1)

1. Compare an ordered linked list, a heap, and a timing wheel implementation of a scheduler. Fill in a table like the following with characteristics of these approaches. Assume N means the number of currently scheduled events.

Algorithm	Expected Insertion Time per event, e.g. O(N), O(log N)	Expected Dispatch Time per event, e.g. O(N) etc.	Worst-case Insertion Time	Worst-case Dispatch Time	Clarify what would cause the worst case situation(s) and/or assumptions you are making
Ordered Linked List	O(N)	O(1)	O(N)	O(1)	Worst case is when elem is inserted into the back
Heap Priority Queue	O(log N)	O(log N)	O(log N)	O(log N)	Worst case happens to an unbalanced tree that spans down like a list
Timing Wheel	O(1)	O(1)	O(N)	O(N)	When everything is hashed into the same node

2) Begin by 0.5 time units
3) At 1.0 time units
4) Because C1 finishes in the late region, meaning A1 must run immediately afterwards. However, C2 finishes early before y = real time and thus has time to wait before A2 starts.
5) Because A1 has not finished running at time 2 yet, but at 2.5.
6) Because at that point, C3 is still behind y = real time + max_delay in the dormant region, and must wait until it is in the runnable region (passing the first line) to start computing.
7) Expand the runnable region by decrease the max_delay parameter so which affects where the y = real time + max_delay line snippis. This would change the graph by decreasing the area of the runnable region by moving the y = real time + max_delay line left.
8) Problem #1: Using a thread for each task such as each drum in a drum machine having a different thread could cause race conditions.
Problem #2: Beats could be out of order due to timing problems, since the order of computation for concurrent threads is nondeterministic and difficult to predict. For example, scheduled timings depend on a global shared, flag, which means threads could execute out of order by reading outdated values of the global shared flag.
9) Use forward synchronous scheduling so that output timing can be precise even when connection has high latency to get the response timing to be accurate within 5ms. We can also delay the latency to 500ms to ensure that it is in sync, since up to 1000ms is acceptable.

10)

```
sched_select(rtsched) // prepare to use rtsched
play_sequence(0)
def play_sequence(n)
  if(n < 10) {
    play_note()
    sched_cause(0.5, nil, ‘play_sequence', n+1)
  }
```