1)

a)

b)

c)

2)

a) LinSearch(A, v, n):

for i from 0 to n-1:

if A[i] == v:

return i

return Nil

b) Loop invariant:

for all j < i, A[j] != v

Initialization:

i=0, no j < 0, statement holds.

Maintenance:

For each loop, either A[i] = v and it returns, or A[i] != v in which case i incriments and the invariant holds.

Termination:

Either A[i] = v, in which case i has been found, or the loop ends and the algorithm returns Nil, in which case v is not in A.

3)

a)

bool search(A, x, l, r):

if r-l == 0:

if A[l] == x:

return true

return false

if l-r < 2: //0 indexed, so [0,1,2]. 2-0 is 2, in which case it can be repartitioned.

if A[l] == x: return true

if A[r] == x: return true

return false

leftSplit = l + floor((r-l)/3)

rightSplit = leftSplit + ceiling((r-l)/3)

if (x < A[leftSplit]): return search(A, x, l, leftSplit)

if (x > A[rightSplit]): return search(A, x, rightSplit+1, r)

else: return search(A, leftSplit+1, rightSplit)

b)

T(1) = 1

T(2) = 1

T(n) = T(n/3) + O(n^0)

By masters therom: d=0, a=1, b=3, 1 = 3^0, T(n) = O(logn)

4)

Height:

a)

int binTreeHeight(T):

if empty(T): return -1

if child(T) == Nil: return 1

leftChild = left(T)

rightChild = right(T)

return max(binTreeHeight(leftChild), binTreeHeight(rightChild)) + 1

b)

T(0) = 1

T(1) = c

Leaf:

a)

int binTreeLeafs(T):

if empty(T): return -1

if child(T) == Nil: return 1

leftChild = left(T)

rightChild = right(T)

return binTreeLeafs(leftChild) + binTreeLeafs(rightChild)

5)

a)

int[][] createSchedule(k): //each subarray int[n] will be the matchups for player n+1. int[n][m] will be n+1’s opponent on day m+1

if k==1: return [[2][1]]

halfProblem = createSchedule(k-1) // should be a 2\*\*(k-1)x(2\*\*(k-1) - 1) 2d array.

offsetHelper = 2\*\*k-1 + 1

for i from 0 to 2\*\*(k-1) – 1:

for j from 2\*\*(k-1) -1 to 2\*\*k - 1:

halfProblem[i][j] = offsetHelper

offsetHelper++

if (offsetHelper > 2\*\*k):

offsetHelper = 2\*\*(k-1) + 1

//players 1 through 2\*\*(k-1) now have their matchups

for i from 0 to 2\*\*(k-1)-2: // copy original halfProblem, but add 2\*\*(k-1). Runs through days. Offset for zero index

for j from 0 to 2\*\*(k-1) – 1:

halfProblem[2\*\*(k-1) + j][i] = halfProblem[j][i] + 2\*\*(k-1)

// players 1 through 2\*\*k now have days 1 through 2\*\*(k-1) – 1

for i from 2\*\*(k-1) -1 to 2\*\*k - 2: //days

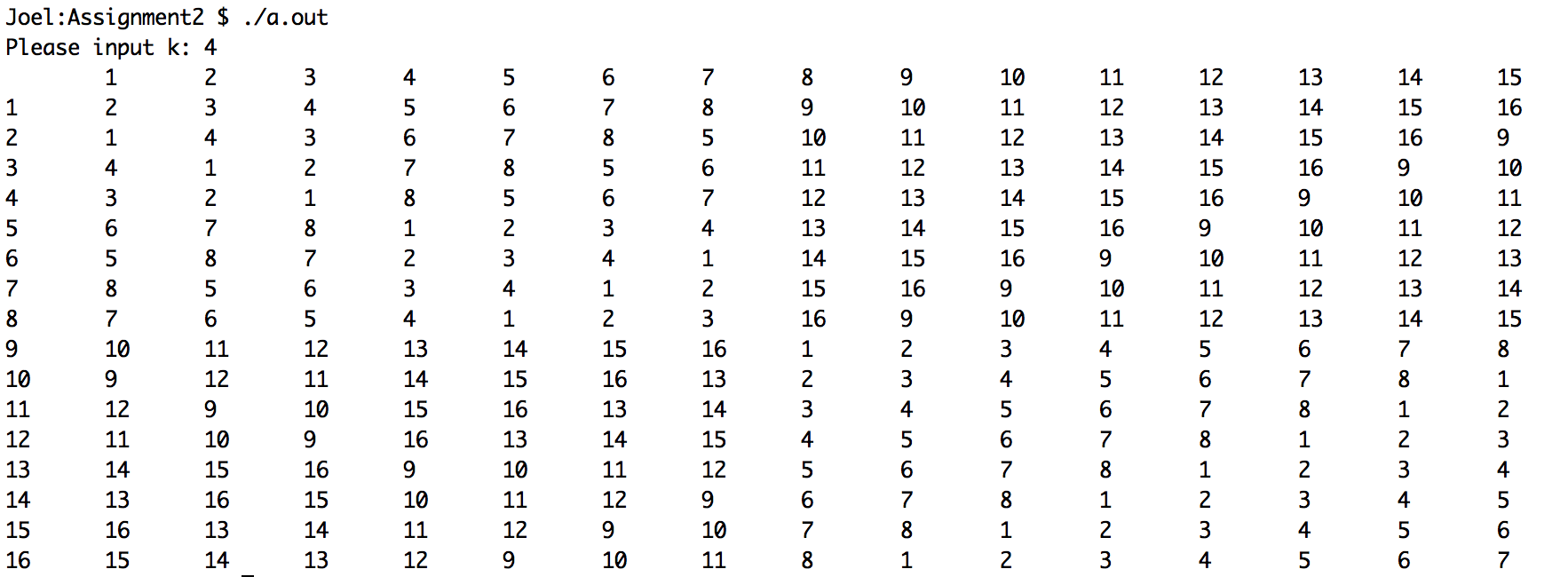
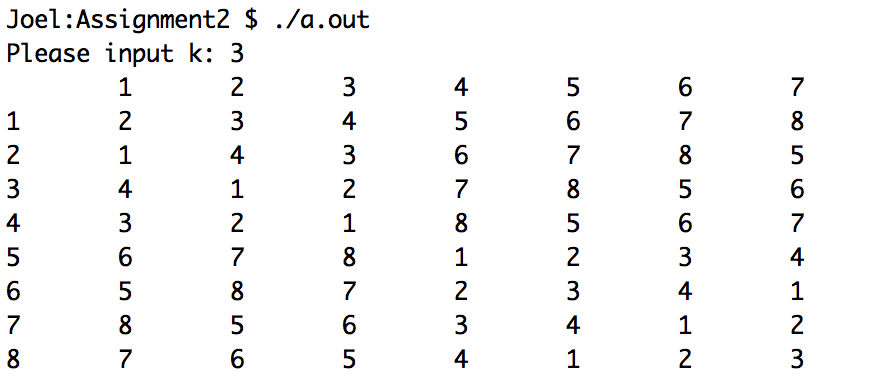
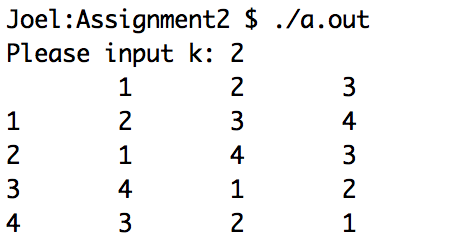
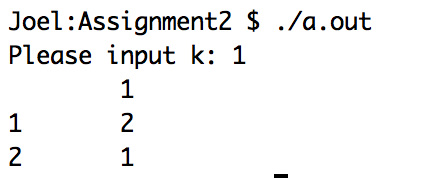
for j from 0 to 2\*\*(k-1)-1: //players

halfProblem[2\*\*(k-1) + j][i] = halfProblem[j][i] – 2\*\*(k-1)

// all players now have all assignments.

return halfProblem

PROBLEM SCREENSHOTS ON NEXT PAGE



6)

a) Code included

b) Worst case is it hits line 19 every time but the leafs, leading to 2 isHeap checks every time. So T(0) = c1; T(n) = 2T(n/2) + c2 = 2(2T(n/4)+c2) + c2 = 2^i T(n/2^i) + ic2 = 2^log2(n)c1 + log2(n)c2 = nc1 + log2(n)c2 = O(n)

7)

a)

void delElem(h, n, v): // n is the size of h

for i from 0 to n-1:

if (h[i] == v):  
 swap(h, i, n-1)

delete(h[n-1])

max-heapify(h[i]) // max-heapify is logn

break

b) Worst case is O(nlogn). n because worst case is it loops all the way to the last element before finding v, logn because it must call max-heapify after swaping and deleting.