Write your answers in the separate answer booklet.

You have 120 minutes (after you get the answer booklet) to answer five questions.

Please return this question sheet and your cheat sheet with your answers.

1. **Short answers:**

   (a) Solve the following recurrences:

   - $A(n) = A(5n/11) + O(\sqrt{n})$
   - $B(n) = 8B(n/2) + O(n^2)$
   - $C(n) = C(n/2) + C(n/3) + C(n/6) + O(n)$

   (b) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrences, and state the running time of the resulting iterative algorithm to compute the requested function value.

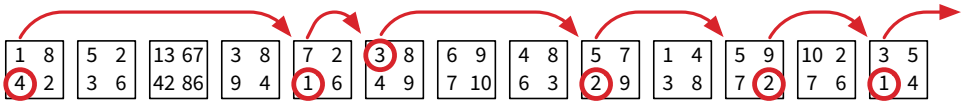
   - Compute $Foo(1, n)$ where

$$Foo(i,k) = \begin{cases} 0 & \text{if } i \geq k-1 \\ \max\left\{ Foo(i,j) + Foo(j,k) \;\middle|\; i < j < k \right\} + \sum_{j=i}^{k} A[j] & \text{otherwise} \end{cases}$$

   - Compute $Bar(n, 1)$ where

$$Bar(i,s) = \begin{cases} \infty & \text{if } i < 0 \text{ or } s > n \\ 0 & \text{if } i = 0 \\ \min\left\{ \begin{array}{c} Bar(i, 2s), \\ X[i] \cdot s + Bar(i-s, s) \end{array} \right\} & \text{otherwise} \end{cases}$$

2. *Quadhopper* is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A quadhopper puzzle that allows six moves. (This is ***not*** the longest legal sequence of moves.)

   (a) ***Prove*** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every quadhopper puzzle.

   (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given quadhopper puzzle.

3. After moving to a new city, you decide to walk from your home to your new office. To get a good daily workout, you want to reach the *highest* possible altitude during your walk (to maximize exercise), while keeping the total length of your walk below some threshold (to get to your office on time). Describe and analyze an algorithm to compute the best possible walking route.

   Your input consists of an undirected graph G, where each vertex v has a height $h(v)$ and each edge e has a positive length $\ell(e)$, along with a start vertex s, a target vertex t, and a maximum length L. Your algorithm should return the *maximum height reachable* by a walk from s to t in G, whose total length is at most L.

4. Suppose you are given a string of symbols, representing a message in some foreign language that you do not understand, in an array $T[1..n]$. You have access to a black-box subroutine IsWord that can decide whether an arbitrary string w is a word in $O(|w|)$ time.

   You eagerly implement and run the text-splitting algorithm we saw in class, only to discover that the given string *cannot* be split into words! Apparently, as a crude form of cryptography, the message has been corrupted by adding extra symbols between words.

   So you decide instead to look for as many non-overlapping words in T as possible. A ***verbal subsequence*** of T is a sequence of non-overlapping substrings of T, each of which is a word. The *length* of a verbal subsequence is the number of words it contains. Describe and analyze an algorithm to find the length of the longest verbal subsequence of a given string T.

   For example, suppose IsWord(w) returns True if and only if w is a common English word. Then (STUDY, AM, ICE, TRAP, RAMBLE) and (DYNAMIC, EXTRA, PROGRAM) are verbal subsequences of the string STUDYNAMICEXTRAPROGRAMBLE:

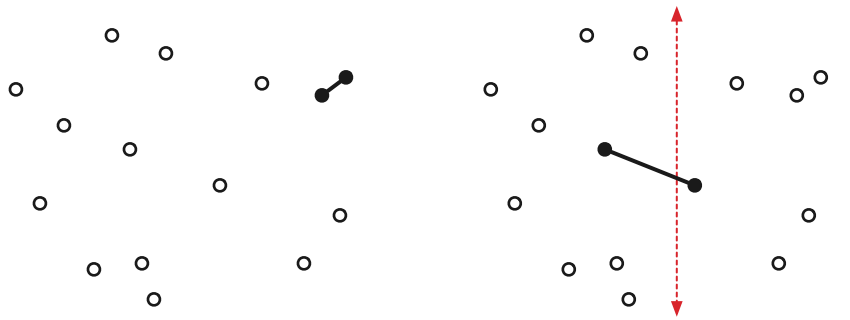
   

   Thus, given the input string STUDYNAMICEXTRAPROGRAMBLE, the output of your algorithm should be *at least* 5, which is the length of the subsequence (STUDY, AM, ICE, TRAP, RAMBLE). (This string may contain longer verbal subsequences.)

5. This problem asks you to design an algorithm to find the closest pair of points in a given set P of n points in the plane. Points in P are represented by an array $P[1..n]$ of coordinate pairs; the coordinates of the ith point are $(P[i].x, P[i].y)$. The points are sorted by their x-coordinates, and all x- and y-coordinates are distinct. In particular, we have $P[i].x < P[i+1].x$ for every index i. Timothy has helpfully implemented a pair of subroutines for you.

   - IWALKTHELINE(P, a) computes the closest pair of points in a given n-point set P that cross the vertical line $x = a$, in $O(n)$ time. Specifically, this function returns the indices i and j of the closest pair of points such that $P[i].x \le a < P[j].x$. See the figure below for an example.
   - DISTANCE(p, q) returns the Euclidean distance between two points p and q, in $O(1)$ time.

   Describe and analyze an algorithm to compute the Euclidean distance between the closest pair of points in P, using Timothy's subroutines as black boxes. (Do *not* try to implement Timothy's subroutines yourself!)



The closest pair of points (computed by your algorithm)

The closest pair crossing a vertical line (computed by IWALKTHELINE)