

# CPT204 Hyflex 2021

## Lab 14 Task Sheet

---

### Overview

Lab 14, the final part of the continuous assessment this semester, consists of 3 parts: Part A, B and C.

In **Part A**, you will implement a data structure called Explicit Minimum Priority Queue. You will then use this data structure to solve a problem called Maximum Attendance in **Part B**. Finally, in **Part C**, you will have to solve three problems of various data structures, problem-solving and object-oriented techniques that are derived from Lab 1 – Lab 13.

Submit all your answers for parts A, B, and C to Learning Mall for grading on the Submission Day.

### Timeline

Week 1 – Week 13	Lab 1 – Lab 13
Week 13, Tuesday, May 25, 2021	Lab 14 Part A, B released (Task Sheet, Skeleton Codes, Partial Test Cases)
Friday, June 4, 2021	Lab 14: Submission Day
- 13.00 - 14.00 CST	Lab 14 Part A, B online submission open – closed
- Lab Group Schedule, 40 minutes	Lab 14 Part C released Lab 14 Part C on-site, online submission open – closed

### Outline

The rest of the task sheet will describe all the three parts and the Submission Day in detail.

## Lab 14 – Part A

### Explicit Minimum Priority Queue

---

Recall that in Lecture 13 and Lab 13, we have constructed a Minimum Priority Queue Data Structure, and achieved amortized  $O(\log n)$  time for add and delMin operations implemented using resizable array.

In part A, you are to implement Explicit Minimum Priority Queue Data Structure using ArrayList. It supports the same operations as the Minimum Priority Queue. However, the priority is explicitly represented as an integer.

### Priority Queue

Minimum Priority Queue supports fast removal of the smallest item in the data structure. It should also support fast item addition.

To have a notion of smallest, either we implement the item to be comparable, or we explicitly specify the priority of an item with a comparable value, for example an integer.

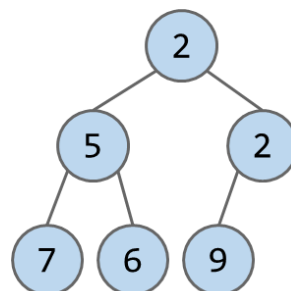
Implemented naively using an array, we can achieve constant-time item addition but linear-time smallest item removal; while on the other hand, using an ordered array, constant-time removal but linear-time addition. Our goal is then to have an implementation that can achieve fast running-time for both operations.

### Binary Heap

Binary heap is a binary tree that is complete and satisfies the heap-property:

- Complete: balanced binary tree, could be left-aligned at the bottom level.
- Heap-property: parent's item/priority is smaller or equal to the children's.

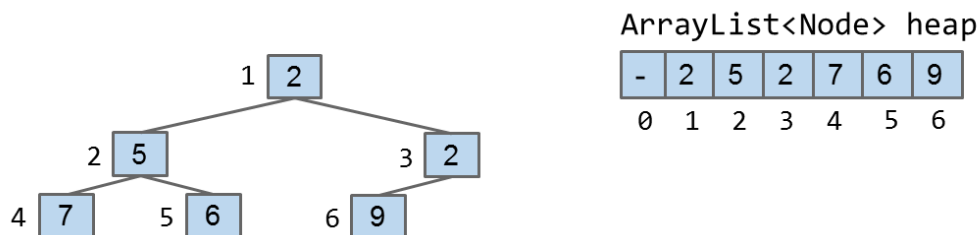
For example:



If we represent the binary heap by an array or an `ArrayList`, set the indices to start at 1, and order the nodes containing the item and the priority by the heap-property:

- There is no explicit links needed between parent node and the children
- We can use the indices to access a node's parent or children
  - Parent of a node at index  $k$  is  $k/2$
  - Children of a node at index  $k$  is  $2*k$  and  $2*k + 1$

For example:



You can then use the following ideas to implement the Minimum Priority Queue:

- Item addition: add the new item at the end, then swap it with its parent repeatedly until the heap-property is restored.
- Minimum item removal: replace the minimum item at the root with the last item, then swap it with the smallest children repeatedly until the heap-property is restored.

## Part A: Explicit Minimum Priority Queue

In part A, your task is to complete an implementation of the Explicit Minimum Priority Queue `ExpMinPQ` interface. It is similar to the `MinPQ` interface that we discussed and implemented in Lecture 13 and Lab 13.

Specifically, you will have to implement the following constructor and methods:

1. `public ALBinHeap()` initializes an empty binary heap.
2. `public void add(T item, int priority)` adds a new item with its corresponding integer priority to this binary heap.  
It throws an `IllegalArgumentException` object if item is null.  
There may be multiple items with the same priorities in the heap. If there are two items with the same priority, you may break ties arbitrarily.
3. `public T getMin()` returns an item with a smallest priority on this binary heap.  
It throws a `NoSuchElementException` object if this binary heap is empty.

4. `public T delMin()` removes and returns an item with a smallest priority on this binary heap.  
It throws a `NoSuchElementException` object if this binary heap is empty.

In addition, you must follow the following rules:

- You will implement a Binary Heap represented using an `ArrayList`. In addition, there is another instance variable `size`, that stores the number of items in the heap. You must *not* add any instance variables.
- `ArrayList` and `NoSuchElementException` will be imported in the Learning Mall Quiz. You must *not* import another libraries.
- You must *not* use any external libraries, such as `java.util.PriorityQueue`.
- All the methods must run in  $O(\log n)$  time.

Failing any of the rules above will result in **0 marks**.

The total marks of all the implementations in part A is **100 points**, for passing all the test cases.

## Advice

The following advice may be found useful in implementing Part A:

1. Use the same Automated Regression Unit Testing and Integration Testing strategy that you have been using in Lab 13. Note that with the use of explicit priority, some input parameters may be different from the ones in Lab 13 test cases. JUnit test cases is provided in `ALBinHeapTest.java` file.
2. Add more test cases, and create a good suite of test cases and practice the Partitioning/Boundary, Black-box/White-box, and Coverage testing.
3. Debug with the help of the `heapVisualize` method provided in `HeapVisualizer.java` file.
4. You may define your own private helper methods. Include them in *each* of your submissions.
5. Do not define your own instance variables. They are not going to be used in the hidden test cases and may cause unpredictable errors in the grading system.

## Lab 14 – Part B

### Maximum Attendance

---

In part B, you are going to use the data structure you have developed in part A to solve an interesting computational problem *efficiently*.

#### Maximum Attendance Problem

As a member of the university management team, you are given a list of the university course activities which has been arranged and scheduled by your colleague. Each course activity object stores a number of data including the activity day, the start time the end time, and the number of students. Each course activity is *at least* 1 hour long.

Due the Covid-19 pandemic mitigation measure, there is a limit on how many students can be inside the campus building at the same time. You are charged to make sure that indeed the limit is not violated.

Your task is then to implement an algorithm for finding the largest number of students attending some course activities at any particular same time.

#### Course Activity Representation

We represent the course activities with a class called `CourseActivity`.

These are the instance variables:

1. `type`. The type of the course activity in `String`.
2. `courseCode`. The code of the course in `String`.
3. `day`. The day the activity is scheduled on represented in `int`, starting from 1 := Monday until 7 := Sunday.
4. `startTime`. The starting time of this course activity in `int` from 0 to 23.
5. `endTime`. The ending time of this course activity in `int` from 1 to 24.
6. `numStudents`. The number of students to attend the course activity in `int`.
7. `roomCodeNumber`. The room code number of a lecture hall, a tutorial classroom, or a lab room in `String`.

## Part B: MaxAttendance

In part B, your task is to complete the skeleton code of the MaxAttendance class in order to figure out what is the maximum attendance of students.

You are given two ALBinHeap objects from Part A as the instance variables:

- `private ALBinHeap<CourseActivity> minPQ1`
- `private ALBinHeap<CourseActivity> minPQ2`

You must use those Explicit Minimum Priority Queue implementation to solve the Maximum Attendance problem efficiently.

You will have to implement in these constructor and method to complete the class:

1. `public MaxAttendance(ArrayList<CourseActivity> activities).`  
The constructor takes an ArrayList of CourseActivity to initialize the two binary heap instance variables.
2. `public int maxAttendance().` This method returns the solution to the Maximum Attendance problem using the two binary heap instance variables.

### Testing File and Test Case 1

You are given the following test case in MaxAttendanceTest.java file:

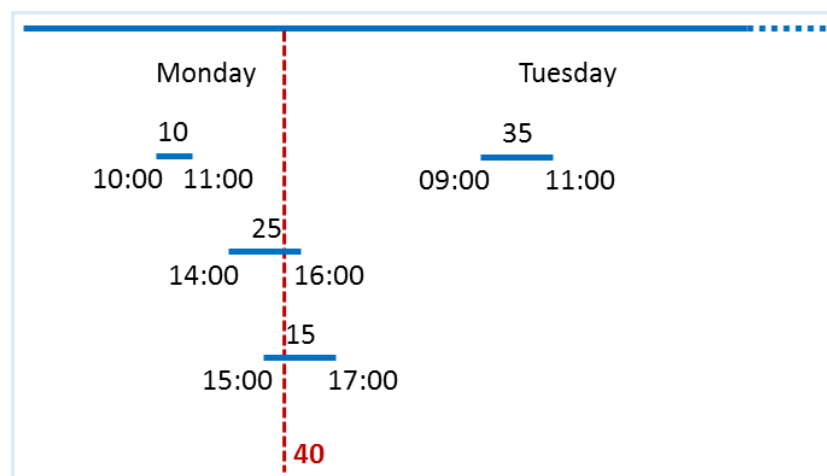
#### Input:

Course Activity 1: Monday, 10:00 - 11:00, 10 students

Course Activity 2: Monday, 14:00 - 16:00, 25 students

Course Activity 3: Monday, 15:00 - 17:00, 15 students

Course Activity 4: Tuesday, 09:00 - 11:00, 35 students



#### Output:

40

### Explanation:

The maximum number of student attending some course activities at the same day and time happens on Monday at about 15:00.

In addition, you must follow the following rules:

- You must use the two Explicit Minimum Priority Queue implementations, which is implemented as `ALBinHeap` in your solution. You must *not* add any instance variables.
- `ArrayList` will be imported in the Learning Mall Quiz. You must *not* import another libraries.
- You must *not* use any external libraries, such as `java.util.PriorityQueue`.
- Your algorithm must support multiple course activities scheduled at the same time.
- Your algorithm must run in  $O(N \log N)$  time, where  $N$  is the total number of course activities in the list.

Failing any of the rules above will result in **0 marks**.

The total marks for part B is **100 points**, for passing all the test cases.

### Advice

The following advice may be found useful in implementing Part B:

1. Add more test cases using the Automated Regression Unit Testing and Integration Testing strategy into the given testing file `MaxAttendanceTest.java`.
2. Create a good suite of test cases and practice the Partitioning/Boundary, Black-box/White-box, and Coverage testing.
3. You may define your own private helper methods. Include them in *each* of your submissions.
4. Do not define your own instance variables. They are not going to be used in the hidden test cases and may cause unpredictable errors in the grading system.

## Lab 14 – Part C

---

In part C, you are going to solve 3 questions, closely related to the works you have done throughout the semester in Lab 1 – Lab 13. It is an open book lab test. You may open the course pdf or bring hardcopies of course notes. You may open your source codes in IntelliJ. You may open and review past course quizzes in Learning Mall.

Relative to the programming questions in the Lab Exercises, there will be 2 *easy* and 1 *hard* coding questions. There are multiple possible candidate questions for each question with the same difficulty, you will be given one of them randomly.

While the specific questions are not going to be revealed here, the range of topics will be given below. You can also practice by reviewing all your works in Lab 1 – Lab 13. You will be able to access the questions in the respective Learning Mall Quizzes on the Submission Day.

In addition, you must follow the following rules:

- You must *not* communicate or share anything in any way with anybody other than the staff at all.
- You must *not* open any other websites.
- You must *not* use any external libraries and you must follow the specific instructions in any particular questions.
- For online students, you *must* show your screen and your hands at any time using a webcam.

Failing any of the rules above will result in **0 marks** with no warnings given.

The marks for each question in part C is **100 points**, for a total of **300 points**.

### Data Structure

List, ArrayList, MyList, SLList, DLList, ARList

Deque, LLDeque, ARDeque

Map, HashMap, HAMap

Set, ARSet, HASet

MinPQ, ARBinHeap

Disjoint Sets, Quick Find, Quick Union, Weighted Quick Union

Generic Data Structure of the above and their subclasses

CPT204 Hyflex

Erick Purwanto – May 2021



## Object-oriented Features and Problem-solving Techniques

Empty Constructor, Default Constructor, Copy Constructor, Deep Copy

Iterative, Recursive, Recursion with Helper Method

Mutates, Not Mutate, Immutable

Resizing Array, Table Doubling/Halving

Checked/Unchecked Exception, Assertion

Iterator, Iterable, Enhanced For Loop, ToString

Interface, ADT, Interface Inheritance, Implementation Inheritance, Casting

Static/Dynamic Type, Dynamic Method Selection, Overloading, Overriding

Equality, Higher Order Functions, Comparator, Comparable, hashCode

## Lab 14 – Submission Day

---

The submission day is on Friday, June 4, 2021, in two separate time periods.

First, submit Part A and Part B online at 13.00 - 14.00 CST.

















Second, submit Part C during your usual lab time according to your lab group schedule. If you are returning students, come to and submit in the lab rooms; and if you are non-returning students, submit online invigilated by a teacher using BBB.

At those times, open your CPT204 course page on Learning Mall.

### Learning Mall Lab 14 Quiz Section

The quizzes will be accessible on June 4, 2021 : Part A, B at 13:00 CST; and Part C 10 minutes after your respective lab group starting time (16:00 or 17:00 or 18:00 CST).

You will see a similar section shown below.

Submission Day		
	Part A.1 ALBinHeap Constructor	
	Part A.2 ALBinHeap Add	
	Part A.3 ALBinHeap Get Min	
	Part A.4 ALBinHeap Del Min	
	Part B Max Attendance	
	Part C.1	
	Part C.2	
	Part C.3	

Additionally for part C, you will also see a problem description and class/method specification outlining each problem, similar to our lab sheets.

Some may include a skeleton code as well. You may want to copy paste the skeleton code into your coding environment, so please have it ready.

## Learning Mall Quiz Submission

**Part A and Part B.** When the quiz open at 13:00 CST, you will see some additional test cases. You may want to check your code against those test cases in your IDE first, *not* in the Learning Mall Quiz.

You have 1 hour to test, debug — if needed, and submit your code. Submit your code before the closing time at 14:00 CST. It is *highly recommended* to submit a few minutes earlier to account for network transmission delay.

**Part C.** The quizzes will open 10 minutes after your lab group starts. You have to complete the method or the class, as specified in the problem description and specification in 40 minutes. You may use *Check* against the test cases multiple times.

In addition, you may include your private helper methods in the submission box.

Adding another elements, such as importing libraries, will result in 0 marks.

The first *Check* will have no penalty if failing the test cases. The subsequent *Checks*, however, will each incur an additional 15% penalty, cumulatively for each incorrect *Check*.

**Question 1**

Not complete Marked out of 100.00

Complete the method `public int someMethod()`.  
It computes something and returns an integer.

**For example:**

Test	Result
<pre>int output = someMethod(); System.out.println(output);</pre>	5

**Answer:** (penalty regime: 0, 15, 30, ... %)

Reset answer

```
1 // Lab 14 Part C.1
2
3 /**
4  * Complete the following method.
5  * @return an integer
6  */
7 public int someMethod() {
8
9
10
11 }
```

Check

## Useful Tips for Submitting Your Work

1. We will release the lab seating arrangement before the Submission Day. If you are a returning student, please make sure you prepare the IDE, the library and the data structure files in your Windows account on the specific lab computer assigned to you.
2. If you are a non-returning student, join our mock online submission session, to help you prepare for the additional online invigilation rules.
3. If you are copy paste from your IDE, please double check the parenthesis, class/method headers, etc., before clicking the *Check* or *Finish Attempt* button.
4. Test your code intensively before submitting your work.
5. Familiarize yourself with the error messages of the autograder systems that we have seen throughout the semester.

## Academic Integrity

This Lab 14 coursework assignment is individual work. Plagiarism (e.g. copying materials from other sources without proper acknowledgement) is a serious academic offence. Plagiarism and collusion will not be tolerated and will be dealt with in accordance with the University Code of Practice on Academic Integrity. Individual students may be invited to explain parts of their code in person, and if they fail to demonstrate an understanding of the code, no credit will be given for that part of the code.