**Analysis of Algorithms**

Spring 2020

**Members Details**

| Group ID | CS311S20PID30 |
| --- | --- |
| Registration Number of Group Members | 2018-CS-22  2018-CS-33 |
| Section | A |

**Project Details**

|  |  |
| --- | --- |
| ***Project*** |  |
| Project Title | Activity Scheduling Tool |
| Executive Summary | Our project is an Activity Scheduling Tool. It is a desktop application that generates the timetable automatically after providing it with certain information. The input given to make timetable will be the list of teachers, classroom in which class will be conducted, contact hour list for each subject in a week, list of courses of a session, and the user has also to tell if it is a lab or theory subject. We are using an “Activity Scheduling” algorithm to make an automatic timetable. Our algorithm will select the activity to be scheduled and will place it in the available slot keeping in the view the subject's contact hours and will manage timetable accordingly. We have an option to choose either a web or desktop application for our project. We made a Desktop application for our project. We worked in .Net Framework with C# (C Sharp) language for the frontend and used SQL server 2014 Management Studio for database and backend. In SQL Server 2014 Management Studio we used DDL (Data Definition Language) and DML (Data Manipulation Language). In the activity scheduling tool, the program generates a timetable depending upon the number of contact hours of the subject. It arranges the subjects such that it takes minimum days of the week and cover all subjects according to its contact hours. It has 5 working days; Monday, Tuesday, Wednesday, Thursday, and Friday. The algorithm checks where slots are available. It also checks the contact hour of the subject and keeping in the view all its dependencies it allocates the lecture slot to the subject. The room will be selected by the user itself. After generating the timetable the user can log in again and can see its generated timetable whenever he/she wants. For this, the user needs to remember the userID he entered while generating the timetable. |
| ***Business Case*** |  |
| Outline the business need for the project | It is an automatic timetable generator and as the name shows it generates timetable. It saves time and manually generation of the timetable. It lessens the workload, time, and is very useful for the users. |
| End-user of the product | Teachers and students are the end-users of the project. |
| Motivation for Project | The timetable generator was a different project than the other two that were file compression and plagiarism checker. These both were boring projects than the timetable one in our view. Moreover, in the timetable generator, we just need data from the user and we will present data to the user. It means that there is not much use of file handling.  It is a hectic task to create a timetable manually and is a time-consuming task. So, an automatic time table generator will be very helpful for the generation of timetable and will be time-saving. |
| Description of the project objective(s) | It takes room number, course Id, semester number, and list of subjects with credit hours and subject type for the subject and generates a timetable according to the input data. |
| State the level of impact expected should the project proceed and implications of not proceeding | Our application is allowing users to sign up as a user and is allowing to login. Users can add data and the timetable is generated. Users can view the time table. We wanted that user can also download the timetable as an excel file but we could not implement that because we already did a lot of effort and we could not understand it completely and it was time taking so that we could not implement it in our project. |
| Functional Requirements | First of all, the user has to login in our application, if the user has not signed up, he/she can sign up  After login, the user can add data and can view time  In Add data, course id, semester number, room number, and subject detail are added by the user, and when the user clicks done it generates timetable.  And users can view a timetable in future by using their user id. |
| ***Benefits*** |  |
| What benefits are expected/ anticipated? | Using this program the user can schedule all the activities in minimum days. It means that maximum activities will be arranged in early weekdays. |
| ***Implementation Details*** |  |
| Link to Github Repository | https://github.com/ayeshatahirme/CS311S20PID30.git |
| Total Number of commits in repository before 8th December 2019 | No commits before 8th December. We started committing after 23rd June. |
| Exact contribution of each member | **2018-CS-22**   * Drew interfaces fro our project on paper in Milestone 1. * Wrote the algorithm by myself. * Wrote correctness and complexity analysis of the algorithm. * Created all the interfaces from the signup page to the timetable generator of the frontend of the project. * Implemented algorithm in C# language. * Created all database tables for storing data from the user table to the generated timetable. * Integrated database with the project. (Made connections with database) * Added all the functionality to enter data, page connections, algorithm implementation except log in and signup in the project. * Stored entered data from user input text boxes to the respective database tables. * Implemented my own created algorithm in the project. * Wrote the whole project configuration document. * Did full report writing except the business case part. * Pushed most of the files on GitHub repo. * Maintained indentation and styling of each file. * Did full project except pseudoCode and the cost of the algorithm and login signup page functionality implementation.   **2018-CS-33**   * Wrote about algorithm selection in Milestone 1 * Wrote pseudoCode and calculated the cost of the algorithm. * Added log in and sign up functionality in project. * Added business case in the report. |
| ***Commits in github repository by each member*** | |
| |  |  | | --- | --- | | **Member Registration No.** | **Total Commits** | | 2018-CS-22 | 80 | | 2018-CS-33 | 11 | | |
| **Details of commits** | |
| |  |  |  |  | | --- | --- | --- | --- | | **Sr. No.** | **Details of commit** | **Date** | **Member Reg No.** | | 1 | Initial commit (made README.md file) | 23 June | 2018-CS-22 | | 2 | Added PseudoCode file | 4 July | 2018-CS-22 | | 3 | Updated pseudocode file | 4 July | 2018-CS-33 | | 4 | Added PseudoCode.md file | 4 July | 2018-CS-22 | | 5 | Uploaded time complexity analysis file | 8 July | 2018-CS-33 | | 6 | Added correctness of algorithm | 8 July | 2018-CS-22 | | 7 | Final changes in file of time complexity and correctness | 8 July | 2018-CS-22 | | 8 | Uploaded algorithm file | 13 July | 2018-CS-33 | | 9 | Made changes in file | 13 July | 2018-CS-22 | | 10 | Added visual studio files with created interfaces | 16 July | 2018-CS-22 | | 11 | Added login and signup functionalities in vs files | 16 July | 2018-CS-33 | | 12 | Added code to store user input data to the database | 17 July | 2018-CS-22 | | 13 | Implemented grid view property | 19 July | 2018-CS-22 | | 14 | Made changes in SQL and generator.cs files | 19 July | 2018-CS-22 | | 15 | Implementing timetable generator functionality | 19 July | 2018-CS-22 | | 16 | Completed algorithm implementation | 19 July | 2018-CS-22 | | 17 | Timetable generated | 19 July | 2018-CS-22 | | 18 | Merging and modifying login and signup functionality | 19 July | 2018-CS-22 | | 19 | Made changes in the program | 20 July | 2018-CS-22 | | 20 | Made changes in the database | 20 July | 2018-CS-22 | | 21 | Updated subjectData | 20 July | 2018-CS-22 | | 22 | Final commits | 20 July | 2018-CS-22 | | 23 | Added comments in code | 20 July | 2018-CS-22 | | 24 | Made little changes | 25 July | 2018-CS-22 | | 25 | Added comprehensive comments in code | 29 July | 2018-CS-22 | | 26 | Added more descriptive comments in vs code | 30 July | 2018-CS-22 | | 27 | Making Change Request | 4 August | 2018-CS-22 | | 28 | Adding markdown file | 4 August | 2018-CS-22 | | 29 | Report writing | 5 August | 2018-CS-22 | | 30 | Added files via upload | 5 August | 2018-CS-33 | | 31 | Added project configuration markdown file | 5 August | 2018-CS-22 | | 32 | Made final changes in project configuration markdown file | 5 August | 2018-CS-22 | | 33 | Added business case in report | 5 August | 2018-CS-22 | | 34 | Made final report | 5 August | 2018-CS-22 | | |
| Have you used built in algorithms or you have implemented yourself? | We have not used any built-in algorithm. Instead, we implemented the algorithm created by ourselves to generate the timetable. The algorithm keeps in check the free slots and arranges lectures accordingly. |
| Formats of input | The user will input data in text form. He will enter the required details for the timetable in the application. There is no file handling in this project. The data will be entered in the respective text boxes and will be stored in database tables. |
| Validations | * The user should only enter positive integer values wherever numbers are to be entered like in userID, contact hour text boxes, etc. * The username once used can not be used again to register in the application because it is a unique value. * The userID once used can not be used again to create a timetable because it is a unique value. * The userID used to enter data must be remembered as the user may need it in the future to view his generated timetable by entering the userID. |
| Format of output | The timetable is generated and is presented in the grid view. The user can view the generated timetable whenever he wants to see using the userID. The output is shown on the design page, generator.cs. After adding the userID, the program matches userID in the database and fetch respective records and displays them in the grid view. So, the output is shown in the grid view. |
| Deployment | No, we have not deployed our project. |
| ***Details of algorithms*** | |
| **PseudoCode with calculated Complexity:**  //Maximum number of subjects a class can have  Maximum **M\_SUBJECT** 8  //Data structure to store details about subject  Structure **SUBJECT**  {  SUBNAME  CHRS  SUBTYPE  }  As it is declarative statement so it will take time  **T(n) = θ(1)**  // Data structure to store number of lectures in a day  Structure **TIMETABLE**  {  T\_ID  LEC1  LEC2  LEC3  LEC4  TBREAK  LEC5  LEC6  LEC7  }  As it is declarative statement so it will take time  **T(n) = θ(1)**  // Function to allocate slot  **COST**  **SLOT\_ALLOCATION (SUBJECT S[ ], TIMETABLE T[ ] , M\_SUBJECT)** θ( n )  Let **WEEKDAY** = 1 θ(1)  For i=1 to M\_SUBJECT θ( n )  If ( i=1 | | i =3 || i =5 || i =7 ) θ( n - 1)  if (S[ i ].SUBTYPE == "Lab" || S[ i ].SUBTYPE == "lab" || S[ i ].SUBTYPE == "l") θ( n - 1)  T[ WEEKDAY ].LEC1=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC2=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC3=S[ i ].SUBNAME θ( n - 1)  else if(S[ i ].SUBTYPE =="Theory" || S[ i ].SUBTYPE == "theory" || S[ i ].SUBTYPE == "th")  θ( n - 1)  if(S[ i ].CHRS=="1") θ( n - 1)  T[ WEEKDAY ].LEC1=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC2= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC3= “ – ” θ( n - 1)  else if(S[ i ].CHRS =="2") θ( n - 1)  T[ WEEKDAY ].LEC1=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC2=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC3= “ – ” θ( n - 1)  else if(S[ i ].CHRS =="3") θ( n - 1)  T[ WEEKDAY ].LEC1=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC2=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC3=S[ i ].SUBNAME θ( n - 1)  else θ( n - 1)  T[ WEEKDAY ].LEC1= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC2= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC3= “ – ” θ( n - 1)  If ( i =2 || i =4 || i =6 || i =8) θ( n - 1)  if (S[ i ].SUBTYPE == "Lab" || S[ i ].SUBTYPE == "lab" || S[ i ].SUBTYPE == "l") θ( n - 1)  T[ WEEKDAY ].LEC4= “ – ” θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC6=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC7=S[ i ].SUBNAME θ( n - 1)  else if((S[ i ].SUBTYPE == "Theory" || S[ i ].SUBTYPE == "theory" || S[ i ].SUBTYPE == "th") &&  (S[ i – 1 ].SUBTYPE == "Theory" || S[ i – 1 ].SUBTYPE == "theory" || S[ i – 1 ].SUBTYPE == "th") &&  (S[ i - 1 ].CHRS || S[ i – 1 ].CHRS == "2" || S[ i – 1 ].CHRS == "3")) θ( n - 1)  if (S[ i ].CHRS == "1") θ( n - 1)  T[ WEEKDAY ].LEC4=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC6= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  else if (S[ i ].CHRS == "2") θ( n - 1)  T[ WEEKDAY ].LEC4=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC6= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  else if (S[ i ].CHRS == "3") θ( n - 1)  T[ WEEKDAY ].LEC4=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC6= S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  else if ((S[ i ].SUBTYPE == "Theory" || S[ i ].SUBTYPE == "theory" || S[ i ].SUBTYPE == "th") &&  (S[ i - 1 ].SUBTYPE == "Lab" || S[ i – 1 ].SUBTYPE == "lab" || S[ i – 1 ].SUBTYPE == "l"))  θ( n - 1)  if (S[ i ].CHRS == "1") θ( n - 1)  T[ WEEKDAY ].LEC4=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC6= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  else if (S[ i ].CHRS == "2") θ( n - 1)  T[ WEEKDAY ].LEC4=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC6= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  else if (S[ i ].CHRS == "3") θ( n - 1)  T[ WEEKDAY ].LEC4=S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC6= S[ i ].SUBNAME θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  Else θ( n - 1)  T[ WEEKDAY ].LEC4=“ – ” θ( n - 1)  T[ WEEKDAY ].TBREAK = “BREAK” θ( n - 1)  T[ WEEKDAY ].LEC5= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC6= “ – ” θ( n - 1)  T[ WEEKDAY ].LEC7= “ – ” θ( n - 1)  WEEKDAY = WEEKDAY+ 1 θ( n - 1)  As we ignore the smaller terms so the time complexity is as,  **T(n) = θ(n)**  **Algorithm description:**  We are using our own generated algorithm. In this algorithm, the user inputs room number, course code, semester, subject names, their contact hours, and their type that is if they are theory or lab subjects. Our algorithm will check if the first subject entered is of how many contact hours and if it is a lab or theory subject. Then it will allot it slots according to its contact hours. Similarly, it will check the next subject and will place it on the available slot. After four lectures there will be a break from 12 pm to 1 pm. In the same way, the algorithm will keep checking if the entered subject is theory or lab. If it will be lab a three-hour slot will be allocated to it or if it is a theory, it will be allotted slots according to its contact hours.  The algorithm will try its best to keep maximum activities in early weekdays so that activities take minimum days.  **The complexity of Algorithm:**  **SLOT\_ALLOCATION** is our main function and it uses SUBJECT and TIMETABLE data structure. Moreover, the subject type is the prime part of the subject and it decides the slot allocation of the subject in the table which is crucial for the whole algorithm and our algorithm takes **θ(n).** Our algorithm is a linear time algorithm.  The time complexity of function; SLOT\_ALLOCATION is Ѳ(n). The declarative statements will take Ѳ(1) time. So, the overall time complexity calculated is Ѳ(n). It means our algorithm has **θ(n)** time complexity.The algorithm is a linear time algorithm. The time complexity of our algorithm is efficient and it seems to be an optimized algorithm. Its time complexity is less. It seems an efficient and good solution. Thus our algorithm is a good one that works efficiently.  **The correctness of the Algorithm:**  If zero activities are added to the algorithm it will generate an optimal solution. After adding desired values in the required files, the algorithm will generate a timetable that will deal with subject contact hours and will create a timetable that will manage all subjects in minimum days.  If we generate a k timetable with input values that are the best solution, then the next one k+1 generated will also be the best one.  The time complexity of function; SLOT\_ALLOCATION is Ѳ(n). The declarative statements will take Ѳ(1) time. So, the overall time complexity calculated is Ѳ(n). This is a linear time complexity. It seems an efficient and good solution. Thus our algorithm is a good one that works efficiently. | |
| ***Interfaces for your project*** | |
| 1. The first page of our application will ask if the user wants to signup or he is already registered. If the user is already registered he will log in to the program and if he is not registered then the user will make his account by signing up and then he can access further functionalities.      1. If the user is not registered he will click the signup button and this page will appear. Here he has to enter some required data that is his first name, last name, username, his email address, and at last the password. The password entered in both text boxes; password and confirm password are matched. If they are not the same then the message is displayed that passwords did not match. And if they are same the user is registered. After signing up the user is good to use the program.      1. If the user is already registered. He will enter his username and correct password to log in to the program. In case of the wrong password, he will be notified that the password he entered is wrong.      1. After logging in/signing up, this page will appear asking the user if he wants to generate a timetable by adding data or if he wants to view the already generated timetable.      1. If the user clicks on the “Add Data” button, he will have to enter userId that is very important as it will be used to access the timetable. The user will enter userID to display a timetable whenever he wants to view the timetable even in the future he can access the generated timetable on this userID. Further, the user will input the code of the course, semester, and room number in which the classes will be held. Moreover, on clicking the “Subjects” button, a page will appear where the user will input subjects and its details.      1. On clicking on the Subject button, this page will appear. Here the user will input the userID so that respective data is stored with this userID. The user will input subject names, contact hours, and either it is a theory or lab subject. Moreover, for recognition, if the subject is of theory or lab, the user must enter L or T for lab and theory with the subject name.      1. At last, the timetable is generated, it shows all necessary data that is the course code, semester, room number, and the generated timetable. There is a logout button that will logout your account and will take you back on the login page. Done button means you are all done and want to switch off. You can access this generated timetable whenever you want by clicking on the “Show Timetable” button and entering the userID of the respective timetable. If the userID is valid the timetable will be displayed. | |
| ***Integration*** | |
| While the integration of User Interface and algorithm, I was confused at a point that how will I show data from different tables in different cells of grid view after performing algorithm functionalities. I searched a lot to find a way to show data from different tables in different cells of the same grid view but I could not have enough understanding to implement it. At last, I decided to make a separate table in the database for storing data of the generated timetable and show it at once in the grid view.  Another problem that I faced was that I could not add weekdays in the grid view along with a timetable. If I add weekdays in the first column, then the timetable was not displaying in the grid view. So, I had to skip the weekdays from the grid view and wrote them on the left side of the grid view so that timetable can be displayed.  In this way, I cope up with my problems. | |
| ***Change Requests*** | |
| We made only one change that was of our algorithm. We first decided to work in the Activity scheduling algorithm and submitted it in the “decision of algorithm” on its respective deadline. We could not implement our algorithm according to what we chose. Then we decided to change our algorithm to the Genetic Algorithm and tried to understand it. We submitted the pseudo-code of the algorithm but its cost was too much that we had to make many changes in the algorithm and finally we implemented our algorithm. It is the like a greedy choice algorithm. It has a linear cost. It means it is an efficient one. Its cost is θ(n) and it is a good and efficient algorithm. The final pseudo code along with its time complexity and correctness is added in a folder named “Change Request” on GitHub. You can check it on our Github repository. | |
| ***Testing*** | |
| *In this section, you are required to mention the issues report and solution proposed.* | |
| ***Technology*** |  |
| Programming Language | C Sharp (C#) |
| Platform | Desktop Application |