Task

This task consists of two parts.

You should prepare your answers electronically, in a Word document or pdf file. Where mathematical formulas, diagrams etc are needed you can use handwriting and insert them into the document as pictures, but any substantial amount of text should be typeset and not handwritten. You should have Part A and Part B in different files.

Please do not include your name or other personally identifiable information in your submission.

Part A

Consider the following pseudocode:

```
f(int n, int d) {
  println(space(d) + "n=" + n + " begins");
  if (n > 1) {
   f(n/2, d+1);

    println(space(d+1) + "hi");

    f(n/2, d+1);
}
 println(space(d) + "n=" + n + " ends");
}
```

where println(s) prints the string s on its own line, space(d) is a string of d spaces, and the + in the println means string concatenation. For example, if n=4 and d=2, the commands

```
println(space(d) + "n=" + n + " begins");

println(space(d+1) + "hi");
```

will print

```
__n=4 begins

___hi
```

(where the space character is replaced by just so you can see it.)

(a) Write down the output printed by f(4,0)

(b) Let T(n) be the no of lines printed by f(n,0). Write down a recurrence formula for T(n), including the base case. You can assume n is a power of 2.

(c Solve the recurrence to find out how many lines are printed by f(n,0), as a function of n. (You must not use the Master Theorem.)

Part B

In some situations, we do not require an array to be fully sorted, but just “approximately” sorted. There are many ways this can be defined. In this question, we consider one particular measure concerning the number of pairs of elements that are “wildly” out of order.

In a fully sorted array A[1..n], whenever i < j we must have A[i] ≤ A[j]. In other words, there are no pairs of elements where i < j but A[i] > A[j]. We relax this definition a bit; for example, we can allow A[i] to be “a little bit” larger than A[j] but not too much. Specifically, given an array A[1..n], a pair of elements A[i] and A[j] is said to be a bad pair if i < j and A[i] > A[j] + 2. In other words, the “earlier” element is larger than the later element by 3 or more. For example, in [1, 5, 2, 3, 10, 9, 8, 4] there are four bad pairs: (5, 2), (10, 4), (9, 4) and (8, 4). We may accept an array to be “approximately sorted” if there are no bad pairs. This is a weaker notion than being fully sorted: an array may have no bad pairs but is still not fully sorted, for example [1, 2, 4, 3, 5, 8, 6, 7].

But before we consider how to approximately sort an array (which is not what most of this question is about), we consider the problem of identifying or counting these bad pairs.

(a) What is the minimum and maximum number of bad pairs an n-element array can have? [5 Points + 5 Points]

(b) Give a straightforward O(n2) time algorithm to output all bad pairs. [10 Points]

(c) If we are only interested in the number of bad pairs (without listing them all), we can do better than (b). First, consider a special case in which the array A is such that the first half A[1..n/2] is fully sorted, the second half A[n/2 + 1..n] is fully sorted, but the whole array may not be fully sorted. An example is A = [1, 3, 6, 7, 2, 4, 5, 8]. Give an O(n) time algorithm to report the number of bad pairs in such an array A.
(Hint: in this special case, a bad pair can only happen when the two numbers involved are in different halves. Also note that you are only required to count the number of bad pairs. Do not attempt to output all bad pairs.) [20 Points]

(d) Give an O(n log n) time algorithm to report the number of bad pairs in an array A (that may not have the special property in (c)). You can assume n is a power of 2.
(Hint: Use divide and conquer and modify merge sort, so that it counts the number of bad pairs while sorting the array at the same time. Make use of part (c).) [20 Points]

(e) A student Alice said the problem of counting the number of bad pairs has a lower bound of Ω(n log n). Here is her argument:

“As we see in (c) and (d), counting the number of bad pairs requires sorting the array. Since sorting takes Ω(n log n) time, this problem also requires at least this amount of time.”

Is her argument correct? Explain your answer. [5 Points + 5 Points]

(f) (Bonus part) Note that all the above is about counting the number of bad pairs, not approximately sorting an array. Unfortunately, sorting an array approximately (to remove all bad pairs) is not easier than sorting an array fully. Prove an Ω(n log n) lower bound on the time complexity of this problem.
[+5 bonus Points]

Points criteria

For all parts with only 5 points for each answer, points given are either 0 or 5 based on correctness. Incorrect answers may be given partial points depending on merit.

For Part A(c), points are given roughly based on this table:

0	Does not even know what “solve a recurrence” means.
4	At least this looks like solving some recurrence but there is almost nothing correct about it.
8	At least some ideas along the right lines are demonstrated, such as expanding the first few steps (probably wrongly).
10	Have broadly the right steps at least up to the first few expansions and the general pattern although may have errors.
12	Have broadly the right steps at least up to the first few expansions, the general pattern and summing up the terms, although may have some errors.

16	Have broadly the right steps all the way but have some errors.
18	Minor/inconsequential mistakes.
20	Everything is correct.

Correct steps following earlier wrong results (e.g. stating the wrong recurrence but solving the wrong one correctly) may still give you most of the allocated points.

In parts that involve design of algorithms (Part B (b)-(d)), you should:

- state the algorithm in pseudocode;
- explain (in words) some intuition behind your algorithm, and/or why it works correctly;
- analyse (by simple reasoning, or more mathematically e.g. a recurrence formula if needed) the time complexity of your algorithm.
  And the answers will be marked roughly according to this table:

0%	Nothing relevant was given.
20%	There are some ideas towards a correct algorithm but they are very vague or would not lead to anywhere near the upper bounds required. Pseudocode / explanation / analysis all very wrong or missing.
40%	Some correct ideas but incomplete, or would not lead to the upper bounds expected. Pseudocode or explanation missing or wrong. Analysis missing, or wrong, or follows the incomplete ideas “correctly” to lead to sub-optimal bounds.
60%	Have the correct main ideas that would lead to the upper bounds expected, but this is not matched by the analysis. Some of pseudocode / explanation / analysis missing or wrong, while the others are in the right direction but have errors.

80%	Have the correct ideas and all required elements, but have some errors in presenting the ideas, e.g. pseudocode or analysis have some errors.
90%	Everything is mostly correct, with some small mistakes.
100 %	Everything is correct and complete.

(In all parts) you can always use the Master Theorem unless expressly forbidden (but you should show the steps in using it). Where manual solving of recurrence is required, solving with Master Theorem instead (and correctly) will yield 40% of the allocated points.