- **Algorithm**

  We sort all applications by starting time such that $b_1 \leq b_2 \leq ... \leq b_n$. Then, starting from last application a_n to earlier ones, we consider whether to choose each application or not.

  Let $f(i)$ denote the maximum total scientific value from an application a_i to the last application a_n.

  - **Case 1: not to choose** a_i. Total scientific value will stay the same as that of the time when the last choice is made, which is $f(i+1)$.
  - **Case 2: choose** a_i. Then all other applications that overlap with its time interval cannot be chosen. We must calculate from the the first application that starts after a_i. Total scientific value will be $(p_i + f(j))$.

  We compare the two cases and choose the one with higher scientific value. Therefore the maximum total scientific value

  $$f(i) = max(p_i + f(j), f(i+1)),$$

  where a_j is the first application that starts after a_i. The maximum total scientific value for the whole problem will be $f(1)$, the maximum total scientific value from first application.

- **Correctness**

  The above DP algorithm is optimal as it has the following three traits.

  - **Complete Choice**: For each application, we considered either to choose or not to choose. We do the same thing to all applications and thus they cover all possibilities of their combinations.
  - **Inductive Structure**:
    We reduce problem P into a subproblem P' with two cases.
    Because $f(j)$ and $f(i+1)$ must have been calculated when we calculate for $f(i)$, it is plausible to compute in the order $f(n)$, $f(n-1)$, ..., $f(1)$.
    Solving for $f(i-1)$ is free from external constraints because we have removed all other applications that overlap with a_i.
  - **Optimal Substructure**: For all cases, maximum total scientific value is that of the optimal solution for P' plus the scientific value of the last application if we choose it. Therefore according to standard contradiction argument, we can conclude that the DP solution to P is also optimal if we find optimal solution to P'.

- **Complexity**

  The worst-case time of the algorithm is **O(n log n)**.

  - To sort all applications by starting time requires O(n log n) time.
  - To find the first non-overlapping application a_j after a_i takes O(n log n) time using binary search. We use an array to store b_i in ascending order.
  - To compare the two choices of subproblem P' takes O(1) time.

  The overall time complexity is thus given by T (n) = O(n log n) + O(n log n) + O(1) = O(n log n).