|  |  |
| --- | --- |
|  | **[Design & Analysis of Algorithm]**  **[BSCS – 5 A]**  **Department of Computer Science**  **Bahria University, Lahore Campus** |

**Assignment: 4**

Name: \_AFFAN AHMAD\_\_ Roll No: \_03-134221-003\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| **Evaluation of CLO** | **Question Number** | **Marks** | **Obtained Marks** |
| **CLO statement**   * **CLO 2:** **Analyze the time and space complexity of different algorithms.** * **CLO 3: Design algorithms to solve simple computational problems and compare the implementations empirically.** | 1  2 | 0.5  4.5 |  |
| **Total Marks** | | **5** |  |

**Part1: CLO 2: Analysis**

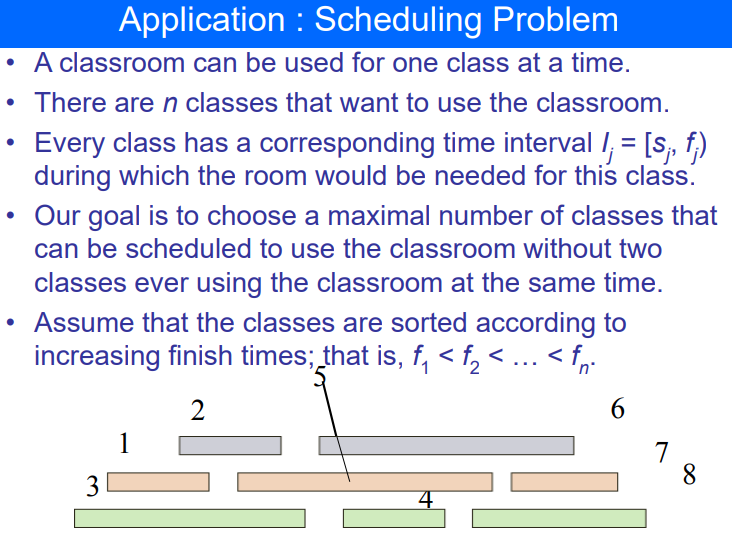
**Reading Task: Explore and briefly discuss time & space complexity of AVL Trees and Red-Black Trees from following sources, you can also use other resources to study. [0.5]**

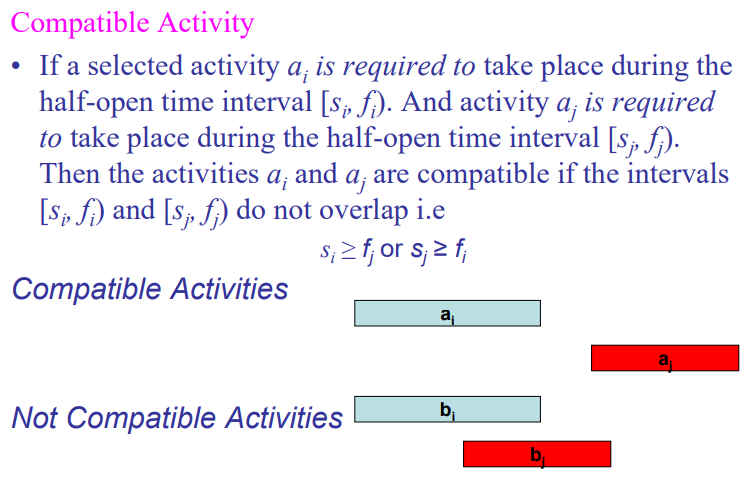
1. <https://iq.opengenus.org/time-and-space-complexity-of-red-black-tree/>
2. <https://iq.opengenus.org/time-complexity-of-avl-tree/>

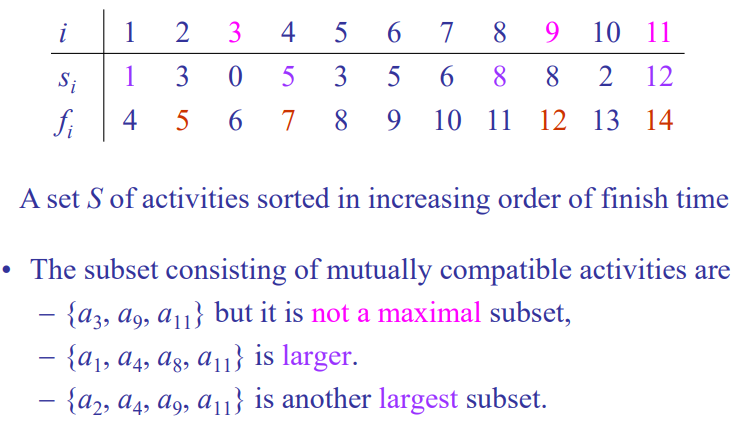
**Part 2: CLO 3: Design algorithms [4.5]**

The problem involves scheduling of several competing activities that require exclusive use of common resource Problem Statement. Suppose we have a set: ***S = {a1 , a2 , ..., an }*** of ***n*** proposed activities. Each activity wish to use a resource which can be used by only one activity at a time. Each activity ***ai*** has starting time ***si*** , finishing time ***fi*** where, 0 ≤ si < f i < ∞

Objective in activity-selection problem is to select a maximum-size subset of mutually compatible activities. Its application is explained through scheduling example given below:

****

****

****

**The activity selection problem is best solved using a greedy algorithm. This approach is optimal for this specific problem due to its nature and constraints.**

**Greedy Algorithm for Activity Selection**

The core idea is to always pick the next activity that finishes the earliest and is compatible with the previously selected activities. This ensures that we leave as much room as possible for future activities.

**Steps of the Greedy Algorithm**

**Sort Activities**: Sort the given activities by their finish times.

**Select Activities**: Initialize the first activity as selected. For each subsequent activity, if its start time is greater than or equal to the finish time of the last selected activity, select it.

**Example Pseudocode:**

Function activitySelection(activities):

Sort activities by their finish times

selectedActivities = []

lastFinishTime = -∞

for each activity in activities:

if activity.start >= lastFinishTime:

selectedActivities.append(activity)

lastFinishTime = activity.finish

return selectedActivities

**Detailed Explanation of the Pseudocode**

**Sort:** Sort the activities based on their finish times.

Initialization: Initialize an empty list selectedActivities to store the selected activities and a variable lastFinishTime to track the finish time of the last selected activity.

**Selection Loop:** Iterate through each activity. If the start time of the current activity is greater than or equal to lastFinishTime, select the activity and update lastFinishTime to the finish time of the current activity.

**Return Result:** The list selectedActivities now contains the maximum number of non-overlapping activities.

**Example in C++**

**#include <iostream>**

**using namespace std;**

**#include <vector>**

**#include <algorithm>**

**struct Activity {**

**int start;**

**int finish;**

**};**

**bool activityCompare(Activity a1, Activity a2) {**

**return a1.finish < a2.finish;**

**}**

**vector<Activity> activitySelection(vector<Activity>& activities) {**

**sort(activities.begin(), activities.end(), activityCompare);**

**vector<Activity> selectedActivities;**

**int lastFinishTime = -1;**

**for (const auto& activity : activities) {**

**if (activity.start >= lastFinishTime) {**

**selectedActivities.push\_back(activity);**

**lastFinishTime = activity.finish;**

**}**

**}**

**return selectedActivities;**

**}**

**int main() {**

**vector<Activity> activities = {**

**{1, 4}, {3, 5}, {0, 6}, {5, 7}, {3, 8}, {5, 9},{6,10},{8,11},{8,12},{2,13},{2,14}**

**};**

**std::vector<Activity> selectedActivities = activitySelection(activities);**

**cout << "Selected activities are:" << endl;**

**for (const auto& activity : selectedActivities) {**

**cout << "Activity with start time: " << activity.start**

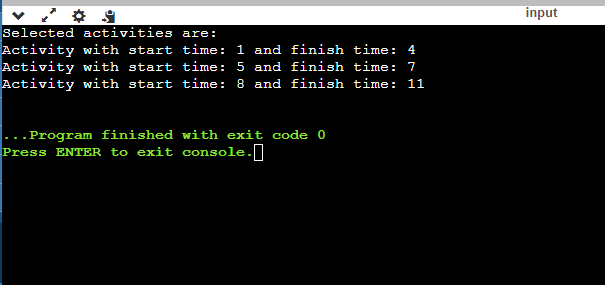
**<< " and finish time: " << activity.finish << endl;**

**}**

**return 0;**

**}**

**Output:**



**Explanation:**

**Activity Structure:** Represents an activity with start and finish times.

**activityCompare:** Comparator to sort activities by their finish times.

**activitySelection Function:** Implements the greedy algorithm to select the maximum number of non-overlapping activities.

**Main Function:** Defines a list of activities, calls the selection function, and prints the selected activities.