1. (10 points) Experiment with stack pruning parameters. What is their affect on...

The purpose of this warm-up exercise is to experiment with tradeoffs in time and search error, and to actually look at some data—looking at data is important!

a) By increasing -s and -k up to around 20, it's possible to increase logprob from -1439 to -1353. After that it's not possible to improve. *A plot of different operating points is welcome here.*
b) Translating with defaults takes 0m1.518s user time, at s=k=20 takes 0m4.115s. *Times may vary by system; the point is to notice it gets slower!*
c) *This question is free-form. There's no right or wrong answer,* ***except*** *that it must include examples, as stated in the question. So I might say:* The output of the decoder using default beam sizes is less fluent than output when beam size is increased. For example, the former produces "honourable senators , I do name person", while the latter produces "honourable senators , I will name people"
d) -1353

2. (15 points) Define a new dynamic program for Part 2.

The purpose of this exercise is to think about the problem mathematically before implementing it; this is an especially critical skill for machine learning implementations. We are lenient on notation ***if*** *we understand it. The focus here is on getting the essentials of the recursion correct (i.e. the terms involving h and s). The distinction between tropical and log semirings isn't too important, so I've used* $\otimes$ *below to abstract over the corresponding operation. (The decoder works in the log semiring to avoid underflow).*
A bit of simplifying notation: Let $LM(w)$ be the language model log probability for a sequence of words w, and $TM(v|w)$ be the translation model log probability for translating string w into string v. Then we can describe the dynamic program using these recursions, where $h(i, e)$ is the probability of the best translation of the first i words, ending in word e, and $s(k, j, i, e)$ is the probability of the best translation of the first i, except those between positions k through j:

$$h(i,e) = \max \left(\begin{array}{l} \max\limits_{j,e',w} h(j,e') \otimes TM(f_{j+1}...f_i|we) \otimes LM(e'we), \\ \max\limits_{k,j,e',w} s(k,j,i,e') \otimes TM(f_{k+1}...f_j|we) \otimes LM(e'we) \end{array} \right)$$

$$s(k,j,i,e) = \max_{k-1,e',w} (h(k,e') \otimes TM(f_{j+1}...f_i|we) \otimes LM(e'we))$$

It may be tempting to define the following dynamic program which does not use the s items [*but since the question explicitly asks for s items, -5 points for this solution*]:

$$h(i,e) = \max \left(\begin{array}{l} \max\limits_{j,e',w} h(j,e') \otimes TM(f_{j+1}...f_i|we) \otimes LM(e'we), \\ \max\limits_{k,j,e',w} h(j,e') \otimes TM(f_{k+1}...f_i|w') \otimes TM(f_{j+1}...f_k|we) \otimes LM(e'w'we) \end{array} \right)$$

This works by translating the swap case all-in-one. This will work but it has two problems: it shares less work in the swap cases (so will be somewhat slower); and it's impossible to extend to part 3.

3. (5 points) What is the complexity of your Part 2 decoder? Explain your reasoning.

The $s(k, j, i, e)$ terms determine complexity: since i, j, k range over sentence positions, complexity is $\mathcal{O}(I^3)$ (where I is the sentence length). If we make the mild assumption that phrases have a maximum length K, this reduces to $\mathcal{O}(IK^2)$, since given a choice of i, there are at most K ways to choose j; and given a choice of j, there are at most K ways to choose k. *Since K is constant, $\mathcal{O}(I)$ is an ok answer.*

4. (5 points) What is the mapping from hypothesis objects to stacks for Part 2?

Let S be a function mapping hypotheses to a stack. Then $S(h(i, e)) = i$ and $S(s(k, j, i, e)) = i - (j - k)$.

6. (15 points) Experiment with stack pruning parameters for Part 2. What is their affect on...

a) With default params, my log prob is -1649, worse than the default decoder—this is because the new decoder searches over a larger search space, and makes worse search errors with harsh pruning heuristics. However, log prob increases to -1300 with s=60 and k=20. No further improvement is possible after this point.

b) Translating with default parameters takes 0m1.482s, similar to the same settings for the default decoder. Translating with s=60 and k=20 takes 0m13.282s, demonstrating a computational cost of finding higher probability translations, relative to the default decoder.
c) *Again, there's no prescribed answer here, but the answer must give examples.* In general, default settings are less fluent than best settings as before, e.g. "i think out of the . replacing" vs. "i do not believe in the alternative ." Notice that even the default has reordering, involving the full stop. On the other hand, the best output is not always obviously better than the monotone decoder. In contrast with answers to 1c), the best decoder now produces "honourable senators name , I do not know . "
d) -1300

Your numbers may be slightly different from the above, but not by much.

7. (10 points) Define a new dynamic program for Part 3.

This is a small change to the part 2 decoder. In cases where a phrase has been skipped, we can still append to the right, as in the new final line of the recursion (the rest is as in part 2):

$$h(i,e) = \max \left(\begin{array}{l} \max_{j,e',w} h(j,e') \otimes TM(f_{j+1}...f_i|we) \otimes LM(e'we), \\ \max_{k,j,e',w} s(k,j,i,e') \otimes TM(f_{k+1}...f_j|we) \otimes LM(e'we) \end{array} \right)$$

$$s(k,j,i,e) = \max \left(\begin{array}{l} \max_{k-1,e',w} (h(k,e') \otimes TM(f_{j+1}...f_i|we) \otimes LM(e'we)) \\ \max_{k,j,e',w} s(k,j,\ell,e') \otimes TM(f_{\ell+1}...f_i|we) \otimes LM(e'we) \end{array} \right)$$

8. (5 points) What is the computational complexity of your Part 3 decoder?

Look at the final line of the recursion, which involves four variables over sentence positions: k, j, ℓ, and i. With no bound on phrase length, this is $\mathcal{O}(I^4)$. With a bound of K, ℓ must be within K positions of i and k must be within K positions of j, but i and j can be arbitrarily far apart, so we get $\mathcal{O}(I^2K^2)$.
I will take an alternative answer here, if explicit reasoning is given: assuming a constant bound on both stack size and phrase length, every item in each stack has a constant bound on the number of possible extensions, giving us $\mathcal{O}(I)$ for the stack decoding algorithm—notice this is much smaller than the size of the actual dynamic programming search space, hence pruning is much more severe here.

9. (5 points) What is the mapping from hypothesis objects to stacks for Part 3?

This doesn't change. It's the same as for the part 2 decoder.

11. (5 points) What is the maximum log-probability your Part 3 decoder can obtain? What do you conclude?

By making the beam size very large, it's possible to substantially improve over the part 2 decoder; For example s=1000, k=100 yields a log prob of -1279. Slight improvements are still possible with larger s, but the log probability essentially plateaus near this value. Moreover, generating these high-probability translations is very expensive, taking 8m2.574s. It's clear that there's a tradeoff in search space expressivity, maximum log probability, and speed. It's not clear that the high-probability translations are better, however. For example, the part 2 decoder gives "that it was struck by the parliamentary associations in 1993 .", while the part 3 decoder gives "it was struck by the parliamentary associations that in 1993 ."—the word "that" has migrated across the sentence, forming a locally fluent sequence of n-grams, but this is not where the word should be! *Your numbers may be slightly different from these, but not by much.*