1- برای اینکه اثبات کنیم زبان ذکر شده در کلاس P قرار می‌گیرد، باید نشان دهیم که در زمان چندجمله‌ای قابل حل است. فرض کنیم گراف G شامل n راس و m یال باشد، در این صورت اگر به ازای هر 4 راسی که در این گراف وجود دارد بررسی کنیم که آیا این 4 راس یک گراف دوری با اندازه 4 تشکیل می‌دهند یا خیر، می‌توانیم مسئله را حل کنیم. لازم به ذکر است که در این حالت باید بررسی کنیم که قطرهای مربع بین یال‌های گراف وجود نداشته باشند که مثلثی تشکیل نشود زیرا در این حالت زیرگراف 4 راسی یک گراف دوری نخواهد بود. به عبارتی باید الگوریتم زیر را اجرا کنیم:

```
function hasSquare(G) do
    foreach (t, u, v, w in G.V) do // O(n^4)
        if {(t, u), (u, v), (v, w), (w, t)} in G.E and {(t, v), (u, w)} not in G.E do // O(m)
            return true
        end
    end
    return false
end
```

برای محاسبه مرتبه زمانی الگوریتم ذکر شده باید به این نکته توجه کنیم که در هر مرحله 4 راس را از بین n راس گراف انتخاب می‌کنیم. در نتیجه تعداد دفعاتی که حلقه اجرا می‌شود از طریق رابطه زیر بدست می‌آید:

$$\binom{n}{4} = \frac{n!}{(n-4)! \times 4!} = \mathcal{O}(n^4)$$

از طرفی در هر بار اجرای حلقه در زمان $\mathcal{O}(m)$ می‌توانیم وجود یا عدم وجود یال‌های ذکر شده را در گراف بررسی کنیم. در نتیجه می‌توان گفت مرتبه زمانی کل الگوریتم برابر با $\mathcal{O}(mn^4)$ خواهد بود.
با توجه به موارد ذکر شده مسئله L در زمان چندجمله‌ای قابل حل است پس $L \in P$ خواهد بود.

2- الف) کلاس P:

1) بسته بودن تحت عمل *: فرض کنیم زبانی مانند L داریم که $L \in P$ است. می‌خواهیم اثبات کنیم $L * \in P$ است. با توجه به اینکه $L \in P$ است، یک decider مانند D برای زبان L وجود دارد که می‌تواند در زمان چندجمله‌ای $\mathcal{O}(n^c)$ (که n اندازه رشته ورودی و c یک عدد ثابت است)، عضویت رشته ورودی را در زبان L مشخص کند. حال می‌خواهیم یک decider برای زبان L* بسازیم. رشته ورودی این زبان را w در نظر می‌گیریم. اگر |w| = n باشد، در این صورت می‌توانیم رشته w را به صورت $x_1, x_2, \dots, x_n$ در نظر بگیریم که x_i کاراکتر i-ام رشته w باشد. حال با استفاده از برنامه‌ریزی پویا (dynamic programming)، وجود رشته w در زبان L* را بررسی می‌کنیم. در واقع باید الگوریتم زیر را اجرا کنیم:

```
function checkExistence(D, w) do
    set n = w.length
    if n == 0 do
        return true
    end
    declare dp[n][n]
    for (i = 1; i ≤ n; ++i) do
        for (j = i + 1; j ≤ n; ++j) do
            dp[i][j] = false
        end
    end
    for (i = 1; i ≤ n; ++i) do
        dp[i][i] = true
    end

    for (l = 2; l ≤ n; ++l) do // O(n)
        for (i = 1; i ≤ n - l + 1; ++i) do // O(n)
            j = i + l - 1
            if D(w[i..j]) do // O(n^c)
                dp[i][j] = true
            end
            for (k = i; k ≤ j - 1; ++k) do // O(n)
                if  dp[i][k] and dp[k + 1][j] do
                    dp[i][j] = true
                end
            end
        end
    end
    return dp[1][n]
end
```

در الگوریتم ذکر شده dp[i][j] به این معنی است که کاراکتر i-ام تا j-ام رشته w در زبان L* وجود دارد یا خیر. اگر زمان حل مسئله توسط D را در نظر نگیریم، مرتبه زمانی الگوریتم برابر با $\mathcal{O}(n^3)$ خواهد بود. همچنین اگر زمان حل مسئله توسط D را برابر با $\mathcal{O}(n^c)$ که پیش‌تر ذکر شد در نظر بگیریم، مرتبه زمانی کل الگوریتم برابر با $max\left(\mathcal{O}(n^3), \mathcal{O}(n^{c+2})\right)$ خواهد بود که چندجمله‌ای است. در نتیجه می‌توان گفت $L* \in P$ است.

2) بسته بودن تحت عمل مکمل‌گیری: فرض کنیم $L \in P$ است. در این صورت یک decider مانند D می‌تواند در زمان چندجمله‌ای وجود یا عدم وجود یک رشته در زبان L را مشخص کند. با انجام عمل مکمل‌گیری می‌خواهیم وجود یا عدم وجود رشته w را در زبان $\bar{L}$ بررسی کنیم. برای انجام این مورد یک decider به نام D’ طراحی می‌کنیم. می‌دانیم اگر رشته w در زبان L باشد در زبان $\bar{L}$ نخواهد بود و بالعکس، پس رشته w را به D می‌دهیم. می‌دانیم D با هر ورودی w در زمان چندجمله‌ای متوقف خواهد شد. اگر D رشته w را پذیرفت، D’ آن را reject می‌کند و اگر D آن را reject کرد، D’ رشته را accept می‌کند. در این حالت توانستیم یک decider برای زبان $\bar{L}$ طراحی کنیم که در زمان چندجمله‌ای متوقف می‌شود. در نتیجه می‌توان گفت $\bar{L} \in P$ است.

ب) کلاس NP:

1) بسته بودن تحت عمل *: فرض می‌کنیم که زبان $L \in NP$ است. در این صورت یک ماشین تورینگ **غیر قطعی** مانند M وجود دارد که با گرفتن ورودی w، در زمان چندجمله‌ای $\mathcal{O}(n^c)$ می‌تواند وجود یا عدم وجود رشته w در L را تشخیص دهد. حال برای زبان *L یک ماشین تورینگ غیر قطعی به نام 'M طراحی می‌کنیم. 'M رشته w را به عنوان ورودی می‌گیرد و مراحل زیر را انجام می‌دهد (فرض کنیم طول رشته w برابر با n باشد):

   I. اگر $w = \epsilon$ بود، accept می‌کند.
   II. به صورت غیر قطعی یک عدد m از 1 تا n انتخاب می‌کند.
   III. به صورت غیر قطعی رشته w را به m قطعه $w_1, w_2, \dots, w_m$ تقسیم می‌کند.
   IV. به کمک M وجود هر کدام از رشته‌های w_i را در L بررسی می‌کند.
   V. اگر به ازای تمام رشته‌های w_i، ماشین M پذیرفت، 'M هم می‌پذیرد. در غیر این صورت اگر 'M حتی یک رشته را نپذیرفت، 'M هم reject می‌کند.

   بررسی مرتبه زمانی: مرحله I در $\mathcal{O}(1)$ قابل انجام است. با توجه به اینکه عدد m حداکثر برابر با n خواهد بود، مراحل II و III به ترتیب در زمان $\mathcal{O}(n)$ و $\mathcal{O}(n^2)$ قابل انجام هستند (در مرحله III به صورت غیر قطعی m − 1 سمبل جدا کننده را در رشته اضافه می‌کنیم). همچنین اگر مرتبه زمانی ماشین M را $\mathcal{O}(n^c)$ در نظر بگیریم، این ماشین حداکثر وجود n رشته در L را بررسی می‌کند پس مرتبه زمانی مرحله IV و در واقع مرتبه زمانی اجرای کل الگوریتم به ترتیب برابر با $\mathcal{O}(n^{c+1})$ و $max\left(\mathcal{O}(n^{c+1}), \mathcal{O}(n^2)\right)$ خواهند بود که یک چندجمله‌ای است. در نتیجه می‌توان گفت $L * \in NP$ است.

2) اگر دو زبان $L_1 \in NP$ و $L_2 \in NP$ را در نظر بگیریم، ماشین تورینگ غیر قطعی M_1 می‌تواند در زمان چندجمله‌ای $\mathcal{O}(n^c)$ وجود یا عدم وجود یک رشته در L_1 را بررسی کند و ماشین تورینگ غیر قطعی M_2 نیز در زمان چندجمله‌ای $\mathcal{O}(n^k)$ وجود یا عدم وجود یک رشته در L_2 را بررسی می‌کند. حال برای زبان $L_1 \cap L_2$ یک ماشین تورینگ غیر قطعی به نام M طراحی می‌کنیم. M ورودی w را می‌گیرد و آن را به M_1 و M_2 به عنوان ورودی می‌دهد. می‌دانیم هر 2 ماشین تورینگ M_1 و M_2 در زمان چندجمله‌ای متوقف می‌شوند. اگر هر 2 ماشین ورودی را پذیرفتند، M نیز آن را accept می‌کند، در غیر این صورت ورودی را reject می‌کند. در این حالت مرتبه زمانی ماشین تورینگ M برابر با $max\left(\mathcal{O}(n^c), \mathcal{O}(n^k)\right)$ خواهد بود که چندجمله‌ای است. در نتیجه می‌توان گفت $L_1 \cap L_2 \in NP$ است.

3- ابتدا باید اثبات کنیم زبان SAT-SAT در کلاس NP قرار دارد. برای این کار یک verifier مانند V نیاز داریم که بتواند در زمان چندجمله‌ای، پاسخ را بررسی کند. فرض کنیم مسئله حل شده و برای عبارت ϕ دو پاسخ A_1 و A_2 پیدا شده است. ماشین تورینگ V ورودی را به همراه گواهی به صورت $< \phi, A_1, A_2 >$ دریافت می‌کند و ابتدا برابر نبودن A_1 و A_2 را بررسی می‌کند. سپس مقادیر A_1 و A_2 را در ϕ قرار داده و برقرار بودن عبارت منطقی ϕ را به ازای هر کدام از این ورودی‌ها بررسی می‌کند. بدیهی است که این موارد در زمان چندجمله‌ای قابل انجام است. در نتیجه می‌توان گفت زبان SAT-SAT در کلاس NP قرار می‌گیرد.

برای اثبات این مورد که این زبان در کلاس NP-Complete قرار می‌گیرد، باید یک زبان NP-Hard را به آن کاهش دهیم. برای این کار از زبان SAT استفاده می‌کنیم. فرض کنیم ماشین تورینگ M مسئله SAT-SAT را حل می‌کند. اگر ورودی مسئله SAT را یک عبارت منطقی ϕ با n متغیر $x_1, x_2, \dots, x_n$ در نظر بگیریم، برای تبدیل این ورودی به ورودی مربوط به ماشین M، یک متغیر جدید مانند y تعریف می‌کنیم. حال عبارت منطقی $\phi' = \phi \wedge (y \vee \bar{y})$ را به عنوان ورودی به ماشین M می‌دهیم. در این صورت برای عبارت منطقی ϕ حداقل یک پاسخ وجود دارد اگر و تنها اگر برای عبارت منطقی ϕ' حداقل دو پاسخ وجود داشته باشد. برای تبدیل پاسخ این 2 مسئله به همدیگر به روش زیر عمل می‌کنیم:

اگر A یک پاسخ برای مسئله SAT باشد، در این صورت (A, y = true) و (A, y = false) دو پاسخ برای مسئله SAT-SAT خواهند بود.

همچنین اگر A_1 و A_2 دو پاسخ برای مسئله SAT-SAT باشند، یکی از آن‌ها را با حذف متغیر y به عنوان پاسخ مسئله SAT در نظر می‌گیریم.

واضح است که تابع تبدیل ذکر شده در زمان چندجمله‌ای تبدیل را انجام می‌دهد.

در این صورت توانستیم مسئله SAT که در کلاس NP-Hard قرار دارد را به مسئله SAT-SAT کاهش دهیم. در نتیجه مسئله SAT-SAT هم در کلاس NP-Hard قرار می‌گیرد. از طرفی پیش‌تر اثبات کردیم که این مسئله در کلاس NP نیز قرار دارد، پس در واقع مسئله SAT-SAT در کلاس NP-Complete قرار می‌گیرد.

4- ابتدا باید اثبات کنیم که مسئله K-SET یک مسئله NP است. برای این کار باید یک verifier به نام V برای این زبان طراحی کنیم که با گرفتن مجموعه M به همراه یک گواهی، درستی یا نادرستی پاسخ را در زمان چندجمله‌ای بررسی کند. فرض کنیم مسئله حل شده است و مجموعه A به عنوان جواب بدست آمده است به طوری که $A \subset S$ باشد. در این صورت، A را همان گواهی در نظر می‌گیریم و آن را به همراه بقیه ورودی‌های مسئله و به شکل $< M, S, k, A >$ به ماشین تورینگ V می‌دهیم. این ماشین تورینگ موارد زیر را بررسی می‌کند:

1) تعداد اعضای مجموعه A حداکثر برابر با k باشد.
2) تمامی اعضای مجموعه A در مجموعه S وجود داشته باشند.
3) روی مجموعه A حرکت می‌کند و به ازای هر عضوی که می‌خواند، آن عضو را در مجموعه M علامت‌گذاری می‌کند. در نهایت باید تمام اعضای مجموعه M علامت‌گذاری شده باشند.

اگر تمام شرط‌ها رعایت شد، V ورودی را accept می‌کند و در غیر این صورت آن را reject می‌کند. بدیهی است که بررسی شرط‌های ذکر شده در زمان چندجمله‌ای قابل انجام است.

حال باید اثبات کنیم که مسئله K-SET در کلاس NP-Hard قرار می‌گیرد. برای این کار باید یکی از مسائل کلاس NP-Hard را به این مسئله کاهش دهیم. در این بخش از مسئله VERTEX-COVER استفاده می‌کنیم. فرض کنیم ماشین تورینگ T ماشینی است که می‌تواند مسئله K-SET را حل کند. ورودی‌های این ماشین تورینگ به شکل $< M, S, k >$ است که M مجموعه اصلی، S مجموعه‌ای از زیر مجموعه‌های M و k حداکثر تعداد زیر مجموعه انتخابی است. ورودی‌های مسئله VERTEX-COVER به شکل $< G, k >$ است که G گراف ورودی و k حداکثر تعداد رئوس انتخابی برای مسئله خواهد بود. برای تبدیل ورودی مسئله VERTEX-COVER به ورودی مسئله K-SET، به شکل زیر عمل می‌کنیم:

1) مجموعه M را مجموعه تمامی یال‌های گراف در نظر می‌گیریم.
2) اگر تعداد رئوس گراف را برابر با n در نظر بگیریم، مجموعه S شامل n زیر مجموعه از مجموعه M خواهد بود که آن‌ها را با $S_1, S_2, \ldots, S_n$ نشان می‌دهیم. هر یک از مجموعه‌های S_i شامل تمامی یال‌هایی است که به راس v_i متصل هستند.
3) برای مقدار k همان k ورودی مسئله VERTEX-COVER را در نظر می‌گیریم.

حال $< M, S, k >$ را به عنوان ورودی به مسئله K-SET می‌دهیم. برای مجموعه M و زیر مجموعه‌های آن در مجموعه S یک K-SET با اندازه k وجود دارد اگر و تنها اگر برای گراف G یک مجموعه با اندازه k برای مسئله VERTEX-COVER وجود داشته باشد. برای تبدیل پاسخ‌های این دو مسئله به همدیگر به روش زیر عمل می‌کنیم:

1) اگر مجموعه $S' = \{S_{u_1}, S_{u_2}, \ldots, S_{u_k}\}$ را یک پاسخ برای مسئله K-SET در نظر بگیریم، با توجه به اینکه مجموعه S' تمام یال‌های گراف را پوشش می‌دهد، هر کدام از یال‌های گراف G توسط حداقل یکی از رئوس $u_1, u_2, \ldots, u_k$ پوشش داده شده‌اند و در نتیجه می‌توان گفت رئوس $u_1, u_2, \ldots, u_k$ یک VERTEX-COVER در گراف G را تشکیل می‌دهند.
2) اگر رئوس $u_1, u_2, \ldots, u_k$ یک پاسخ برای مسئله VERTEX-COVER باشند، با توجه به اینکه این رئوس تمام یال‌های گراف را پوشش می‌دهند، می‌توان گفت $S' = \{S_{u_1}, S_{u_2}, \ldots, S_{u_k}\}$ یک پاسخ قابل قبول برای مسئله K-SET خواهد بود.

واضح است که تبدیل ورودی‌های مسئله VERTEX-COVER به ورودی‌های مسئله K-SET در زمان چندجمله‌ای $O(|V| + |E|)$ قابل انجام است. پس توانستیم مسئله VERTEX-COVER که یک مسئله NP-Hard است را در زمان چندجمله‌ای به مسئله K-SET کاهش دهیم و در نتیجه مسئله K-SET هم یک مسئله از کلاس NP-Hard خواهد بود. پیش‌تر اثبات کردیم که مسئله K-SET یک مسئله NP است، در نتیجه می‌توان گفت این مسئله در کلاس NP-Complete قرار می‌گیرد.

5- ابتدا باید اثبات کنیم که مسئله H-CLIQUE در کلاس NP قرار دارد. برای این کار یک verifier به نام V طراحی می‌کنیم که ورودی اصلی مسئله را به همراه یک گواهی از مسئله حل شده دریافت می‌کند و در زمان چندجمله‌ای صحت گواهی را بررسی می‌کند. این گواهی را رئوسی از گراف در نظر می‌گیریم که در Clique وجود داشته باشند. V ابتدا تعداد رئوس V می‌شمارد و آن را با تعداد رئوس گراف مقایسه می‌کند. اگر تعداد رئوس گراف را m در نظر بگیریم، تعداد رئوس گواهی نباید کمتر از $\frac{m}{2}$ باشد. سپس به ازای هر دو راس متمایز u و v که در گواهی قرار دارند، بررسی می‌کند که یال (u, v) در گراف وجود داشته باشد. بدیهی است که بررسی تمام این شروط در زمان چندجمله‌ای امکان‌پذیر است. پس می‌توان گفت مسئله H-CLIQUE در کلاس NP قرار دارد.

حال برای اینکه اثبات کنیم این مسئله در کلاس NP-Complete قرار می‌گیرد، باید یکی از مسائل کلاس NP-Hard را به این مسئله کاهش دهیم. برای این کار از مسئله CLIQUE استفاده می‌کنیم. فرض کنیم ماشین تورینگ M مسئله H-CLIQUE را حل می‌کند. ورودی این ماشین تورینگ گراف H است. از طرفی ورودی‌های مسئله CLIQUE را به صورت $< G, k >$ نشان می‌دهیم به طوری که k حداقل تعداد رئوس Clique است. اگر تعداد رئوس گراف G را برابر با m در نظر بگیریم، برای تشکیل گراف H سه حالت ممکن است رخ دهد:

1) $k = \frac{m}{2}$: در این صورت گراف H را دقیقا همان گراف G در نظر می‌گیریم.
2) $k > \frac{m}{2}$: در این حالت تعداد 2k – m راس با درجه 0 به گراف اضافه می‌کنیم که با این کار تعداد کل رئوس گراف برابر با 2k خواهد شد. گراف جدید همان گراف H خواهد بود ($(\mathcal{O}(m)$).
3) $k < \frac{m}{2}$: اگر این حالت رخ دهد، باید تعداد رئوس گراف و k را به طور همزمان افزایش دهیم. در این حالت t راس به گراف اضافه می‌کنیم و از این رئوس به تمام رئوس قبلی و همچنین به تمام رئوس جدید یال قرار می‌دهیم ($(\mathcal{O}(m^2)$). حال باید یک Clique با اندازه k + t در گراف جدید (H) پیدا کنیم. در نتیجه تعداد رئوس اضافه شده (t) از رابطه زیر بدست می‌آید:

$$k + t = \frac{m + t}{2} \longrightarrow t = m - 2k$$

در هر کدام از حالا ذکر شده می‌توان گفت در گراف G یک Clique با حداقل اندازه k وجود دارد اگر و تنها اگر مسئله H-CLIQUE برای گراف H دارای پاسخ باشد. برای تبدیل پاسخ‌های دو مسئله به یکدیگر، هر کدام از حالات را جداگانه بررسی می‌کنیم:

1) $k = \frac{m}{2}$: در این حالت گراف‌های H و G با هم یکسان هستند و Clique پیدا شده در هر یک از مسائل، دقیقا پاسخ مسئله دیگر است.
2) $k > \frac{m}{2}$: در این حالت هم رئوس اضافه شده در گراف H عضو Clique نخواهند بود و پاسخ پیدا شده برای هر یک از مسائل، پاسخ قابل قبولی برای مسئله دیگر نیز هست.
3) $k < \frac{m}{2}$: در این حالت ممکن است تعدادی از t راس اضافه شده در پاسخ مسئله H-CLIQUE قرار داشته باشند. اگر A یک پاسخ برای مسئله H-CLIQUE باشد، می‌دانیم $|A| \geq t + k$ است. با توجه به اینکه با اضافه کردن دقیقا t راس به گراف G، گراف H ساخته شده است، حداقل k راس از رئوس A باید از گراف قدیمی انتخاب شده باشند. پس اگر مجموعه رئوس اضافه شده را T در نظر بگیریم، B = A / T یک پاسخ برای مسئله CLIQUE است ($|B| \geq k$).
اگر B را یک پاسخ برای مسئله CLIQUE با حداقل اندازه k در نظر بگیریم، آنگاه $A = B \cup T$ یک پاسخ با حداقل اندازه t + k (نصف تعداد رئوس گراف H) برای مسئله H-CLIQUE خواهد بود.

تبدیل ذکر شده در زمان چندجمله‌ای قابل انجام است. با توجه به این مورد، توانستیم در زمان چندجمله‌ای مسئله CLIQUE که یک مسئله NP-Hard است را به مسئله H-CLIQUE کاهش دادیم. پس می‌توان گفت مسئله H-CLIQUE هم یک مسئله NP-Hard است. پیش‌تر اثبات کردیم که این مسئله در کلاس NP نیز قرار دارد. در نتیجه مسئله H-CLIQUE در کلاس NP-Complete قرار می‌گیرد.

6- برای اثبات این مورد که زبان مورد نظر در کلاس NP قرار می‌گیرد، کافی‌ست verifier مورد نظر لیست کلاس‌بندی را به عنوان گواهی به همراه ورودی‌های اصلی مسئله دریافت کند و عدم تداخل کلاس‌ها، صحیح بودن شماره دروس و کلاس‌ها و همچنین تعداد دروس و انطباق ساعت برگزاری دروس با ورودی اصلی را بررسی کند. این بررسی‌ها در زمان چندجمله‌ای امکان‌پذیر است که نشان می‌دهد این مسئله در کلاس NP قرار می‌گیرد.

در این بخش لازم است ذکر کنم که به نظرم در صورت سوال اشتباهی وجود دارد. اگر S[i] ساعت دقیق برگزاری درس i-ام باشد، با قرار دادن دروس در یک جدول و در واقع شبیه‌سازی کلاس‌بندی، مشخص می‌شود که امکان برگزاری کلاس‌ها بدون تداخل وجود دارد یا خیر. در این حالت یک درس در یک کلاس جدید برگزار می‌شود اگر و تنها اگر با تمام دروسی که در کلاس‌های قبلی در حال برگزاری هستند تداخل زمانی داشته باشد. در این حالت می‌توان گفت حداقل تعداد کلاس‌ها برای برگزاری صحیح دروس برابر با بیشترین تعداد تداخل بین دروس در یک زمان خواهد بود. در نتیجه با محدودیت k کلاس اگر در یک زمان بیشتر از k درس به صورت 2 به 2 با هم تداخل داشته باشند، امکان برگزاری دروس وجود نخواهد داشت. در غیر این صورت، می‌توانیم دروس را به طور صحیح در حداکثر k کلاس برگزار کنیم. واضح است که بررسی شرط ذکر شده در زمان چندجمله‌ای قابل انجام است که نشان می‌دهد این مسئله در کلاس P قرار می‌گیرد که با فرض NP-Complete بودن آن تناقض دارد. به همین دلیل این مسئله را به شکل یک مسئله مشابه و به صورت زیر مطرح می‌کنم:

در یک ترم n درس و k بازه زمانی برای ارائه وجود دارد. S[i] نیز نشان‌دهنده دروسی است که دانشجو i-ام اخذ کرده است. دو درس تنها به شرطی می‌توانند در یک بازه زمانی و به طور همزمان برگزار شوند که هیچ دانشجویی هر دو درس را به طور همزمان اخذ نکرده باشد (در این مسئله فرض می‌کنیم محدودیتی برای تعداد کلاس‌ها وجود ندارد). در این بخش سوال این است که آیا می‌توانیم دروس را در k بازه زمانی بدون تداخل (رعایت شرط داده شده) برگزار کنیم یا خیر. در ادامه اثبات می‌کنیم که این مسئله در کلاس NP-Complete قرار می‌گیرد.

ابتدا نشان می‌دهیم که این مسئله در کلاس NP قرار دارد. برای این مورد یک verifier به نام V طراحی می‌کنیم که برنامه زمان‌بندی دروس (اینکه هر درس در چه بازه زمانی برگزار می‌شود) را به عنوان گواهی به همراه ورودی‌های اصلی مسئله (تعداد دروس ارائه شده و تعداد بازه‌های زمانی و همچنین لیست دروس اخذ شده توسط هر دانشجو) را می‌گیرد و ابتدا بررسی می‌کند که تمامی دروس در برنامه قرار داده شده باشند، سپس معتبر بودن تمام بازه‌های زمانی مورد استفاده را بررسی می‌کند (بیشتر از k بازه استفاده نشده باشد) و همچنین تداخل نداشتن ارائه دروس برای هر دانشجو را بررسی می‌کند. بررسی شروط ذکر شده در زمان چندجمله‌ای قابل انجام است. در نتیجه می‌توان گفت این مسئله، یک مسئله NP است.

حال باید نشان دهیم که این مسئله NP-Hard نیز هست. برای این کار لازم است یک مسئله کلاس NP-Hard را به آن کاهش دهیم که در این بخش از مسئله K-Coloring استفاده می‌کنیم. فرض کنیم ماشین تورینگ M می‌تواند مسئله Scheduling را حل کند. می‌دانیم ورودی‌های این ماشین تورینگ به صورت $< n, k, S >$ خواهد بود. مسئله K-Coloring به این صورت است که یک گراف و عدد k را به صورت $< G, k >$ دریافت می‌کند و بررسی می‌کند که آیا می‌توان رئوس گراف را با حداکثر k رنگ مشخص کرد به طوری که دو سر هر یال دلخواه در گراف دارای رنگ متفاوت باشند یا خیر. برای تبدیل ورودی‌های این مسئله به ورودی‌های ماشین تورینگ M، مراحل زیر را انجام می‌دهیم:

1) هر راس را به عنوان یک درس در نظر می‌گیریم، در نتیجه تعداد دروس برابر با |V| خواهد بود.
2) هر رنگ را معادل به یک بازه زمانی در نظر می‌گیریم، در این صورت مقدار k در هر 2 مسئله با هم برابر خواهد بود.
3) به ازای هر یال یک دانشجو در نظر می‌گیریم که هر 2 درس معادل با رئوس دو سر یال را به طور همزمان اخذ کرده است.

واضح است که تبدیل ذکر شده در زمان چندجمله‌ای $O(|V| + |E|)$ قابل انجام است. مقادیر بدست آمده را به ماشین M می‌دهیم. نشان می‌دهیم که برای مسئله Scheduling پاسخی وجود دارد اگر و تنها اگر برای مسئله K-Coloring پاسخی وجود داشته باشد. با توجه به اینکه هر یال را معادل با یک دانشجو در نظر گرفتیم که هر دو درس مذکور را اخذ کرده است، این دو درس نمی‌توانند در یک بازه زمانی قرار بگیرند و در نتیجه رنگ راس‌های دو سر یک یال یکسان نخواهد بود.

برای تبدیل پاسخ‌های این دو مسئله به یکدیگر به شیوه زیر عمل می‌کنیم:

1) اگر پاسخ مسئله K-Coloring را داشته باشیم، شماره هر راس را با شماره دروس و شماره هر رنگ را با شماره بازه زمانی معادل می‌کنیم. در این صورت پاسخ برای مسئله Scheduling بدست می‌آید.
2) اگر پاسخ مسئله Scheduling را داشته باشیم، شماره هر درس را با شماره رئوس و شماره هر بازه زمانی را با شماره یک رنگ معادل می‌کنیم. در این صورت پاسخ برای مسئله K-Coloring بدست می‌آید.

با این کار توانستیم مسئله K-Coloring که یک مسئله NP-Hard است را در زمان چندجمله‌ای به مسئله Scheduling کاهش دهیم. پس می‌توان گفت مسئله Scheduling نیز NP-Hard خواهد بود. پیش‌تر اثبات کردیم که این مسئله NP نیز هست، در نتیجه می‌توان گفت مسئله Scheduling در کلاس مسائل NP-Complete قرار می‌گیرد.