Exercise 1.3.1. Let $A[1..n]$ be in a *rotated* sorted order. That is, there exists an index $i \in [n]$, called the *rotation index*, such that the concatenation of $A[i..n]$ and $A[1..i-1]$ is sorted in increasing order. (One can think of $A[1..n]$ as being sorted initially, and then rotated cyclically to the right by $i-1$ slots). The goal is to compute the unknown rotation index i. For simplicity, you may assume all the elements are distinct.

Solution.

def func `computeRotationIndex(`$A[1..n]$`):`

/ Given an array of length n in 'rotated' sorted order, return the 'rotation' index. */*

1. If $n \leq 1$, return 1.
2. Let $k := \lfloor \frac{n}{2} \rfloor$.
3. If $(A[mid] > A[n])$:
   A. Return $k +$ `computeRotationIndex`$(A[k+1..n])$.
4. Else if $(k = 1)$ OR $(A[k] > A[n])$:
   A. Return k
5. Else:
   A. Return `computeRotationIndex`$(A[1..k-1])$

□

Exercise 1.3.2. We say that $A[1..n]$ is a *mountain* if there exists an index $i \in [n]$, called the *peak*, such that $A[1..i]$ is sorted in increasing order and $A[i+1..n]$ is sorted in decreasing order. Given a mountain $A[1..n]$, find the peak. For simplicity, you may assume all elements are distinct.

Solution.

def func `findPeak(`$A[1..n]$`):`

/ Given a 'mountain' of length n, find the index of the 'peak'. */*

1. If $(n \leq 1)$, return 1
2. Let $k := \lfloor \frac{n}{2} \rfloor$
3. If $(A[k] < A[k+1])$:
   A. Return k + `findPeak`$(A[k+1..n])$
4. Else if $(k > 1)$ AND $(A[k-1] < A[k])$:
   A. Return k
5. Else:
   A. Return `findPeak`$(A[1..k])$:

□

Exercise 1.3.3. We say an index $i \in [n]$ is a *local minimum* if either

(a) $i = n = 1$,

(b) $i = 1$ and $A[1] \leq A[2]$,

(c) $i = n$ and $A[n] \leq A[n-1]$, or

(d) $1 < i < n$, $A[i-1] \geq A[i]$ and $A[i] \geq A[i+1]$.

The goal is to find a local minimum from an array $A[1..n]$.

Solution.

def func `findLocalMinimum(`$A[1..n]$`):`

/ Given an array of length n, identify the 'local minimum'.* **/*

1. If $(n = 1)$: return 1 *// (a)*
2. Let $k := \lfloor \frac{n}{2} \rfloor$
3. If $\big[(k = 1)$ AND $(A[k] \leq A[k+1])\big]$ OR *// (b)*
   $\big[(k = n)$ AND $\big(A[k] \leq A[k-1]\big)\big]$ OR *// (c)*
   $\big[(1 < k < n)$ AND $(A[k-1] \geq A[k])$ AND $(A[k] \leq A[k+1])\big]$: *// (d)*
   A. Return k
4. If $\big[(k > 1)$ AND $A[k-1] < A[k]\big]$:
   A. Return `findLocalMinimum`$(A[1..k-1])$
5. Else:
   A. Return `findLocalMinimum`$(A[k+1..n])$

□

Exercise 1.3.4. Prove that each of the problems above have a lower bound of $\Omega(\log n)$ queries in the comparison model.

Solution. Each of the problems above have n possible values for the specified index. Any given comparison can have 2 possible outcomes: **True** or **False** . Thus in the worst case scenario we have $2^k \geq n$ where k is the number of queries. Solving for k gives us $k \geq \log_2 n$. Therefore, the lower bound is $\Omega(\log n)$. □

Exercise 2.1.2. In the *cyclic towers of Hanoi* problem, a ring can only be moved in "cyclic" order from post A to post B, from post B to post C, and from post C to post A. The goal is to move n sorted rings from post A to post B.

Solution.

def func `cyclic-Hanoi(`n, $from$, to`):`

/ Given an integer n and starting/ending posts, A, B, or C, move n sorted rings from post A to post B. */*

1. If $n = 0$ or $from = to$:
   A. return
2. If $\big[(from = A)$ AND $(to = B)\big]$ OR
   $\big[(from = B)$ AND $(to = C)\big]$ OR
   $\big[(from = C)$ AND $(to = A)\big]$: *// One step*
   A. If $\big[(from \neq A)$ AND $(to \neq A)\big]$:
      1. Let other := A
   B. Else if $\big[(from \neq B)$ AND $(to \neq B)\big]$:
      1. Let other := B
   C. Else:
      1. Let other := C
   D. `move-Cyclic`$(n-1, from, other)$
   E. Move the top ring from $from$ to to
   F. `move-Cyclic`$(n-1, other, to)$
3. Else: *// Two steps*
   A. If $from = A$:
      1. $mid := B$
   B. Else if $from = B$:
      1. $mid := C$
   C. Else:
      1. $mid := A$
   D. `moveCyclic`$(n-1, from, mid)$.
   E. Move the top ring from $from$ to mid
   F. Move the top ring from mid to to
   G. `moveCyclic`$(n-1, mid, to)$.

□

Exercise 2.1.3. The *double-cyclic towers of Hanoi* is like the cyclic towers of Hanoi problem except now the goal is to move the n sorted rings from post A to post C.

Solution. To call the function, use `double-cyclic-Hanoi`(n, A, C, B).

def func `double-cyclic-Hanoi`(n, $from$, to, $spare$):

/* *Given an integer n, a starting post, ending post, and spare post, move n sorted rings from the starting post from to the ending post to.* */

1. If $(n = 0)$ OR $(from = to)$:
   A. Return
2. Let $oneStep :=$ **False** .
3. If $\big[(from = A)$ AND $(to = B)\big]$ OR
   $\big[(from = B)$ AND $(to = C)\big]$ OR
   $\big[(from = C)$ AND $(to = A)\big]$: // *One step*
   A. `double-cyclic-Hanoi`$(n - 1, from, spare, to)$.
   B. Move the top ring from $from$ to to.
   C. `double-cyclic-Hanoi`$(n - 1, spare, to, from)$.
4. Else: // *Two steps*
   A. If $from = A$:
      1. $next := B$
   B. Else if $from = B$:
      1. $next := C$
   C. Else:
      1. $next := A$
   D. `double-cyclic-Hanoi`$(n - 1,\ from,\ next,\ to)$.
   E. Move the top ring from $from$ to $next$.
   F. Move the top ring from $next$ to to.
   G. `double-cyclic-Hanoi`$(n - 1,\ next,\ to,\ from)$.

□

Exercise 2.1.4. In the *thick towers of Hanoi* problem, we have $3n$ rings of n distinct sizes with three copies of each ring. (Rings of the same size can stack on top of each other.) The goal is to move all $3n$ rings from post A to post B.

Solution.

`def func thick-Hanoi(`n, A, B, C`):`

`/*` *Given an integer n, and posts A, B, and C, move $3n$ rings from post A to post B.* `*/`

1. If $(n > 0)$:
   A. `thick-Hanoi`$(n-1, A, C, B)$
   B. For i from 1 to 3:
      1. Move the top ring from A to B
   C. `thick-Hanoi`$(n-1, C, B, A)$

□

Exercise 2.1.5. In the *triple towers of Hanoi* problem, we have $3n$ rings of n distinct sizes with three copies of each ring. The goal is to distribute the rings so that each post has n rings, one of each size, in order.

Solution.

def func `triple-Hanoi`(n, A, B, C):

/ Given an integer n, a starting post, an ending post, and a spare post, distribute $3n$ rings so that each post has n rings, one of each size, in order. */*

1. If $n = 1$:
   A. Move the top ring from A to B
   B. Move the top ring from A to C
2. Else:
   A. `triple-Hanoi`($n-1$, A, B, C) *// Distribute the smaller rings*
   B. Move the top ring from A to B
   C. `triple-Hanoi`($n-1$, C, B, A) *// Free up C*
   D. Move the top ring from A to C
   E. `triple-Hanoi`($n-1$, B, A, C) *// Place the smaller rings on top*

□

Exercise 2.1.6. In the *American towers of Hanoi* problem, we have n rings with each ring colored red, white, or blue. The three posts are also colored red, white, and blue. Initially all the rings are stacked (in order of size) on the red post. The goal is to stack all the red rings on the red post, the blue rings on the blue post, and the white rings on the white post (again, in order of size).

Solution.

def func `American-Hanoi(`n`, `$start$`):`

/ Given an integer n and a starting post, distribute the n colored rings to the post of corresponding color such that each post is in order. Assume we have functions* `getColor` *and* `getPost` *such that* `getColor`*(k) will return the color of the* $n-k$*th largest ring and* `getPost(`*color*`)` *will return the post of that color.* */

1. If $n > 0$:
   A. Let $largestColor = getColor(n)$
   B. Let $target = postOfColor(largestRingColor)$
   C. If $(target = start)$ THEN
      1. `American-Hanoi`$(n-1,\ start)$
   D. Else:
      1. Let *spare* be the post that is neither *start* nor *target*
      2. `Towers-of-Hanoi`$(n-1,\ start,\ spare,\ target)$
      3. Move the top ring from *start* to *target*
      4. American-Hanoi$(n-1,\ spare)$

def func `Towers-of-Hanoi(`$k,\ A,\ B,\ C$`):`

/ Standard Towers of Hanoi helper function* */

1. If $k > 0$:
   A. `Towers-of-Hanoi`$(k-1,\ A,\ C,\ B)$
   B. Move the top ring from A to B
   C. `Towers-of-Hanoi`$(k-1,\ C,\ B,\ A)$

□

Exercise 2.2.3. Given an undirected graph $G = (V, E)$, return the maximum size of any matching.

(A *matching* is a set of edges $M \subseteq E$ with disjoint endpoints.)

Solution.

def func `max-matching`($G = (V, E)$)`:`

/ Given an undirected graph $G = (V, E)$, return the maximum size of any matching. */*

1. If $E = \emptyset$:
   A. Return 0
2. We know there is at least one edge $e \in E$, say $e := (u, v)$
3. Let $E_{\text{temp}} := \{d \in E \mid u \in d \text{ or } v \in d \text{ and } d \neq e\}$ be the set of edges sharing a vertex with e, but not including e
4. Let $G_{\text{in}} := (V \setminus \{u, v\},\ E \setminus E_{u,v})$ be the graph obtained by removing u, v and all edges incident to either vertex (besides e) from G
5. Let $count_{\text{in}} := 1 +$ `max-matching`(G_{in}) *// Branch 1: Include e*
6. Let $G_{\text{ex}} := (V,\ E \setminus \{e\})$ be the graph obtained by removing the edge e from G
7. Let $count_{\text{ex}} :=$ `max-matching`(G_{ex}) *// Branch 2: Exclude e*
8. Return $\max(count_{\text{in}}, count_{\text{ex}})$

□

Exercise 2.2.4. Given an undirected graph $G = (V, E)$, return the maximum size of any independent set of vertices.

(An *independent set* is a set of vertices $S \subseteq V$ such that no two vertices $u, v \in S$ are connected by an edge.)

Solution.

def func `max-independent-set`$(G = (V, E))$`:`

/ Given an undirected graph $G = (V, E)$, return the maximum size of any independent set of vertices. */*

1. If $|V| = 0$:
   A. Return 0
2. Else if $|V| = 1$:
   A. Return 1
3. For each vertex $v \in V$:
   A. Let G_{in} be the graph obtained by removing all edges containing v and all other vertices contained in said edges from G
   B. Let $count_{\text{in}} := 1 +$ `max-independent-set`(G_{in}) *// Branch 1: Include v*
   C. Let G_{ex} be the graph obtained by removing v from G
   D. Let $count_{\text{ex}} :=$ `max-independent-set`(G_{ex}) *// Branch 2: Exclude v*
   E. Return $\max(count_{\text{in}}, count_{\text{ex}})$

□

Exercise 2.2.5. Given a sequence of numbers, $x_1, \ldots, x_n$, return the length of the longest (strictly) increasing subsequence of $x_1, \ldots, x_n$ over all subsequences that include x_1. (You may assume $n \geq 1$.)

(A *subsequence* of $x_1, \ldots, x_n$ is a sequence of the form $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$, where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. A sequence of numbers $y_1, \ldots, y_\ell$ is strictly increasing if $y_i < y_{i+1}$ for $i = 1, \ldots, \ell - 1$.)

Solution.

def func `longest-increasing-subsequence-including-first`($x_1, \ldots, x_n$): *// LISIF for short*

/ Given a sequence of numbers, $x_1, \ldots, x_n$, return the length of the longest strictly increasing subsequence of $x_1, \ldots, x_n$ over all subsequences that include x_1. */*

1. Let $longest = 1$
2. For each index i from 2 to n:
   A. If $x_i > x_1$:
      1. Let $tail := (x_{i+1}, x_{i+2}, \ldots, x_n)$
      2. Let $current := 1 + \texttt{LISIF}(tail)$
      3. Set $longest = \max(longest, current)$
3. Return $longest$

□

Exercise 2.2.6. Given a directed graph G and two vertices $s, t \in V$, return true if s can reach t in G, and **False** otherwise.

(s can reach t if either $s = t$ or there is a sequence of (directed) edges of the form $(s, v_1), (v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, t)$. Note that the endpoint of one edge is the initial point of the next edge. This is also called a (directed) walk in G from s to t.)

Solution.

def func `reachable`$(G = (V, E),\ s,\ t)$`:`

/ Given a directed graph G and two vertices $s, t \in V$, return true if s can reach t in G, and* ***False*** *otherwise.* */*

1. If $s = t$:
   A. Return **True**

/ Random traversal through all paths accessible from vertex s* */*

2. For each edge $e = (s, v) \in E$:
   A. Return `reachable`(v, t)
3. Return **False**

□

Exercise 2.2.7. Given n integers $x_1, \ldots, x_n \in \mathbb{Z}$, returns **True** if there is a subset of indicies $S \subseteq [n]$ such that $\sum_{i \in S} x_i = \sum_{i \in [n] \setminus S} x_i$, and **False** otherwise.

Solution.

def func `partition`$(G = (V, E))$`:`

/ Given n integers $x_1, \ldots, x_n \in \mathbb{Z}$, returns **True** if there is a subset of indicies $S \subseteq [n]$ such that $\sum_{i \in S} x_i = \sum_{i \in [n] \setminus S} x_i$, and **False** otherwise. */*

1. Let $total = \sum_{i \in [n]} x_i$
2. If $total$ is odd:
   A. Return **False**
3. Else: *// ETS $\exists S \subseteq [n]$ such that $\sum_{i \in S} x_i = \frac{total}{2}$*
   A. Return `subset-sum`$(x_1, \ldots, x_n, \frac{total}{2})$
4. IDK! I GIVE UP I AM SO TIRED SORRY

□