we need to waste is:

$(q - s) * 2^m + s - 1 + 2^m - t_s + r$.

which can be easily done by first wasting time r, then wasting time $2^m - t_s$, then wasting time $s-1$ (which can be obtained by moving the head through the cells used to record the time already spent as in part (d) above). Then, one can waste time $(q - s) * 2^m$.