Where to start your thinking

Always keep in mind: interviewers rarely expect you to invent new algorithms. They almost always test your understanding and skills to apply algorithms you've learned at school.
So, what algorithms have you learned at schools that are usually used to solve questions involving an array? DP, search/sort, divide and conquer, greedy.... Hmm... this question reminds me of the question about scheduling meetings with limited meeting rooms, which is solved by greedy algorithm. Even if you don't know the scheduling meeting question, you can swiftly attempt with DP and divide-and-conquer, and will find it is not very straight forward to define the subproblem of DB, or to find the split point of divide-and-conquer. Hmm... so greedy algorithm looks like the right one. Let's try that.

Greedy algorithm intuition

Greedy algorithms are usually very intuitive (but not necessarily correct. it requires proof). What would you do, if you have multiple equally important meetings to run, but can only make some of them? Most people probably would choose to go to the one that is going to end soon. And after that meeting, pick the next meeting from those that are still available.

Greedy algorithm proof

At some day, suppose both events `E1` and `E2` are available to choose to attend. For contradictory purpose, suppose the event `E1` that is going to end sooner is not the best choice for now. Instead, `E2` that ends later is the best choice for now. By choosing `E2` now, you come up with a schedule `S1`.

I claim that I can always construct another schedule `S2` in which we choose `E1` instead of `E2` for now, and `S2` is not worse than `S1`.
In `S1`, from now on, if `E1` is picked some time after, then I can always swap `E1` and `E2` in `S1`, so I construct a `S2` which is not worse than `S1`.
In `S1`, from now on, if `E1` is **not** picked some time after, then I can aways replace `E2` in `S1` with `E1`, so I construct a `S2` which is not worse than `S1`.

So it is always better (at least not worse) to always choose the event that ends sooner.

Greedy algorithm implementation

As we go through each days to figure out the availability of each events, it is very intuitive to first sort the `events` by the starting day of the events. Then the question is, how to find out which (still available) event ends the earliest? It seems that we need to sort the **currently available** events according to the ending day of the events. How to do that? Again, the interviewers don't expect you to invent something realy new! What data structures / algorithm have you learned that can efficiently keep track of the biggest value, while you can dynamically add and remove elements? ...... Yes! Binary search/insert and min/max heap! Obviously, heap is more efficient than binary search, because adding/removing an elements after doing binary search can potentionally cause linear time complexity.

Python code

```
import heapq
class Solution(object):
    def maxEvents(self, events):
        # sort according to start time
        events = sorted(events)
        total_days = max(event[1] for event in events)
        min_heap = []
        day, cnt, event_id = 1, 0, 0
        while day <= total_days:
                    # if no events are available to attend today, let time 
flies to the next available event.
            if event_id < len(events) and not min_heap:
                day = events[event_id][0]

                       # all events starting from today are newly available.
add them to the heap.
            while event_id < len(events) and events[event_id][0] <= day:
                heapq.heappush(min_heap, events[event_id][1])
                event_id += 1

                       # if the event at heap top already ended, then discard
it.
            while min_heap and min_heap[0] < day:
                heapq.heappop(min_heap)

                       # attend the event that will end the earliest
            if min_heap:
                heapq.heappop(min_heap)
                cnt += 1
                       elif event_id >= len(events):
                break  # no more events to attend. so stop early to save time.
            day += 1
        return cnt
```