CMPE 242
Spring 2021
Programming Homework 4

This assignment is due by 23:59 on Saturday, June 5, 2021.

You are welcome to ask your HW related questions. Please use one of these options:

1. Moodle Homework **Question Forum**: HW Question-and-Answer (Q&A) Forum on Moodle is always available. Use the "Forum" link at the course Moodle page.
2. Homework **RECITATION HOURS**: There will be two Q&A RECITATION HOURs on the following days:
   - CMPE242-HW4-OfficeHour1: May 31, 06:00-07:00 PM, Zoom ID: 975 6275 6390
   - CMPE242-HW4-OfficeHour2: June 4, 06:00-07:00 PM, Zoom ID: 988 7635 3095

Note: Please make sure that you have read the HW document well before participating.

Partner Selection (Due by 23:59 on Thursday, May 27, 2021)

For this assignment, you may work with one partner. If you do so, the two of you will turn in only one assignment and, except in extraordinary situations, receive the same grade (and, if applicable, have the same number of late days applied). Working with a partner is optional; it is fine to complete the assignment yourself.

If you choose to work with a partner, **one of you must submit group names before the partner-selection due date**. If you choose to work alone or have an existing partner, also submit to Moodle and inform us by submission. After the partner selection deadline, you are not allowed to make any change on how you will work on the assignment.

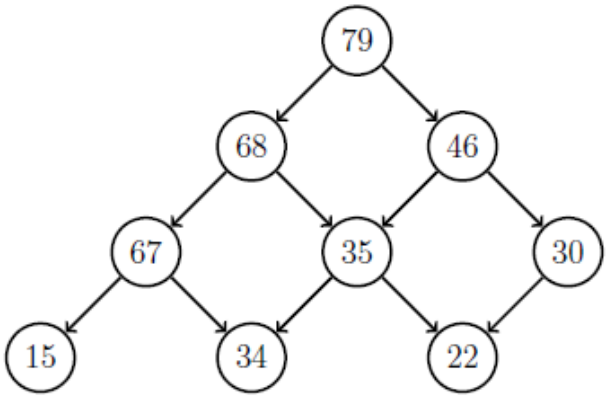
If you choose to work with a partner, you may divide the work however you wish, but both partners must understand and be responsible for everything that is submitted. The TA may meet with you to go over your homework. Beyond working with a partner, all the usual collaboration policies apply.

PART I (30 points)

A *pyramid* is a data structure similar to a heap that can be used to implement the priority queue ADT. As with a heap, a pyramid is defined by two properties:

- Structural property: A pyramid P consists for x levels, $x >= 0$. The ith level, for $0 <= i <= x$, contains at most $i + 1$ entries, indicated as $P_{i,j}$ for $0 <= j <= i$. All levels but the last are completely filled, and the last level is left-justified.
- Ordering property: Any node $P_{i,j}$ has at most two children: $P_{i+1,j}$ and $P_{i+1,j+1}$, if those nodes exist. The priority of a node is always greater than or equal to the priority of either child node.

For example, the following diagram shows a pyramid with 9 nodes and 4 levels. The arrows indicate " >=" relationships.



Answer the four following questions below. **Your own answers must be in paper & pencil format (handwriting) and prepared as a single maximum 2-pages PDF document. If you work with a partner, please also write all the group information on it.**

1. (5 points) A pyramid P with n nodes can be stored in an array A of size n, similarly to an array-based heap. So, for example, if $n >= 1$, the top of the pyramid $P_{0,0}$ will be stored in $A[0]$. Give formulas for the array index that the following pyramid entries will have. Assume that n, i, and j are such that all indicated pyramid nodes actually exist. You do not need to justify your answers.

i. $P_{i,j}$
ii. Left and right children of $P_{i,j}$
iii. Left and right parents of $P_{i,j}$
iv. Left and right children of the node corresponding to $A[i]$
v. Left and right parents of the node corresponding to $A[i]$

2. (5 points) Give upper and lower bounds for the number of nodes *n* in a pyramid *P* with *x* levels, where *x* >= 1. For example, a pyramid with 2 levels has at least 2 and at most 3 nodes. Make your bounds as tight as possible.

3. (8 points) Give pseudocode for the *insert* operation that takes a pyramid stored in an array *A* of size *n* and an element *x* and inserts the new element into the pyramid. Show that your algorithm has time complexity $O(\sqrt{n})$.

4. (12 points) Consider the *contains* problem: Given an array *A* of size *n* and a number *x*, determine whether *x* is an element of *A*. For example, if *A* is sorted, the contains problem can be solved in *O*(log n) time by using a binary search. Give pseudocode for an algorithm to solve the contains problem when the input array *A* is a pyramid as described above. Show that the running time of your algorithm is $O(\sqrt{n}\log n)$. (**Hint**: A pyramid contains some sorted lists.)

PART II (70 points)

Priority queues are frequently used in real world. Different jobs like processes have different priorities for scheduling. Some jobs need to be completed before than others that is named as priority-based scheduling. The mad scientist, *Sheldon James*, is a very busy person and having trouble with arranging his daily tasks within the specified time and duration. **In this part, you must implement your own heap-based priority queue data structure by taking inspiration from your textbook and use it to help him schedule his real-time tasks. Since he is a mad scientist, number of tasks is not limited so that you should also overcome this issue while implementing your priority queue (e.g. using `Vector` from `java.util`). You are not allowed to use any external library or .jar file.**

You read tasks from the input text file which includes lines in two forms such that: `schedule` *task deadline duration*, or `execute` *time* where *task* is an arbitrary `string`, *deadline*, *duration* and *time* are `longs` representing time. Here are the `sampleinput1.txt` file contents:

```
schedule lunch 1300 30
schedule fun 7200 2300
schedule programming 2359 300
run 2400
```

Even if it does not matter of what the units of time are, you may think first two digits as hours and the last two digits as minutes. So, programming should be scheduled with a deadline of 23:59 (pm) and 3:00 (three hours) should be allowed to complete the task.

Each line beginning with `schedule` tells your program to insert the corresponding item into the priority queue, with a priority corresponding to its deadline. Items with smaller deadlines should be executed first.

Your program must have a long integer variable to represent time. That variable is initially set to zero, and is increased as the program runs.

Each line beginning with `run` *time* tells your program to advance your internal time variable until it reaches *time*. You do this by repeatedly removing the first item from the priority queue, and advancing the internal time variable by the *duration* of the task, until the *time* is reached. If the task completes before the *time* is reached, a new task is taken from the priority queue. If there are no tasks left in the priority queue, the internal time is simply set to *time*. If, on the other hand, the current task would run past *time*, it is re-inserted in the priority queue with duration equal to the remaining time.

So in the `sampleinput1.txt` file above, time starts at zero, and when your program reads the line that says `run 2400` it starts to run the first task in the queue, which will be "lunch". Lunch takes 30 minutes, so your program will remove "lunch" from the queue and set its time variable to 30. The next item removed from the queue will be "programming" (with an earlier deadline than "fun"), which takes time 300. So, the time variable is set to 330. Finally, "fun" is removed from the queue. The time is now 330, and fun is scheduled to take 2300 time units. But, this run ends at time 2400, whereas if all the fun was done now, the run would end at time 2630. So, the time variable is set to 2400, and your program must re-insert "fun" into the priority queue with a deadline of 7200 (the deadline hasn't changed) and a duration of 230 (which is 2300 - (2400 - 330), the duration minus the difference between the end time and the start time).

Here are the output specifications:

- Every time a task is removed from the queue, your program should print
  current time: busy with *task* with deadline *deadline* and duration *duration*
  (where *current time* is the value of the time variable when the task is first removed from the queue.)
- When a task completes, your program should print
  current time: done with *task*
- If the task completes after its deadline, your program should print
  current time: done with *task* (late)
- Every time a task is inserted into the queue, your program should print
  current time: adding *task* with deadline *deadline* and duration *duration*

The output for the `sampleinput1.txt` file is given as follows. Please check your program with this input file as well as the others that you will create. Please note that we will use other input files when grading your assignments.

```
% java Scheduler sampleinput1.txt

0: adding lunch with deadline 1300 and duration 30
0: adding fun with deadline 7200 and duration 2300
0: adding programming with deadline 2359 and duration 300
0: busy with lunch with deadline 1300 and duration 30
30: done with lunch
30: busy with programming with deadline 2359 and duration 300
330: done with programming
330: busy with fun with deadline 7200 and duration 2300
2400: adding fun with deadline 7200 and duration 230
```

WHAT TO HAND IN

A zip file for both parts containing:

- The Java sources for your program.
- The Java sources should be WELL DOCUMENTED as comments, as part of your grade will be based on the level of your comments
- A **maximum-2 pages** PDF document that explains your own answers for PART I in a clearly readable form.
- If you work with a partner, single submission should be made from one of your group members.

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. This assignment is due by 23:59 on Saturday, June 5th, 2021.
2. You should upload your homework to Moodle before the deadline. No hardcopy submission is needed. You should upload files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., zip, rar).
3. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.
4. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.

5. Your classes' name MUST BE as shown in the homework description.

6. The submissions that do not obey these rules will not be graded.

7. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

8. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-------------------------------------------------------
// Title: Scheduler tester class
// Author: Ibrahim Ileri
// ID: 2100000000
// Section: 1
// Assignment: 2
// Description: This class tests the …
//-------------------------------------------------------
```

9. Since your codes will be checked without your observation, you should report everything about your implementation. Add detailed comments to your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void setVariable(char varName, int varValue)
//-----------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
//-----------------------------------------------------------
{
      // Body of the function
}
```

10. Indentation, indentation, indentation...

11. This homework will be graded by your TAs, İbrahim İleri and Hamid Ahmadlouei (ibrahim.ileri@tedu.edu.tr, hamid.ahmadlouei@tedu.edu.tr). Thus, you may ask them your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course instructors Tolga Çapın and Ulaş Güleç for help.

GRADING RUBRIC

PART II (70 points)						
Performance Element	**Weight**	**Master (4/4)**	**Advanced (3/4)**	**Developing (2/4)**	**Beginner (1/4)**	**Insufficient (0/4)**
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	5%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc.).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
Code Design	5%	Moduler source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
Abstract Data Type: Heap-based Priority Queue	10%	Uses heap-based priority queue data structure for implementation. And provides all functionality.	Uses heap-based priority queue data structure for implementation. And provides most of expected functionalities.	Implements heap-based priority queue with major deviations from the specification.	Used different data structure from heap-based priority queue to solve this problem.	Not even an attempt.

Functionality	65%	All functions are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all functions but some of them do not work.	Only few functions are implemented correctly. Compiles but several warnings.	No working solution or does not compile.
Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set.	Provided a tester class. Able to identify key test points.	Sporadic.	No test case presented.