­­­lWinning

f1.a.i)x

closest [x1, x2] = (x1, x2)  
closest (x1:x2:xs) = (x1, x2) ⊓ closest (x2:xs)

This is linear as it performs one comparison for each of the n-1 pairs in the list (assuming ⊓ is constant time)

1.a.ii)

An algorithm that

1. Divides the problem into distinct sub-problems.
2. Recursively calls itself to solve each subproblem, obtaining sub-solutions.
3. Combines the sub-solutions into a solution for the main problem.

To avoid infinite recursion, the algorithm must be able to solve a small enough problem directly without dividing it.

1.a.iii)

closest' [x1, x2] = (x1, x2)  
closest' [x1, x2, x3] = (x1, x2) ⊓ (x2, x3)  
closest' xs = closest' left ⊓ closest' (last left : right)  
  where  
    (left, right) = splitAt (length xs `div` 2) xs

Alternative using head function as requested by the question:

closest' [x1, x2] = (x1, x2)  
closest' xs = closest' (left ++ [head right]) ⊓ closest' (last left : right)  
  where  
    (left, right) = splitAt (length xs `div` 2) xs

Another alternative using head function as requested by the question:

closest' [] = (-maxInt, maxInt)  
closest' [x] = (-maxInt, maxInt)  
closest' [x1, x2] = (x1, x2)  
closest' xs = (closest' left) ⊓ (last left, head right) ⊓ (closest' right)  
  where  
    (left, right) = splitAt (length xs `div` 2) xs

Tclosest’(2) = 0

Tclosest’(3) = 1

Tclosest’(n) = TsplitAt(n) + Tlast(n/2) + 2 × Tclosest’(n/2)

= n/2 + n/2 + 2 × Tclosest’(n/2)

= n + … + n (log n times)

Tclosest’(n) ∈ O(n log n)

1.b.i)

head :: AList a -> a  
head (AList fst \_ arr) = arr ! fst  
last :: AList a -> as  
last (AList \_ lst arr) = arr ! lst  
split :: AList a -> (AList a, AList a)  
split (AList fst lst arr)   
  = (AList fst (fst + half) arr,  
          AList (fst + half + 1) lst arr)  
    where  
      half = (lst - fst + 1) `div` 2

Alternative implementation of half => (fst + lst) div 2

1.b.ii)

closest'’ :: [Integer] -> (Integer, Integer)  
closest’’ = closest’’’ . fromList  
f  
closest’’’ :: AList Int Integer -> (Integer, Integer)  
closest''’ xs@(AList f l a)  
  | l - f == 1 = (a ! f, a ! l)  
  | l - f == 2 = (a ! f, a ! (f + 1)) ⊓ (a ! (f + 1), a ! l)  
  | otherwise  = closest'' left ⊓ closest'' right  
    where  
      (left, AList fst lst arr) = split xs  
      right = AList (fst - 1) lst arr

Alternative answer

closest’’ :[Integer]->(Integer, Integer)

closest’’ xs = go (fromList xs)o

where

go xs@(AList i j axs)

| j - i == 0 = (-maxInt, maxInt)

| j - i == 1 = (axs ! i, axs ! j)

| otherwise = (go us) ⊓ (last us, head vs) ⊓ (go vs)

where

(us,vs) = split xs

2.a.i)

catalan' n = arr ! n  
  where  
    arr = tabulate (0, n) memo  
    memo 0 = 1  
    memo n = sum [(arr ! i) \* (arr ! (n - i - 1)) | i <- [0..(n-1)]]

2.a.ii)

Each entry n in the array takes a list comprehension over n elements (0 to n – 1) to build, and each iteration of the list comprehension performs 2 constant time look-ups. Therefore each entry took linear time O(n) to build, so we simply take the sum of each entry i ∈[0..n]:

∑ i = n(n + 1)/2 ∈O(n2)

2.b.i)

(A0 × A1) × A2 => (2 × 2 × 2) + (2 × 2 × 3) = 20

A0 × (A1 × A2) => (2 × 2 × 3) + (2 × 2 × 3) = 24

2.b.ii)

|  |
| --- |
| minMatrix xs = chain (fromList xs) (0, length xs - 1)  chain :: Array Int Int -> (Int, Int) -> Int chain arr (i, j)  | i == j - 1 = 0  | otherwise = minimum [count k | k <- [i + 1 ... j -1]]  where  count k = chain arr (i, k) + chain arr (k, j) + arr ! i \* arr ! k \* arr ! j |

2.b.iii)

|  |
| --- |
| minMatrix' xs = chain' (fromList xs) (0, length xs - 1)  chain' :: Array Int Int -> (Int, Int) -> Int chain' arr (i, j) = table ! (i, j)  where  table = tabulate ((i, i), (j, j)) memo  memo (i, j)  | i == j - 1 = 0  | otherwise = minimum [count k | k <- [i + 1, j -1]]  where  count k = table ! (i, k) + table ! (k, j) + arr ! i \* arr ! k \* arr ! j |

TestCases for above solution:

minMatrix' [2,2,2,3] = 20

minMatrix' [10, 20, 30] = 6000

minMatrix' [10, 20, 30, 40, 30] = 30000

minMatrix' [40, 20, 30, 10, 30] = 26000

[Executable code](https://gist.github.com/alyata/de9a65a549e30d87879cd8ca7477a816) for a different solution