# Data Structures and Algorithms Final Project: Event Scheduler and Calendar Data Structures, Algorithms, and Complexity Analysis

**Innocent Farai Chikwanda**
Ashesi University
innocent.chikwanda@ashesi.edu.gh

**Tendai Terrence Machaya**
Ashesi University
terrence.machaya@ashesi.edu.

**Jim-clifford Edward**
Ashesi University
jim.edward@ashesi.edu.gh

**Doron Uyi Pela**
Ashesi University
doron.pela@aucampus.onmicrosoft.com

## Background

While not as awe-inspiring as the groundbreaking software systems of today Calendars and Event planners are one of the most important tools that have been created by humanity. Their utility never diminishes as they help people manage the scarcest and most precious resource of all: time. Almost every phone, laptop, and tablet has an application to perform this functionality. We devoted this project to build our own version of a Calendar and Event Planning Software and in this paper we discuss the computational considerations,i.e. data structures and algorithms and their resultant time complexity, necessary to build an efficient piece software.

## Data Structures

To implement the Calendar we decided to use an object-oriented and modular approach. We divided the project into four components separated by abstraction barriers. The sections were as follows:

- The event
- The day
- The calendar
- The graphic user interface

Each of these components were separated by a logical abstraction that made it possible for each group member to develop the a part of the software by just knowing the methods of the other portions of the system but without knowing the underlying implementation.

## The Event

An event was a compound object that could be instantiated. It had several useful attributes like title, date, startTime, endTime, Location, recurrence, and reminder. The event also had several methods to modify its attributes such as setReminder, setLocation, getLocation(), getEndTimeOfEvent() to name a few. All methods in the event class executed simple mathematical operations that can be considered as constant time or O(1).

Below is a demonstration using the most complex part of the class. We place a comments about the time complexity in square brackets beside each line of code:

```
public Event(String title, String description, int
year, int month, int day, int hour, int minutes,
    frequency recurring)

 {
   this.title = title; [                          [1 unit]
   this.description = description;               [1 unit]

   this.date = LocalDate.of(year, month, day);
                                                 [2 unit]
```

```
  this.startTimeOfEvent = LocalTime.of(hour,
minutes);                                      [2 unit]
  this.reminderTime =
startTimeOfEvent.minusMinutes(60);
                                               [2 unit]
  if (reminderTime.getHour() < 23) {
    this.reminderDate = date;                  [2 unit]
  } else
    this.reminderDate = date.minusDays(1);
                                               [3 unit]

  this.hasReminder = true;                     [1 unit]
  this.isRecurring = true;                     [1 unit]
  this.recurrence = recurring;                 [1 unit]
 }
```

**Note:** While there are functions used in the code above whose time is not necessarily 1 unit like date.minusDays(1) which subtracts 1 day from the current day the functions are also simple functions equivalent to mathematical expressions and can be approximated to be constant time operations therefore overally the Event class had a time complexity of O(1).

**The Day**

The Day object was an object that could model a day on a calendar which could hold a sequence of events. To implement the day we used an underlying data structure of a Priority Queue that would organize events based on their entry time but would give priority to events with an early start time to ensure events are always sorted in the proper chronological order, even if the user first created a later event first. The time for insertion, retrieval, and deletion of an event in this order was O(log n) since the priority queue in Java is implemented using a binary heap. However, operations like displaying all events in a day which require us to visit all events in the queue would have a time complexity of O(n) since each event has to be visited at least once.

**The Calendar**

The calendar was implemented using a hashtable with the date of an event as the key and the day i.e. a priority queue of events as the value. We chose to use the hashtable as underlying data structure as accessing of each day would be a constant time operation i.e O(1). This is due to the implementation of the hashcode() hash function used by the hash table that uniquely maps each key to a value. The calendar view operation in the calendar implementation was the most time consuming portion of the Calendar as it required an iteration through all the dates stored as keys in the hashtable for a month. In the worst case scenario all days of the month have n events. Since each month has a maximum of 31 days, the maximum time complexity would be 31 x n where n is the maximum number of events per day therefore the time complexity would still be O(n).

**The GUI**

The graphic user interface implemented using JavaX.Swing was based on an event-driven execution that implemented the background logic on a click  event. Each operation of a button click is itself only a constant time operation i.e. O(1).

**Total Time Complexity**

Due to the modular structure of our code the overall worst case time complexity can easily be calculated by adding the worst case time complexities of the four independent sectionctions i.e.

**O(1) + O(n) + O(n) + O(1)**

Therefore the overall time complexity for our entire project was O(n).

**Conclusion**

It can be seen from our analysis of the data structures and algorithms implemented that the overall time complexity of the Event Scheduler and Calendar can be O(n).