ENGR 3520: Foundations of Computer Science
Professor Lynn Andrea Stein
Fall 2010

Assignment 3

This assignment is due on Tuesday, 19 October. Please email your completed assignment to las **and** bring a **stapled** paper copy to class.

See also 2010Assignment3Faq

Sort algorithms

Consider the following list:

```
9 3 1 7 5 0 4 2 9 -2 8 6 10 89 7 3
```

A. Show how mergesort first divides the list into sublists (show each level of division) and then merges into sorted lists. You may wish to consult the descriptions of mergesort on Wikipedia or in Cormen or another algorithms book. Your answer should take the format of several lines of subdivided sorts followed by several lines of partially merged sorted lists followed by the final sorted list. For example, for the list

```
12 5 3 4
```

your answer would be

```
12 5 3 4
12 5 | 3 4
12 | 5 | 3 | 4
5 12 | 3 4
3 4 5 12
```

B. Repeat the above exercise using quicksort instead of mergesort. Again, you may consult Wikipedia, Cormen, or another resource. You should choose as your pivot the first element in each (sub-)list. You do not need to perform an in-place quicksort, though you may do so if you choose. (For in-place quicksort, there's a nice if painfully detailed animation at NYU.

C. Insert the elements, in the order given, into a heap. Show the heap after each insertion.

NB: A heap is like a partially ordered binary tree: the parent is larger than both children (this is called the "heap property"), and the children need not be ordered with respect to one another. A heap can be stored in an array using the children-at-2n+1,2n+2 trick. When a new element is inserted into the next free position, you need to re-heapify by comparing it with its parent to maintain the heap property (and so on all the way up to the root). For more details, look up **heapify** or **heapification** in Wikipedia, Cormen, or another resource (search for *heapsort*). I like the animation at Oneonta but it does not actually explain heapification: lavender nodes are insertions that don't need correction, while orange nodes have just been re-heapified.

Grammars

Consider the grammar over the terminal alphabet {a, b} with start symbol S (and e representing the empty string):

```
S -> a S b
S -> a b S
S -> e
```

A. Show that the grammar is ambiguous, i.e., find a string that has two different (leftmost) parses and give both parse trees for the string.

B. Find an equivalent unambiguous grammar.

C. Construct a PDA that recognizes this language.

Grammars in Prolog

Expand the grammar that we discussed in class (for English sentences) so that it handles:

A. adjectives (e.g., [ambidextrous, ophelia, pirouettes] and [rodriguez, chastizes, the, ambidextrous, whangdoodle])
B. prepositional phrases (e.g., [a, hippogriff, chastizes, the, whangdoodle, with, compassion] or [ophelia, pirouettes, with, the, hippogriff])
C. Extra Credit: Also handle adverbs (which can modify either adjectives or verbs).
D. Extra Credit/Challenge: Read about additional arguments to prolog grammars and modify the numeric matching grammar in grammarNM.pl (either as above or to handle person matching, pronoun case matching or -- presumably in a language other than English -- gender matching). in another language

You need not be able to generate all possible sentences with your grammar; it is sufficient for your grammar to *accept* sentences in this form. Remember that test parses have the form:

- `sentence( [ ambidextrous, ophelia, pirouettes ], [] ).`

while "generate-all-examples" queries have the form:

- `sentence( CapitalizedName, [] ).`

Also remember that you can consult i.e., read in) a file named filename.pl (in the same directory from which you're running gprolog) using

- `[filename].` *Watch out!* Prolog is old school - this will not work if gprolog.exe is running from a directory with spaces in it.

and you can consult the user (i.e, read what you type from the keyboard as a grammar) using

- `[user].`

In both cases, a successful consult will result in various messages (perhaps including warnings) ending in

- `yes`

Also note that the results of prior consults remain in prolog's context, so if you want to *actually overwrite* what you've previously consulted in, you should restart gprolog. (There are other ways, but restarting is probably easiest to explain....)

English grammar

grammar.pl:

```
sentence --> noun phrase, verb phrase.
noun phrase --> determiner, noun.
noun phrase --> proper noun.
verb phrase --> verb intransitive.
verb phrase --> verb transitive, noun phrase.


determiner --> [a].
determiner --> [the].
noun --> [whangdoodle].
noun --> [hippogriff].
proper noun --> [ophelia].
proper noun --> [rodriguez].
verb intransitive --> [pirouettes].
verb_transitive --> [chastises].
```

number matching grammar

grammarNM.pl:

```
sentence --> noun phrase(Number), verb phrase(Number).

noun phrase(singular) --> determiner, noun(singular).
noun phrase(plural) --> [the], noun(plural).
noun phrase(plural) --> noun(plural).
noun phrase(singular) --> proper noun.

verb phrase(Number) --> verb intransitive(Number).
verb phrase(Number) --> verb transitive(Number), noun phrase(Any).


determiner --> [a].
determiner --> [the].

noun(singular) --> [whangdoodle].
noun(singular) --> [hippogriff].
noun(plural) --> noun(singular), [s].

proper noun --> [ophelia].
proper noun --> [rodriguez].

verb intransitive(plural) --> [pirouette].
verb intransitive(singular) --> verb intransitive(plural), [s].

verb transitive(plural) --> [chastises].
verb_transitive(singular) --> verb_transitive(plural), [s].
```