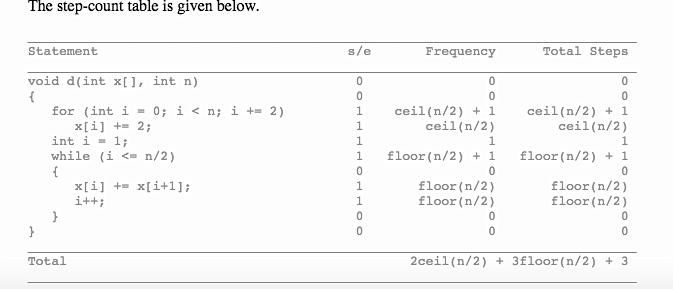
**Question 1:**

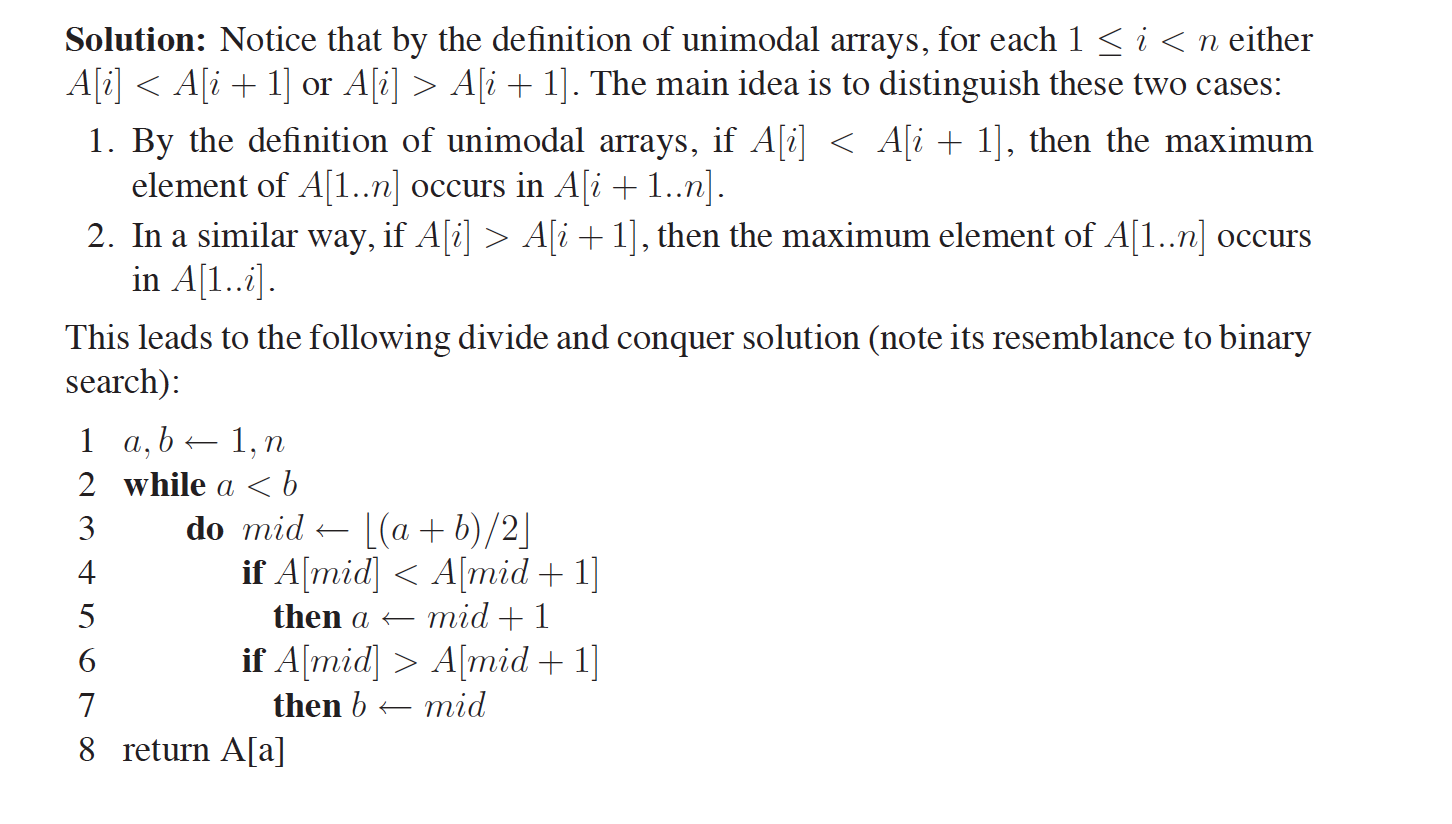


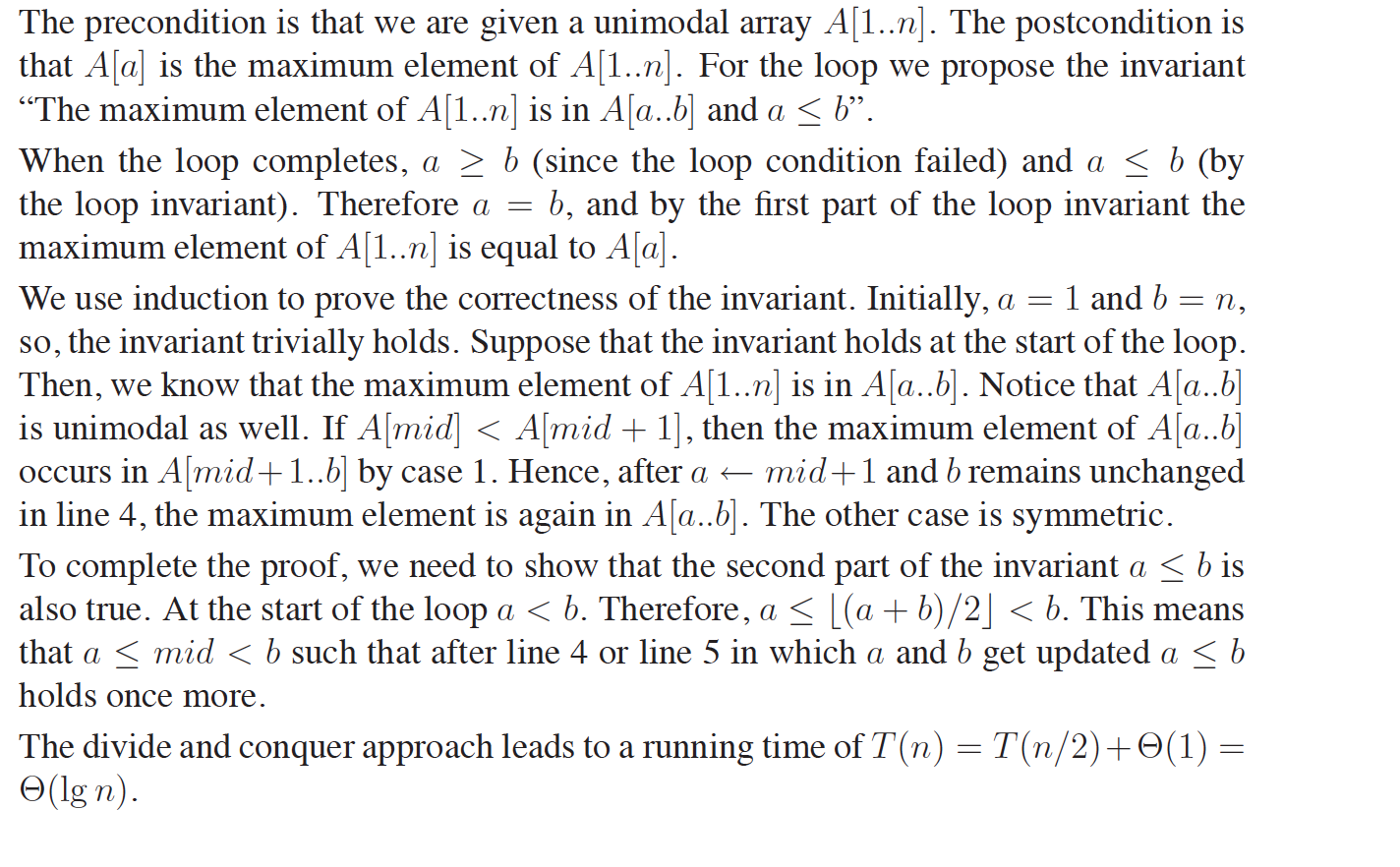
**Question 2:**

**Part 1:**

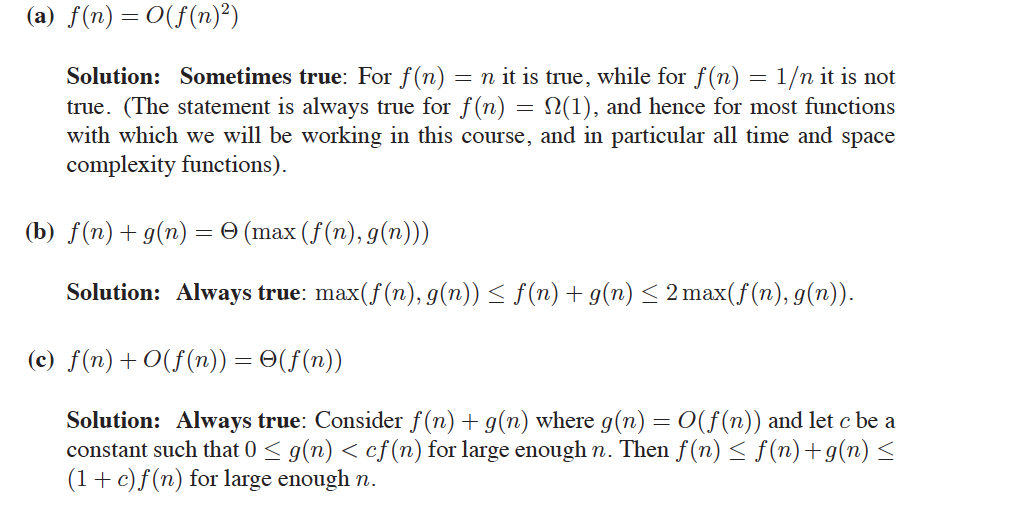
1. Let i and j be the row and column index of an element. Elements with i-j+n-1 = p define diagonal p, 0 <= p <= 2n-2 of a matrix. All elements on diagonal p of a Topelitz matrix have the same value. Therefore, there can be at most 2n-1 different values.
2. The diagonal p element is stored in position p of a one-dimensional array.

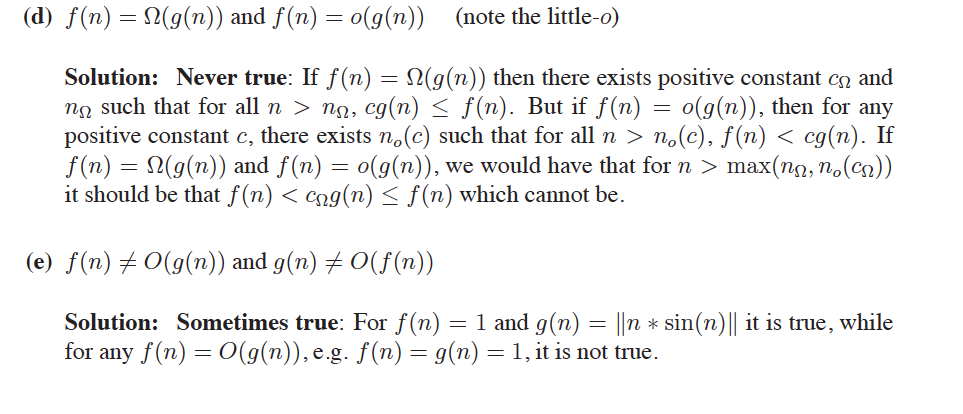
**Part 2:**





**Question 3:**





**Question 4:**

1. Following are the two ways in which shared pointers could be defined.   
     
   A. shared\_ptr<Dog> p(new Dog("Gunner”));  
   B. shared\_ptr<Dog> p = make\_shared<Dog>(“Tank”);  
     
   1a. Which one of the above two is a safe approach? Give reasons either way? [2]  
     
   (B) is a safe approach.   
   In (A) assigning the shared pointer “p” is a 2-step process (create Gunner, followed by create p). Assume a situation where new object Gunner is created successfully but the shared pointer “p” could not be created because of memory allocation failure. In such situations, Dog Gunner is not being managed by shared pointer “p”. Therefore, Gunner will not be deleted and its memory will be leaked.  
     
   1b. Which one of the above two is a fast approach? Give reasons either way? [2]  
     
   (B) is a faster approach.   
   In (A) assigning the shared pointer “p” is a 2-step process, whereas in (B) it is done in a single step.
2. Consider the following code snippet  
     
   1 void f(unique\_ptr<Dog> p) {  
   2 p->bark();  
   3 }  
   4 void test() {  
   5 unique\_ptr<Dog> pD(new Dog(“Gunner”));  
   6 f(move(pD));  
   7 }  
   8 int main () {  
   9 test();  
   10 }  
     
   In which line ‘Gunner” is destroyed? [4]  
     
   Line 6
3. What is cyclic reference in terms of memory pointers in C++? [2]  
     
     
   Shared pointers provide a shared ownership of an object and when shared pointers of that object go out of scope, the object will be deleted automatically. If a shared pointer does not go out of scope, it will not be deleted. This is what happens in cyclic-reference where two shared pointers refer to each other and none of them go out of scope and the objects to which the shared pointers were pointing will not be deleted, leading to memory leaks.