A) Regarding the 3b sentences, I don’t know if I’m straying too far from the overall material by giving this answer, but I’d honestly argue that center-branching structures are in essence right-branching structures. “The actor the boy met won” is arguably a case of that-ellipsis where the phrase can be understood as “The actor that the boy met won”. Because of this, when I did the top-down parse for 3b to fill the missing value in the table, I honestly didn’t notice much of a difference in terms of computational difficulty. In terms of computational difficulty for the LC-parser, I don’t think I noticed an increase in difficulty when doing the tables for 1-3b using left-corner parsing.

For 1c, I’d argue there isn’t really any configuration that has noticeably more computational difficulty for bottom-up and LC parsing. Bottom-up parsers can just reduce the configuration at (NP POSS N, ‘s baby won) to (NP, ‘s baby won) and parse another possessive structure without increasing memory loads. LC parsing can simply LC-PREDICT a possessive from (NP, ‘s boss’s baby won), and once that’s done, LC parsing can just LC-predict the second possessive branch further on. In essence, there’s essentially no configuration differences here between 1b and 1c for these two parsers. However, for top-down parsing, predicting completely is significantly more expensive because it needs to predict that the innermost possessive’s NP is a series of nested possessives, leading to the expensive prediction of (NP POSS N POSS N VP, Mary’s boss’s baby won).

For 2c, bottom-up parsing would have to parse an entire right-branching structure to fully be able to reduce it. As such, before being able to reduce using S → NP VP, this configuration from 2c would be particularly expensive due to having to use shift transitions for the entire structure:

(NP V D N THAT V D N THAT V D N, epsilon). Everything after the first V needs to reduced into a single VP, but that entire configuration can’t be reduced until the end of the sentence because of the embedded that-clauses. The VP within a that-clause needs to be fully parsed before the D N SRC rule can be applied, but the embedded VPs also have NP → D N SRCs that need to be fully parsed first before being able to reduce them. 2c doesn’t pose a problem for top-down and LC-parsing because the use of predicts (LC-predicts) helps shave off the already-parsed Ds and Ns for NP → D N SRC while predicting for SRC → THAT VP helps shave off the memory load of the that’s.

For 3c, I straight up would like to refuse to offer a judgement here because 3c simply isn’t possible in generative syntax. No parser would be able to parse 3c properly here.

B) Arc-standard parsing poses a minor memory usage increase. Mostly, it involves a few extra steps for canceling, but LC-predict still helps keep down memory usage overall, making almost no difference in memory usage, if at all.

i)

0) --- --- (S-bar, 0)

1) SHIFT D → the (D S-bar, 1)

2) SHIFT N → actor (D N S-bar, 2)

3) LC-PREDICT NP → D N ORC (ORC-bar NP S-bar, 2)

4) SHIFT D → the (D ORC-bar NP S-bar, 3)

5) LC-PREDICT NP → D N (N-bar NP ORC-bar NP S-bar, 3)

6) MATCH N → boy (NP ORC-bar NP S-bar, 4)

7) LC-PREDICT ORC → NP V (V-bar ORC ORC-bar NP S-bar, 4)

8) MATCH V → met (ORC ORC-bar NP S-bar, 5)

9) CANCEL --- (NP S-bar, 5)

10) LC-PREDICT S → NP VP (VP-bar S S-bar, 5)

11) LC-PREDICT VP → V (V-bar VP VP-bar S S-bar, 5)

12) MATCH V → won (VP VP-bar S S-bar, 6)

13) CANCEL --- (S S-bar, 6)

14) CANCEL --- (6, 6)

ii) 1b) (VP-bar S S-bar, won); note that multiple possible configs here have 3 max elements on the stack, so not much difference this time around out of coincidence.

2b) (VP-bar SRC SRC-bar, 5); there’s again a bunch of ties here, as far as I can see mostly everything goes to 3 terms on the left side of the configuration at most.

3b) (V-bar ORC ORC-bar NP S-bar, 4)

1c) (VP-bar S S-bar, won); note that because LC-parsing is good at reducing branching structures, there’s essentially no differences from 1b. Also again, lots of ties.

2c) (VP-bar SRC SRC-bar, 5); again, same reason as above. Also lots of ties again.

3c) Considering the aforementioned reasoning that ORC structures are essentially that-elipsis, constituting right-branching structures, 3c is completely ungrammatical. If forced to parse it with an LC-parser, I’d imagine that given LC-parsing is good at reducing structures to keep down memory usage, it’d be the same as 3b.