Algorithm:

Let S be the set of n Jobs and each job J_i has a start time s_i and finish time f_i.
Let m be the number of available machines for the jobs to be scheduled upon.

```
Optimum-Schedule( {S}, m )
    {
        Sort the jobs by their finishing time.
        Initialize a list for each job, containing all the machines

        For each job in the sorted order
            {
                assign a machine that's most recently used from it's machine list
                Remove assigned machine from the lists of all overlapping jobs
            }
    }
```

Analysis:

Sorting the input will be O(nlogn**),**
Initializing the list of machines for each job takes O(mn)

For loop runs n times, as we are explicitly stating it to run for each job.
Finding the most recently used machine will be O(m)
Removing the assigned machine from all overlapping jobs will be O(n)
Overall, the for loop will be $O(mn+n^2)$ and as $n \geq m$, it'll be $O(n^2)$.

Runtime complexity is $O(n^2)$.

Design and Proof:

Let **S** be the set of all jobs that are given as input.
Let $\boldsymbol{\partial}$ be the set of jobs returned by the proposed algorithm with $|\boldsymbol{\partial}| = K$.
And $\boldsymbol{\Omega}$ be the optimal schedule with $|\boldsymbol{\Omega}| = L$

$\boldsymbol{\partial} = \{R_1, R_2, R_3, \ldots\ldots R_K\}$ & $\boldsymbol{\Omega} = \{r_1, r_2, r_3, \ldots\ldots r_L\}$
(Let's suppose these are sorted in order of finish times)

$\boldsymbol{\partial}$ U $\boldsymbol{\Omega}$ belongsTo/Subset **S**

we have to prove K = L

We will try to make arguments that the proposed greedy algorithm stays ahead.

Let $J_1, J_2, J_3, J_4, \ldots\ldots J_i \ldots\ldots J_n$ be the jobs in S sorted by their finish times.

Let $s(J_i)$ be the start time of J_i and $f(J_i)$ be the finish time of J_i.
Let $A(m_p)$ denote the next availability of machine m_p.

The idea underlying this will be to maximize the number of jobs scheduled at each step and at the same time it should also make optimal use of the machines/resources by accommodating any anticipated requests.

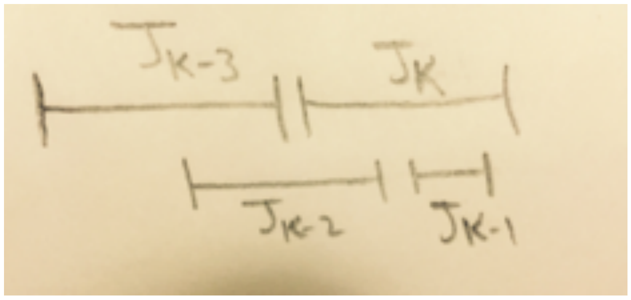
Algorithm will choose, in any schedule, the first m jobs form the sorted list(sorted by finish times) even if they overlap as we have m machines and we are greedily choosing starting from J_1. Hence, for each of i = 1 —> m, $r_i = J_i$.

Scheduling of these jobs will consume all the m machines only if $J_1 \ldots J_m$ overlap at a point on timeline.
If there exists a (there can be more) J_i with $i \leq m$ that doesn't overlaps with atleast one other Job and first of it, then this J_i has $s(J_i) \geq f(J_1)$ as Jobs are sorted by finish time.

This job J_i under consideration will be assigned to m_1 as this will be the most recently used machine. This assignment will avoid making use of an extra machine and making it available for any anticipated jobs popping up ahead in the schedule.

The greedy approach having the assignments as above will have stay ahead when scheduling the jobs first m jobs, this can be proved by contradiction.
Suppose if the **Ω** has an assignment that has better resource availability, this means each of jobs chosen by **Ω** finish early making machines available. But, this cannot be true because we are choosing all the first finishing m jobs.

By inductive hypothesis, this will be stay as above till i = K-m jobs. After K-m jobs are scheduled, $A(m_p)$ for every p = 1 —> m in **∂** is better placed than **Ω**.



the pivotal case may be to avoid assigning J_{k-1} to the same machine to which J_{k-3} is assigned to, and risking skipping J_K. As **∂** greedily chooses greedily, $A(m_p)$ for every p is well placed than the **Ω** as we are explicitly choosing most recently used machine.