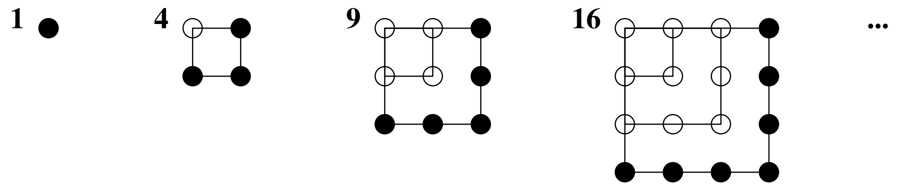
Worth: 8% **Due:** *Before* 10pm on Tuesday 6 March 2012.

Remember to write the *full name*, *student number*, and *CDF/UTOR email address* of *each group member* 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 (other than group members), the* ***title*** *of every additional textbook you consulted, the* ***source*** *of every additional web document you used,* etc.

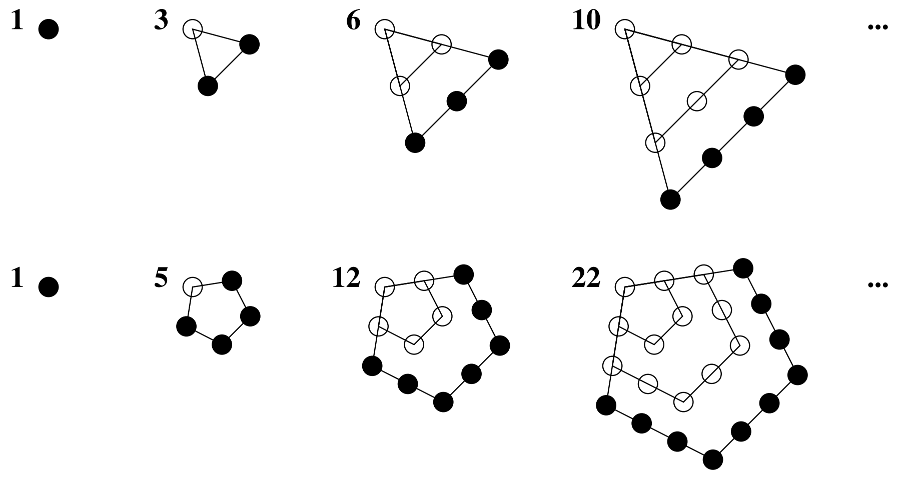
For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified,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

---

1. When we think of "square numbers", we tend to think in terms of algebraic expressions: $1^2, 2^2, 3^3, \ldots$ But there is another, older, geometric interpretation: a positive integer is "square" if it is possible to arrange that many dots in a regular square pattern, as shown below:



This geometric point of view can be extended to other polygons, so that we can speak of "triangular" numbers, "pentagonal" numbers, *etc.* This is illustrated below for triangular and pentagonal numbers.



Notice that the n-th number in all of these sequences is obtained from the $(n-1)$-st number by adding one "layer" of dots to obtain an outer polygon with n dots per side. (This extra layer is coloured black in the pictures above.) For example, the 4th pentagonal number simply adds just enough dots to the 3rd pentagonal number in order to create an outer pentagon with 4 dots on each side.

(a) By analogy with the sequences shown above, define $T(k, n)$ to be the n-th number in the sequence of "k-sided polygonal numbers". For example, $T(3, 2) = 3$ is the second triangular number and $T(5, 3) = 12$ is the third pentagonal number.
For all $k \geqslant 3$ and all $n \geqslant 1$, write down an exact recurrence relation satisfied by $T(k, n)$. Justify that your recurrence is correct, *i.e.*, explain what each term represents and how you computed it.

(b) Solve your recurrence to find a closed-form expression for the values of $T(k, n)$, and give a rigorous proof that your solution is correct, based on your recurrence relation from the previous part.

2. Suppose that you have to choose between three different algorithms A, B, C to solve a certain problem.

   (a) Algorithm A solves problems of size n by dividing them into *five* subproblems, each of size $n/2$, recursively solving each subproblem, and combining the solutions in *linear* time.

   (b) Algorithm B solves problems of size n by dividing them into *two* subproblems, each of size $n - 1$, recursively solving each subproblem, and combining the solutions in *constant* time.

   (c) Algorithm C solves problems of size n by dividing them into *nine* subproblems, each of size $n/3$, recursively solving each subproblem, and combining the solutions in *quadratic* time.

   Which algorithm would you choose and why? Explain your reasoning and show your work.

3. Suppose that you have $k \geqslant 1$ sorted arrays, each one containing $n \geqslant 1$ elements, and you wish to combine them into a single sorted array with kn elements.

   (a) One possibility is to merge the first two arrays, then merge the third array into the result, then merge the fourth array into the result, and so on.

   What is the worst-case running time of this algorithm, as a function of k and n? Explain how you obtained your answer (show all of your work).

   (b) Give a divide-and-conquer algorithm that solves this problem more efficiently than the algorithm in the first part, and justify that your solution is more efficient.

   Include a high-level description of the main idea of your algorithm (or enough comments throughout your pseudo-code to make it clear why your algorithm is doing what it does), and give a detailed explaination to justify that your algorithm is more efficient (show all of your work).

4. Recall that $\gcd(x, y)$ is the *greatest common divisor* of x and y, for all natural numbers x, y. (*For this question, we* ***include*** 0 *in the set of natural numbers, for convenience.*) Consider the following recursive algorithm to compute $\gcd(x, y)$.

```
REC-GCD($x, y$):
    # Precondition: $x, y \in \mathbb{N}$.
    # Postcondition: the algorithm returns $\gcd(x, y)$.
    **if** $x = 0$ or $y = 0$:    **return** $x + y$
    **if** $x = 1$ or $y = 1$:    **return** $1$
    **if** $x$ is even:
        **if** $y$ is even:   **return** $2 \times$ REC-GCD($x/2, y/2$)
        **else**:             **return** REC-GCD($x/2, y$)
    **else**:
        **if** $y$ is even:   **return** REC-GCD($x, y/2$)
        **else**:
            **if** $x < y$:     **return** REC-GCD($(y - x)/2, x$)
            **else**:           **return** REC-GCD($(x - y)/2, y$)
```

   (a) Give a detailed analysis of the algorithm's worst-case running time (starting from a recurrence relation, showing your work to obtain a closed form, and giving rigorous proofs that your closed form is correct).

   Let n be the total number of bits to represent both x and y (*i.e.*, $n = n_1 + n_2$ where it takes n_1 bits to represent x and n_2 bits to represent y). Also, assume that it takes linear time to perform basic arithmetic operations (like $+$ and $-$).

(b) Write a detailed proof that the algorithm is correct.

HINT: Take the time to write down a precise definition of “greatest common divisor” (gcd) and clearly state the properties of gcd you need to use in your proof.