Q 2.1)

Begin by sorting an array M in descending order in $O(n \log n)$ time. Then initializing a customers' minimum prices $stack$ and push all elements from M into it in order (so that the top of the stack is the least minimum price) in $O(n)$ time. (Storing original index positions during sorting and mapping it back at the end).

For each element in an array P, checking if current element is less than the value on the top of $stack$. If it is true then iterating to the next element without popping from $stack$ (which means the customer couldn't buy the jewelry because its price is not at least his minimum price). Otherwise, allocating that jewelry (using its index) to the customer from the top of $stack$ (using his original index that is mapped) and pop him out before moving on to the next element (similar to allocating the lowest price item the customer could buy to him). Continue until $stack$ is empty (no more customers) or have iterated through all m elements (making a jewelry could be allocated only once and a customer could only buy 1 jewelry).

We allocate each customer the jewelry with the lowest price possible that hasn't been already assigned. Then the number of allocated jewelries would be the maximum number of jewelries that could be successfully allocated.

Let an allocation of a jewelry to a customer be given by a pair (c,j), where c is the customer, and j is the jewelry. let $M(c)$ be the minimum price of the customer, and let $P(j)$ be the price of the jewelry.

Claim. The greedy strategy allocates customer c_1 the jewelry j_1, but the alternative strategy allocates the same item to the customer c_2.

Since the greedy strategy allocated customer c_1 the jewelry j_1, then $M(c_1) \leq P(j_1)$. Also, since the greedy strategy allocates each customer the jewelry with the least price but higher than his minimum price (that hasn't already been allocated), we necessarily have $P(j_1) \leq P(j_2)$, otherwise, the greedy strategy would not have allocated j_1 to c_1. If the alternative strategy allocated j_1 to a different customer, say c_2, this customer must also have a minimum price lower than the price of j_2 (since $P(j_1) \leq P(j_2)$). Hence, we can modify the allocation made by the alternative strategy to adhere to the greedy policy by swapping the allocations of the two customers.

Claim. The greedy strategy allocates customer c_1 the jewelry j_1, but the alternative strategy doesn't allocate c_1 any jewelry.

In this case, the alternative strategy must have allocated j_1 to some customer, otherwise the allocation would not be optimal, as we would be able to add the allocation (c_1, j_1). Hence, suppose that the alternative strategy allocated j_1 to a customer c_2. Since the greedy strategy considers customers in increasing order of minimum price, we necessarily have $M(c_1) \leq M(c_2)$. (If $M(c_1) \geq M(c_2)$ then the greedy strategy would have allocated c_2 a jewelry first) But since the greedy strategy allocated j_1 to c_1, $M(c_1) \leq P(j_1)$. Hence, we could modify the allocation made by the alternative strategy to adhere to the greedy policy by allocating j_1 to c_1 rather than allocating it to c_2.

Hence, we have shown that for each possible violation, a modification could be made to the allocation to make it adhere to the greedy policy. Hence, if an allocation contains multiple violations, we could apply these modifications one by one, transforming the allocation into one that would be produced by the greedy strategy. Thus, any optimal allocation could be transformed into an allocation that adheres to the greedy policy, and therefore the greedy strategy is optimal.

Overall time complexity is; $O(n\log n)+O(n)$ for sorting an array then initializing and pushing items into the stack, $O(m)$ for iterating through m length array which checking condition, peeking and popping all take $O(1)$, totaling $O(n\log n+n+m*1)\cong O(n\log n+m)$.

Q 2.2)

Begin by constructing a biograph, having first set containing n vertices for each customer and second set containing m vertices for each jewelry (by their indexes). And adding a source s and a sink t vertices into it.

For each customer's vertex, adding an edge from s to the vertex with a capacity of 1 (because 1 customer could only buy 1 jewelry), and checking for every jewelry, if the customer could buy it ($M[i]\leq P[j]\leq B[i]$) then adding an edge from the customer to the jewelry's vertex with a capacity of 1 (interpreting as the jewelry could be allocated to the customer). For each jewelry's vertex, adding an edge from itself to t with capacity of 1 (since 1 jewelry could only be allocated to 1 customer).

After that, using the Ford-Fulkerson algorithm to find the maximum flow of the graph which would provide the maximum number of jewelries that could be successfully allocated. Each chosen edge from customer's vertex to jewelry's vertex from the maximum flow is the allocation we do (allocating the jewelry to the customer using their indexes).

This allocates jewelries to as many customers possible since the maximum flow would maximize the number of customers getting a jewelry because each edge from s to each customer all have a capacity of 1, so it provides which customer getting which jewelry.

Some customers might not get a jewelry even though they are able to afford those, but it means that those jewelries are already allocated to other customers, affecting nothing on the number of customers walking away with no jewelries.

Overall time complexity is; initializing all vertices in $O(n+m+2)$ time. Iterating through n customers to add edges, and for each, iterating through m jewelries to check for conditions to add edges then adding edges for each jewelry in $O(n+nm+m)=O(nm)$. Performing Ford-Fulkerson in $O\big((n+nm+m)(n)\big)=O(n^2m)$ since the most edges we could have would happens when every customer could buy every jewelry (nm), and the largest maximum flow is when every customer get a jewelry (since $n<m$). Totaling, $O(n+m+2+nm+n^2m)=O(n^2m)$.

Q 2.3)

By using Q2.2 algorithm, we would have pairs of customers with their jewelry in $O(n^2m)$ time. But we wouldn't allocate any jewelry yet. (Maximum pairs of a customer and a jewelry possible are n pairs when all customers get a jewelry, $n < m$).

Then constructing another graph, having first set containing vertices for each day until Valentine's Day (k days), second set containing vertices for each customer who is supposed to get a jewelry and third set containing vertices for each jewelry that is supposed to be allocated from Q2.2 algorithm (using their original indexes). And adding a source s and a sink t vertices into it.

Adding an edge with a capacity of 5 from source s to each day's vertex (since only 5 orders could be processed per day) in $O(k)$ time.

Then, by iterating through an array F to check each customer's available day ($F[customer][day] = 1$), adding an edge of a capacity of 1 from each day to the customers who are available on that day. This takes $O(nk)$ iterating all elements in F. (So that if one of the available days is chosen in maximum flow, no other available days would be able to be assigned to the same customer).

Adding an edge of a capacity of 1 from a customer to a jewelry according to the pairs we got in $O(n)$ time (each customer would have a total of only 1 edge to his own jewelry since we determined who to get which jewelry using previous algorithm). This makes each customer being able to make 1 order in 1 available day only. And for each jewelry's vertex, adding an edge from itself to t with capacity of 1 in $O(n)$ time.

Now we could use the Ford-Fulkerson algorithm to find the maximum flow of the graph which would provide the maximum number of jewelries that could be successfully allocated in a given day. Each chosen edge from customer's vertex to jewelry's vertex from the maximum flow is the allocation we do (allocating the jewelry to the customer using their indexes) and assign the day with a chosen edge toward that customer to the customer.

This allocates jewelries to as many customers possible due to the same fact as Q2.2 that the maximum flow would maximize the number of customers getting a jewelry, but this time, in an available day without excessing limited order in the day because the capacity of the source to each day is limited to 5 and customers only connect with their available day in a capacity of 1 (since the maximum flow would have chosen another available day or not assign any).

Some customers might not have been assigned a day even though they have available days (no jewelries allocated if no days has assigned), but that means those days are full in order (been assigned to other customers) and he really have no other choices, affecting nothing on the number of customers walking away with no jewelries.

Overall time complexity is; $O(n^2m)$ from using Q2.2 algorithm. Initializing all vertices in $O(k + 2n + 2)$ time. Initializing all edges in $O(k + nk + 2n) = O(nk)$. Performing Ford-Fulkerson in $O(nk(n)) = O(n^2k)$ since the most edges we could have would happens when every customer is available on every day (nk), and the largest maximum flow is when every customer get a jewelry. Totaling, $O(nk + n^2k + n^2m) = O(n^2m)$ since $k < m$.