Worth: 2% **Due:** *Before* 10pm on Tuesday 28 February 2012.

Remember to write your *full name* and *student number* prominently on your submission.
Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission ***every*** *source of information you used to complete this homework (other than your own lecture and tutorial notes, and materials available directly on the course webpage). For example, indicate clearly the* ***name*** *of every student with whom you had discussions, the* ***title*** *of every additional textbook you consulted, the* ***source*** *of every additional web document you used,* etc.

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.

---

1. Given an unsorted list of $n \geqslant 3$ distinct numbers, we wish to find the three smallest numbers, in order. There is an obvious naive algorithm to solve this problem: first find the smallest element (using $n-1$ comparisons between list elements), then find the smallest of the remaining elements ($n-2$ comparisons), and then find the smallest of the rest ($n-3$ comparisons). This algorithm makes a total of $3n-6$ comparisons between list elements, and we would like to do better. (*Note that for this exercise, we count only the number of comparisons* ***between list elements*** *and ignore all other operations performed by the algorithms.*)

   Suppose that you have access to the following subroutines.

```
# Returns (min(a, b), max(a, b)), using 1 comparison between list elements.
MINMAX(a, b):
    if a < b:  return (a, b)
    else:      return (b, a)

# Returns the three smallest of a, b, c, in order, using 3 comparisons between list elements.
SOLVE_THREE(a, b, c):
    (a, c) = MINMAX(a, c)    # after this, a < c
    (b, c) = MINMAX(b, c)    # after this, b < c
    (a, b) = MINMAX(a, b)    # after this, a < b
    return (a, b, c)         # at this point, a < b < c

# Returns the three smallest of a, b, c, d, in order, using 5 comparisons between list elements.
SOLVE_FOUR(a, b, c, d):
    # ... code omitted...

# Returns the three smallest of a, b, c, d, e, in order, using 7 comparisons between list elements.
SOLVE_FIVE(a, b, c, d, e):
    # ... code omitted...
```

   Give a divide-and-conquer algorithm that takes a list of n distinct elements (where $n \geqslant 3$) and returns the three smallest elements (in order). Your algorithm may make calls to any of the subroutines defined above. Moreover, if $C(n)$ is the maximum number of comparisons between list elements that your algorithm makes when working on a list of size n, then C should satisfy the following recurrence relation:

$$\begin{aligned} C(3) &= 3, \\ C(4) &= 5, \\ C(5) &= 7, \\ C(n) &= C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor) + 3, \qquad \text{for all } n \geqslant 6. \end{aligned}$$

Write your algorithm in pseudo-code (in a style similar to the subroutines above) together with a *short* English description of how your algorithm works, as a justification that it is correct. *In particular, please explain any notation in your algorithm that is not standard Python syntax, to avoid misunderstandings.*

HINT: Notice that the form of the recurrence relation satisfied by C dictates the way in which the algorithm must work.

2. Remember the goal of doing all this work: to solve the problem using fewer than $3n-6$ comparisons between list elements (the number performed by the naive algorithm).

   Show how to apply the Master Theorem to the recurrence relation for $C(n)$ given in the previous question. State clearly how you use the recurrence relation to obtain the values of the various constants in the statement of the theorem, and what the theorem allows you to conclude about $C(n)$.

3. You should find that the answer provided by the Master Theorem is not useful: it does not allow you to conclude that your algorithm is necessarily better than the naive algorithm...

   To confirm that your algorithm is better, find an exact closed-form expression for the value of $C(n)$. Then, write a rigorous proof that your expression is correct. (What matters here is the proof, because it shows that your expression is correct — how you find your expression is irrelevant, as long as you can prove it. This means that you do *not* need to show how you obtained your expression. All we want to see is your proof.)