**Question 1: (3 mark)**
Arrange in ascending order as per the increasing runtime management cost of these entities: (One line justification for each entity)

**Processes, ULTs, Tasks, Threads, and Coroutines**

**Answer:**
Tasks → Coroutines → ULTs → Threads → Processes //+0.5 marks (only if all entities are arranged correctly)
Each correct justification below carrier 0.5 marks (5 x 0.5)

a) Task is only a combination of function pointer and arguments to this function.
b) Coroutines preserve its current execution state and support context switching, but without an entire thread stack (stack required only for the current function).
c) ULTs have all the states associated with a thread, but are managed inside the user space.
d) Threads inside a process shares parent process's address space, but are managed by the kernel.
e) Processes do not share address space with each other, and are managed by the kernel.

**Question 2: (1 marks)**
Briefly justify if concurrency is the same as parallelism.
**Answer:**
Concurrency is not same as parallelism. //+0.5 marks
Concurrency is ability to divide a program execution into small chunks where these chunks supports interleaved execution. //+0.25 marks
Parallelism is when the execution of above chunks are actually interleaved, i.e., being executed in parallel. //+0.25 marks

**Question 3: (2 marks)**
Give two reasons: Why the cost of context switches performed by the OS is NOT insignificant with large number of running processes/threads.
**Answer:**

a) Context switch requires switching the execution from user space (user stack) into kernel space (kernel stack), and it happens frequently with increasing number of threads/processes. //+1 marks
b) Frequent invocation of kernel scheduler to decide which thread/process to run next. //+1 marks

**Question 4: (1 marks)**
Which is more scalable for launching 4 tasks on a 4-core processor: work-sharing or work-stealing? (No justification required) **Answer:**
Both will behave the same (+0.5 marks) as creating four tasks will not have any observable overheads unlike creating several tasks. (+0.5 marks)

**Question 5: (3 marks)**
What will be the output of the following program? (No justification required).

```
std::mutex mt;
int main() {
   boost::fibers::fiber F1 ([=]() {
      cout<< "Stage-1\n";
      boost::this_fiber::yield();
      int count=1;
      boost::fibers::fiber F2 ([&]() {
         cout<< "Stage-2\n";
          std::unique_lock<std::mutex> lk(mt);
         count+=2;
      });
      cout<< "Stage-3\n";
      std::unique_lock<std::mutex> lk(mt);
      count+=1;
      F2.join();
      cout<<"count="<<count<<"\n";
   });
   boost::fibers::fiber F3([=]() {
      cout<< "Stage-4\n";
   });
   F3.join();
   F1.join();
   cout<< "Stage-5\n";
}
```

**Answer:**
Stage-1
Stage-4
Stage-3
Stage-2
<HANG> //as F2 is trying to take a lock held by F1

Each line from Stage-1 to Stage-2 carries +0.5 marks.
Notifying a hang at correct place carries +1 marks.