Name of the Student: _____

Q 1

Define a function DEFRAG that takes a list of zero or more *s and +s and rearranges them so that all the stars precede the pluses. For instance (defrag '(+ + * + * +)) gives (* * + + + +).

Q 2

Take an integer n and generate n many random tosses of a fair coin. Your output should be a list of Hs and Ts.

Q 3

Given a list of coin tosses, count the number of successes – let head be a successful, and tail be an unsuccessful toss.

Q 4

Given a list of coin tosses, calculate the proportion of Hs. For instance, a call like (berno (toss-coin 100)) should give a number between 0 and 1. Remember that you can convert ratios to floating point numbers via FLOAT.

Q 5

Let us call "an experiment" a hundred successive tosses of a fair coin. You can represent an experiment as a hundred element list of Hs and Ts. Take an integer n and generate n many experiments, collecting them in a list.

Q 6

Given a list of values, construct a frequency table. For instance (freq-table '(a b r a c a d a b r a)) should give ((A . 5) (R . 2) (B . 2) (D . 1) (C . 1)), or (freq-table '(1 0 1 1 1 0 0 1 0 0 1)) should give ((1 . 6) (0 . 5)).

Q 7

Take an integer n, generate n many experiments, and tabulate the frequency of heads across the experiments. For instance a run for 1000 experiments – each of them 100 tosses of a fair coin – may give

```
((46 . 5799)
 (50 . 7853) (41 . 1593) (54 . 5907) (51 . 7716)
 (56 . 3949) (57 . 2993) (45 . 4934) (43 . 3026)
 (49 . 7761) (52 . 7227) (55 . 4884) (44 . 3853)
 (47 . 6638) (48 . 7379) (42 . 2245) (53 . 6866)
 (58 . 2193) (60 . 1093) (40 . 1063) (59 . 1537)
 (38 . 461) (61 . 693) (64 . 155) (71 . 1) (37 . 295)
 (35 . 85) (39 . 670) (33 . 26) (62 . 458) (68 . 10)
 (63 . 244) (36 . 156) (70 . 4) (65 . 77) (66 . 58)
 (34 . 51) (67 . 26) (32 . 12) (31 . 6) (72 . 1)
 (69 . 1) (73 . 1))
```

meaning, for instance, that in 6638 of 100000 experiments the number of heads was 47.

Q 8

The output for the previous exercise would look much informative if it is sorted. The builtin function SORT takes two arguments. One is the list (or sequence) to be sorted and the other is a two argument predicate (= a function that returns T or NIL) that compares its arguments. The predicate should be given in such a way that when it gives T, it would mean the first argument precedes the second. For instance, to sort a list of integers named intlist in ascending order (from smaller to larger), do (sort intlist #'<). The predicate can be as complex as you like – lambda's are very helpful with sort.

Sort your freq-tables with respect to first and second components.