

Due February 13, 2:45pm

Instructions: Write your name, the username for your instructional account, your TA's name, and your discussion section time prominently on the first page of your homework. Also list your study partners for this homework, or "none" if you had no partners.

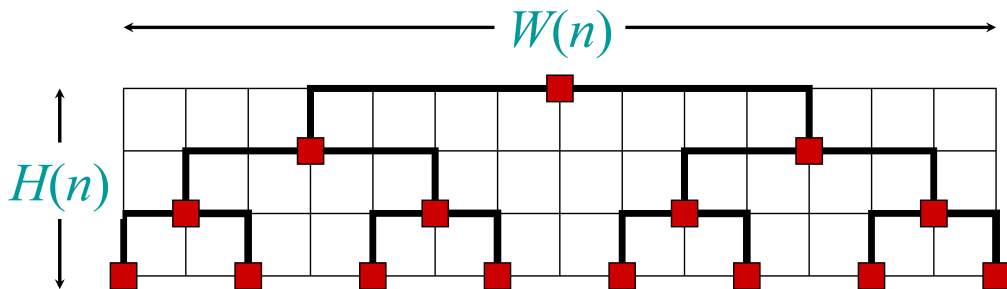
Any time that you are asked for an algorithm, please provide: (a) the pseudocode of your algorithm, (b) a brief explanation of the main idea/intuition underpinning your algorithm, (c) a proof of correctness, (d) the asymptotic running time of the algorithm, using $\Theta(\cdot)$ notation, (e) justification for your running time calculation.

1. (30 pts.) Chip design

A chip designer approaches you with the following problem. She wants to lay out a complete binary tree with n leaves on the chip, where n is a power of two. (Imagine that the leaves contain n inputs, and the goal is to compute the Logical AND of these signals, using two-input AND gates.) The nodes and wires are required to follow a rectangular grid. She wants to minimize the total area required for the tree.

In the following parts, show your work and justify your answers.

(a) Her first thought is to lay out the wires like this:



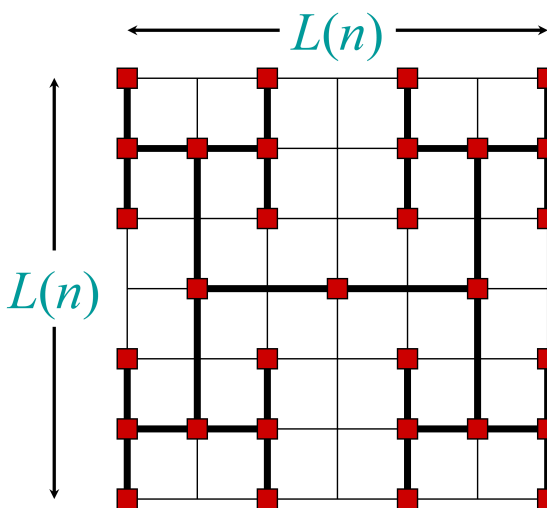
Let $W(n)$ denote the width of this arrangement, in number of grid-squares, and $H(n)$ the height of this arrangement. For instance, in the above picture we have $n = 8$, $H(8) = 3$, and $W(8) = 14$.

Find an explicit formula for $H(n)$.

(b) Find a recurrence relation for $W(n)$.

(c) Solve the recurrence relation for $W(n)$, to obtain an asymptotic expression for $W(n)$. In other words, find a function $f(n)$ such that $W(n) \in \Theta(f(n))$.

- (d) The total area of her scheme is given by $A(n) = H(n) \times W(n)$. Find an asymptotic expression for $A(n)$. In other words, find a function $f(n)$ such that $A(n) \in \Theta(f(n))$.
- (e) A flash of inspiration strikes you in the shower, and you have another idea for a possible chip layout, like this:



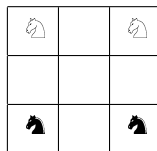
Assume that n is an even power of 2 (e.g., 1, 4, 16, 64, etc.). Then this arrangement will be a square, so let $L(n)$ denote the length of the side of the square, as a function of n . For instance, in the picture above we have $n = 16$ and $L(n) = 6$.

Find a recurrence relation for $L(n)$.

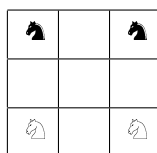
- (f) Solve the recurrence relation for $L(n)$, to obtain an asymptotic expression for $L(n)$ (up to a multiplicative factor). In other words, find a function $f(n)$ such that $L(n) \in \Theta(f(n))$.
- (g) The total area of your scheme is given by $A'(n) = L(n)^2$. Find an asymptotic expression for $A'(n)$. In other words, find a function $f(n)$ such that $A'(n) \in \Theta(f(n))$.
- (h) Which is better, for large n ? The chip designer's layout depicted in part (a), or your layout from part (e)?

2. (30 pts.) The four-knight puzzle

Alice sets up a 3×3 chessboard, with two white knights in the upper corners and two black knights in the lower corners, like this:



She asks you whether it is possible to find a sequence of legal chess moves, starting from the above position, so that you reach the following position:



- (a) Design an algorithm to answer her question. Your algorithm must be guaranteed to give the correct answer. Your algorithm must not have any chance of looping forever (even if there is no way to get from the first position to the second position). Your algorithm should be reasonably efficient.

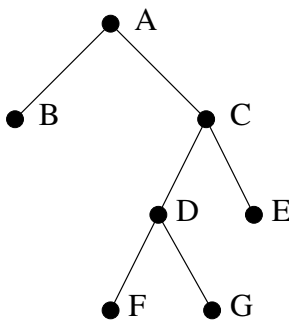
Your algorithm should work correctly for any $m \times m$ board with any number k of knights and any start and end configuration. If N denotes the set of possible configurations of k knights on a $m \times m$ board, your algorithm should run in $O(kN)$ time.

- (b) Implement your algorithm, for the specific case Alice asked you about (i.e., with $m = 3, k = 4$, and the start and end configurations shown above). What is the answer to Alice's question? Attach a printout of your code to the end of this homework.

3. (30 pts.) Tree-splitting

Design an algorithm to answer the following query: given a tree T with n nodes and an integer k , find an edge (u, v) in the tree such that deleting this edge splits the tree into two connected components with k and $n - k$ nodes. If no such edge exists, your algorithm should report this fact. Your algorithm should run in $O(n)$ time.

Example. Consider the following tree T :



If $k = 4$, we could delete the edge (C, D) to split T into one connected component with 4 nodes (namely, A, B, C, and E) and one connected component with 3 nodes (namely, D, F, and G).

If $k = 5$, we can delete the edge (A, C) to split T into one connected component with 5 nodes (namely, C, D, E, F, and G) and one connected component with 2 nodes (namely, A, and B).

4. (10 pts.) Fast resets

Bob has a Java program that manipulates lots of objects, and he has asked you to help him with the following problem. Each object can be logically in either a marked or unmarked state. Objects should start out in the unmarked state. You need to be able to support the following four operations:

- **MARK(O):** Causes the object O to become marked.
- **UNMARK(O):** Causes the object O to become unmarked.
- **ISMARKED?(O):** Returns true if O is currently marked, or false otherwise.
- **RESET():** Unmarks every object in existence.

In other words, Bob wants to be able to use these operations in his program, and asks you for help with how to implement them. Provide Bob with an algorithm that efficiently supports these four operations. Each operation must run in $\Theta(1)$ machine instructions, i.e., each operation must take at most T machine instructions, for some constant T that is independent of the number of objects manipulated by Bob's program. You can assume that each basic operation in Java (dereferencing a reference, performing a single arithmetic or logical operation, etc.) takes $\Theta(1)$ machine instructions. You can assume there is a field in each object that's reserved for your use; you can specify what it will get initialized to. You may assume that Bob's program does not execute `RESET()` more than 2^{60} times.

Note. If you only had to support the first three operations, this would be an easy problem. For instance, you could add a boolean field *marked* (initially initialized to false) to every object O . Then, you could use the following algorithms:

`MARK(O):`

1. Set $O.marked := \text{true}$.

`UNMARK(O):`

1. Set $O.marked := \text{false}$.

`ISMARKED?(O):`

1. Return $O.marked$.

But supporting all four operations, in $\Theta(1)$ time for each, takes some cleverness.