QUESTION 2: Interrupt Handling Problem

- TL is the time to perform one insert or remove operation in a linked list implementation
- TA is the time to perform one insert or remove operating in the proposed array implementation
- OH is the overhead time to temporarily extend the array
- P is the probability that any given insert operation will overrun the normal array size n.

A) Derive a formula for computing the value of P, below which the proposed scheme will outperform the linked list implementation

LINKED LIST:
TIME = TL

ARRAY:
TIME = TA + P * OH

For the second alternative to be worth it, need P > than then both times are equal
TL = TA + P * OH
P = (TL - TA)/OH

B) What is the value of P when TL = 10 * TA and OH = 100 * TA?
P = (TL - TA)/OH
P = ([10*TA] – TA)/[100*TA]
P = 9 * TA / (100 * TA)
P = 9/100

If the probability (P) that any given insert operation will overrun the normal array size n is greater than 9%, then a linked list should be implemented.

QUESTION 3: OS Data Structures

Assume at time 500 there are four processes, p1 through p4, waiting for a timeout signal. The four processes are scheduled to wake up at time 520, 645, 695, 710 respectively.

A) Assuming an implementation using a priority queue with time differences, show the queue and the contents of the countdown timer at time 500.

->p1->p2->p3->p4->NULL

B) After p1 wakes up, it immediately issues another call to timer() for 70 units of time. Assuming that processing the call takes no time, show the priority queue after the call.

P1 wakes up at 520; then gets assigned call to timer(70), so it's scheduled to wake up at 520+70 = 590.
P1 gets inserted in front of the Q
->p1->p2->p3->p4->NULL

C) Assuming p1 issues the same call again (timer(70)) immediately after it wakes up for the second time. Show the new priority queue.

P1 wakes up at 590; then gets assigned call to timer(70), so it's scheduled to wake up at 590+70 = 660.
P1 gets inserted behind p2(645) and in front of p3(695)
->p2->p1->p3->p4->NULL

QUESTION 4: Consider a banking system with many different accounts. Processes may transfer money between any two accounts, Ai and Aj, by executing the following instruction:

Lock Ai; Lock Aj; Update Ai; Update Aj; Unlock Ai; Unlock Aj;

A) Show how a deadlock can occur in such a system.

Assuming the banking system is spread out over a network (processes are ran by many CPUs). If 2 people wish to make a transfer at the same time. One process (Pa) wants to transfer from i to j, the other (Pb) wants to transfer from j to i.
The first process (Pa) locks i, and simultaneously, the Pb locks j.
Then Pa waits for j to be freed (gets in a wait Q)
Pb waits for I to bo freed (gets in a wait Q)

And they wait there until...? Deadlock.

B) How can the ordered resource policy be implemented to prevent deadlock if the set of possible accounts is not known a priori or changes dynamically?

Have a DB recording transaction (There should already be one implemented!!)
Then the instructions could be:

New entry in DB for (transaction ID, Ai, Aj, amount, completed = FALSE);
Lock Ai; Update Ai; Unlock Ai; Lock Aj; Update Aj; Unlock Aj;
Update DB (transaction ID.completed = TRUE)

Have a second system monitor the DB:

```
while(transaction ID.completed = FALSE)

if(waited)
        tell someone //something went wrong

else
        waited = TRUE;
        wait (1 minute);
```