CS 4407
Sample Mid-Term Exam, Period 2
Solutions

1. Apply the loop invariant approach to insertion-sort-reverse, which sorts a list of integers in reverse order. Use this approach to show the complexity of the algorithm.

Solution

The algorithm

```
for j <- 2 to length(A) do
// Invariant 1:  A[1..j-1] is a sorted permutation of the original A[1..j-1]
   key <- A[j]
   i <- j-1
   while (i > 0 and A[i] > key) do
// Invariant 2:  A[i .. j] are each >= key
      A[i+1] <- A[i]
      i <- i -1
   A[i+1] <- key
```

Correctness proof

We will prove the correctness of this sorting algorithm by proving that the loop invariants hold and then drawing conclusions from what this implies upon termination of the loops.

First, we note that Invariant 1 is true initially because in the first iteration of the loop j = 2 so A[1..j-1] is A[1..1] and a single element is always a sorted list.

In order to show that Invariant 1 is maintained by the loop and true during the next iteration, we must examine the body of the loop. It must be true that after the last line of the outer loop (A[i+1] <- key), A[1..j] is a sorted permutation of the original A[1..j]. We will show this is true by examining Invariant 2 and reasoning from it.

Invariant 2 is true upon initialization (the first iteration of the inner loop) because i = j-1, A[i] was explicitly tested and know to be > key, and A[j] == key.

The inner loop maintains this invariant because the statement A[i+1] <- A[i] moves a value in A[i], known to be > key, into A[i+1] which also held a value >= key. Thus this statement does not change the validity of the invariant. (If after setting i <- i-1 the invariant does not hold, the loop test of A[i] > key catches the fact and terminates the loop.)

We also note that the inner loop does not destroy any data because the first iteration copies a value over A[j], the value stored in key. As long as key is stored back into the array, we maintain the statement that A[1..j] contains the first j elements of the original list.

Upon termination of the inner loop, we know the following things about the array A:

- A[1..i] are sorted and <= key (true by default if i == 0, true because A[1..i] is sorted and A[i] <= key if i > 0)
- A[i+1 .. j] are sorted and >= key because the loop invariant held before i was decremented and the invariant said A[i .. j] >= key.
- A[i+1]==A[i+2] if the loop executed at least once and A[i+1] == key if the loop did not execute at all.

Given these facts, we see that A[i+1] <- key does not destroy any data and gives us A[1..j] is a sorted permutation of the original j elements of A.

Thus, Invariant 1 is maintained after an iteration of the loop and it remains to note that when the outer loop terminates, j = length(A) + 1 so A[1..j-1] is A[1..length], thus the entire array is sorted.

2. Consider the following **2Clique** problem:

   INPUT: An undirected graph *G* and an integer *k*.

   OUTPUT: 1 if *G* has two vertex disjoint cliques of size *k*, and 0 otherwise.

   Show that this problem is *NP*-hard. Use the fact that the clique problem in *NP*-complete. The input to the clique problem is an undirected graph *H* and an integer *j*. The output should be 1 if *H* contains a clique of size *j* and 0 otherwise. Note that a clique is a mutually adjacent collection of vertices. Two cliques are disjoint if they do not share any vertices in common.

   Solution:
   2Clique $\in$ NP: We will provide as an oracle a graph *H(V',E')*, and two *j*-cliques C_1 and C_2, which are subgraphs of H. We need to test: (i) if C_1 and C_2 are disjoint; (ii) if C_1 and C_2 are both cliques. Using an incidence-graph representation of H, both tests can be done in time polynomial in E'. Hence we have shown 2Clique is in NP.

   NP-Hardness: We will reduce CLIQUE to 2Clique as follows. Given an instance of CLIQUE [*G(V,E),k*], we create an instance of 2Clique by: adding a sub-graph H* which consists of a set of *k* vertices N, and a set of edges A such that H* is

fully-connected. There will be no edges connecting nodes in N to nodes in G. Hence we have *H(V',E')* such that $V'=V\cup N$ and $E'=E\cup A$.

$\Rightarrow$ First prove that if *G(V,E)* has a *k-clique*, then *H* has two vertex-disjoint *k-cliques.* By our construction, if *G(V,E)* has a *k-clique C*, then *H* has two *k-cliques*, *C* and *H**. By the construction, $C\cap H^* =\varnothing$.

$\Leftarrow$ Suppose that *H* has two vertex-disjoint *k-cliques*, *C* and *H**. By our construction, then if *G(V,E)* has a *k-clique C*, since the only other *k-clique* that might exist in *H* must be in the subgraph *G* of *H*.