COMP4500/7500
Advanced Algorithms and Data Structures

School of Information Technology and Electrical Engineering
The University of Queensland, Semester 2, 2018

Assignment 2

Due at 4pm, Friday 19th of October 2018.
This assignment is worth 20% (COMP4500) or 15% (COMP7500) of your final grade.

This assignment is to be attempted **individually**. Please read this entire handout before attempting any of the questions.

Submission. Answers to each of the written (not programming) questions (i.e. Q1(a), Q1(b), Q2(b) and Q2(d)) should be clearly labelled and included in a pdf file called `A2.pdf`.

You need to submit (i) your written answers in `A2.pdf`, as well as (ii) your source code files `Recursive.java` and `Dynamic.java` electronically using Blackboard according to the exact instructions on the Blackboard website: `https://learn.uq.edu.au/`

You can submit your assignment multiple times before the assignment deadline but only the last submission will be saved by the system and marked. Only submit the files listed above. You are responsible for ensuring that you have submitted the files that you intended to submit in the way that we have requested them. You will be marked on the files that you submitted and not on those that you intended to submit. Only files that are submitted according to the instructions on Blackboard will be marked.

Submitted work (including code) should be neat, legible and simple to understand – you may be penalised for work that is untidy or difficult to read and comprehend.

For the programming parts, you will be penalised for submitting files that are not compatible with the assignment requirements. Code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks.

Late submission. Late assignments will lose 10% of the maximum mark for the assignment immediately, and a further 10% of the maximum mark for the assignment for each additional day late. Assignments more than 5 days late will not be accepted.

If there are medical or exceptional circumstances that will affect your ability to complete an assignment by the due date, then you can apply for an extension as per Section 5.3 of the electronic course profile (ECP). Requests must be made at least 48 hours prior to the submission deadline. Assignment extensions longer than 7 calendar days will not be granted.

School Policy on Student Misconduct. You are required to read and understand the School Statement on Misconduct, available at the School's website at:

`http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

This is an individual assignment. If you are found guilty of misconduct (plagiarism or collusion) then penalties will be applied. Please don't publish solutions to your assignment, even after the due date, since it may facilitate future cases of plagiarism.

Question 1 (20 marks total)

A computer disk consists of a head, H, that reads from or writes to numbered tracks of a disk platter. Suppose it takes one unit of time to move H one track, i.e., if the H is currently at track 10, it takes 167 time units to move it to track 177. Given a set of track requests and a start position for H, your task is to find an ordering of the requests so that they are completed in the minimal overall time. For example, suppose H is currently at track 200 and the set of requests is $\{40, 180, 300, 10\}$. The minimal ordering of the requests is $\langle 300, 180, 40, 10 \rangle$, which achieves a minimal time of 390.

Your algorithms must output both the minimal time, $minTime$, and the minimal ordering, $minOrder$, that achieves the minimal time. You may use pseudocode or actual computer language code. You must describe any data structures, etc that you use and code must be clearly commented.

a. (5 marks) Provide an efficient greedy algorithm for solving the disk seek problem.

b. (15 marks) Prove that your algorithm in Q1(a) produces a sequence that achieves the minimal time, i.e., an optimal solution.

Question 2 (80 marks total)

Suppose you are in charge of operating a high performance computing system (HPCS) for k consecutive days. Continuous operation of the system without performing maintenance activities can degrade the performance of the system. There are two kinds of maintenance activity that can be applied to the system: the system can either be *fully rebooted*, or *partially rebooted*. The maximum amount of data that the system is able to process in a given day is dependent on the last maintenance activity applied to the system, and the number of days since that maintenance activity.

When the last maintenance activity was a full reboot, the maximum amount of data that is able to be processed i days after that activity (for $0 \leq i$) is described by an array $fullRebootCapacity$ of n non-negative integers (indexed from 0): on the day of the full reboot activity the capacity of the system is $fullRebootCapacity[0]$; i days after the full reboot activity the capacity of the system is $fullRebootCapacity[i]$, if $1 \leq i < n$, and $fullRebootCapacity[n-1]$, if $i \geq n$.

Similarly, when the last maintenance activity was a partial reboot, the maximum amount of data that is able to be processed i days after that activity (for $0 \leq i$) is described by an array $partialRebootCapacity$ of m non-negative integers (indexed from 0): on the day of the partial reboot activity the capacity of the system is $partialRebootCapacity[0]$; i days after the partial reboot activity the capacity of the system is $partialRebootCapacity[i]$, if $1 \leq i < m$, and $partialRebootCapacity[m-1]$, if $i \geq m$.

Only one maintenance activity can take place on any one day, but otherwise there is no limit to the number of maintenance activities that can be performed, or the order in which they can be done. (For the k days that you are in charge of the system, it is up to you to determine when these activities should occur.)

You have been given a schedule, represented by an array $data$ of k non-negative integers (indexed from 0), for the amount of data that is scheduled to be processed for each of the k days that you are in charge of the system, e.g. the amount of data scheduled to be processed for the first day (day 0) is $data[0]$, for the second day (day 1) is $data[1]$ etc. Any data that could not be processed at the end of each day must be discarded, hence **the cost to your company is the amount of data that could not be processed**.

Given arrays $fullRebootCapacity$, $partialRebootCapacity$ and $data$, and the knowledge that the last maintenance activity that was performed on the system was a full reboot that took place the day before you were put in charge of the system, **your task is to find the least cost that can be incurred by your company over the k days that you operate the machine.**

Example

As an example, consider the following scenario:

$$\begin{aligned} fullRebootCapacity &= [0, 10, 4, 5, 1] \\ partialRebootCapacity &= [2, 8, 3, 2, 0] \\ data &= [15, 5, 12, 17, 7, 10] \end{aligned}$$

where you are in charge for $k = 6$ days, and a full system reboot took place the day before you were put in charge of the system.

For each of the k days (i.e day $0, 1, 2 \cdots k-1$) you are in charge of the system, you can either choose to perform one of the two possible maintenance activities, or no activity at all. You choices can have an impact on the cost incurred by your company. For example,

- If you choose to perform no maintenance activities at all:
  Total cost = (15-10) + (5-4) + (12-5) + (17-1) + (7-1) + (10-1) = 44
- If you choose to perform a full reboot on day 1 (i.e. the second day):
  Total cost = (15-10) + (5-0) + (12-10) + (17-4) + (7-5) + (10-1) = 36
- If you choose to perform a full reboot on both day 2 and day 4 (i.e. the third and fifth days):
  Total cost = (15-10) + (5-4) + (12-0) + (17-10) + (7-0) + (10-10) = 32
- If you choose to perform a partial reboot on day 1 (i.e. the second day) and a full reboot on day 4 (i.e. the fifth day):
  Total cost = (15-10) + (5-2) + (12-8) + (17-3) + (7-0) + (10-10) = 33

There are many other possible maintenance schedules. For this example, the least cost that can be incurred by your company for the $k = 6$ days that you operate the machine is 32.

a. (10 marks) Implement the public static method optimalCostRecursive from the Recursive class in the assignment2 package that is available in the zip file that accompanies this handout, to provide a recursive algorithm to determine the least cost that can be incurred by your company over the k days (i.e. day 0 to day k-1) that you operate the HPCS system. To implement that method, you will need to provide an implementation for the private static method optimalCostRecursive from the same class.

   The recursive solution does NOT need to find a schedule of activities that produces the least cost – it just needs to determine the least cost. Efficiency is not at all a concern for this part, so focus on an elegant solution.

b. (20 marks) It is expected that your recursive algorithm will not be polynomial-time in the worst case. For the case where the number of days that you are responsible for the HPCS system is k, give an asymptotic lower bound on the worst-case time complexity of your recursive algorithm in terms of parameter k. Make your bound as tight as possible.

   Give an argument explaining why the time-complexity is exponential in terms of a (lower bound) recurrence derived from your algorithm.

   [Make your answer as concise as possible – it should be no more than half a page using minimum 11pt font. Longer answers will not be marked.]

c. (30 marks) Develop a bottom-up dynamic programming solution to the problem **(not memoised)** by implementing the public static method optimalCostDynamic in the Dynamic class from the assignment2 package that accompanies this handout.

Your dynamic programming solution should run in polynomial time (in terms of k, the number of days you are responsible for the HPCS system).

This dynamic solution does NOT need to find a schedule of activities that produces the least cost – it just needs to determine the least cost.

d. (10 marks) Provide an asymptotic upper bound on the worst-case time complexity of your dynamic programming solution for part (c) in terms of the parameter k, the number of days you are responsible for the HPCS system. Make your bounds as tight as possible and justify your solution.

   [Make your answer as concise as possible – it should be no more than half a page using minimum 11pt font. Longer answers will not be marked.]

e. (10 marks) Extend your bottom-up dynamic programming solution from part (c) to calculate an optimal schedule of activities (that produces the least cost) by implementing the public static method optimalActivitiesDynamic in the Dynamic class from the assignment2 package.

   Like method optimalCostDynamic, your implementation of this method should run in polynomial time (in terms of k). It should be a bottom-up dynamic programming **(not memoised)** solution.

Practicalities

Do not change the class name of the `Recursive` or `Dynamic` classes or the package to which those files belong. You many not change the signatures of the methods that you have to implement in any way or alter their specifications. (That means that you cannot change the method name, parameter types, return types or exceptions thrown by the those methods.) Do not modify any of the other classes or interfaces or enumerated types defined in package `assignment2`.

You are encouraged to use Java 8 SE API classes, but no third party libraries should be used. (It is not necessary, and makes marking hard.) Don't write any code that is operating-system specific (e.g. by hard-coding in newline characters etc.), since we will batch test your code on a Unix machine. Your source file should be written using ASCII characters only.

You may not write and submit any additional classes. Your solution to Q2(a) should be self-contained in the `Recursive` class. Similarly your solution to parts Q2(c) and Q2(e) should be self-contained in the `Dynamic` class. Both of these classes will be tested in isolation and should not depend upon each other.

The zip file for the assignment also some junit4 test classes to help you get started with testing your code. The JUnit4 test classes as provided in the package `assignment2.test` are not intended to be an exhaustive test for your code. Part of your task will be to expand on these tests to ensure that your code behaves as required.

Your programming implementations will be tested by executing our own set of junit test cases. Code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks. A Java 8 compiler will be used to compile and test the code. The `Recursive` class will be tested in isolation from the `Dynamic` class.

Implementations that do not satisfy the assignment requirements will receive 0 marks even if they pass some of the test cases (e.g. if the solution given to Q2(c) is not a bottom-up dynamic programming solution, then it will receive 0 marks.)

You may lose marks for poorly structured, poorly documented or hard to comprehend code, or code that is not compatible with the assignment requirements. Line length should be less than or equal to 80 characters so that it can be printed – please use spaces to indent your code instead of tabs to ensure compatability with different machines. Don't leave print statements in your submitted code.

Evaluation Criteria

Question 1

- **Question 1 (a) (5 marks)**

  5 : A clear, correct and efficient greedy algorithm is given for solving the disk seek problem.

  3 : A correct and efficient greedy algorithm is given for solving the disk seek problem. There may be minor problems with the clarity of the solution.

  0 : Answer not given, or is incorrect or unclear.

- **Question 1 (b) (15 marks)**

  Zero marks will be given for this question if the answer to Q1(a) is not correct. Otherwise, the following marking scheme applies.

  15 : A clear and correct proof is given to show that the algorithm from Q1(a) produces an optimal solution to the question.

  10 : A mostly clear, correct and complete proof is given to show that the algorithm from Q1(a) produces an optimal solution to the question.

  5 : An attempt is made to prove that the algorithm from Q1(a) produces an optimal solution, but it may be unclear, incomplete, or contains mistakes.

  0 : Work with little or no academic merit.

Question 2

- **Question 2 (a) (10 marks)**

  Given that your implementation satisfies the requirements of the question, your implementation will be evaluated for correctness by executing our own set of junit test cases.

  10 : All of our tests pass

  8 : at least 80% of our tests pass

  6 : at least 60% of our tests pass

  4 : at least 40% of our tests pass

  2 : at least 20% of our tests pass

  0 : less than 20% of our test pass or work with little or no academic merit

  Note: Code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks. A Java 8 compiler will be used to compile and test the code.

  Implementations that do not satisfy the assignment requirements will receive 0 marks even if they pass some of the test cases.

  The Recursive class will be tested in isolation from the Dynamic class.

- **Question 2 (b) (20 marks)**

  For this part of the question, the analysis should be no more than 1/2 of a page using minimum 11pt font. Longer solutions will receive 0 marks. Also, if a plausible, neat, legible and simple to understand solution to Q2(a) has not been given, this question will receive 0 marks. Otherwise the following marking criteria applies.

- 20 : A correct asymptotic lower bound on the worst-case time complexity the recursive algorithm from Q2(a) is given in terms of parameter k. The lower bound, which should be exponential in k, should be as tight as reasonably possible for the algorithm at hand. The time-complexity given should be clearly justified by giving and solving a correct (lower bound) recurrence derived from your algorithm. Any assumptions made in the analysis are reasonable and clearly stated. Asymptotic notation should be used correctly and the asymptotic time complexity given has been simplified to remove lower order terms and unnecessary constant factors.
- 15 : A correct asymptotic lower bound on the worst-case time complexity the recursive algorithm from Q2(a) is given in terms of parameter k. The lower bound should be exponential in k. The time-complexity given should be mostly clearly justified by giving and solving a correct (lower bound) recurrence derived from your algorithm. Any assumptions made in the analysis are mostly reasonable and clearly stated.
- 10 : A reasonable attempt has been made to give a tight asymptotic lower bound on the worst-case time complexity of the recursive algorithm from Q2(a) in terms of parameter k, and to justify it with respect to a recurrence derived from the algorithm, however the analysis or justification may contain minor mistakes or omissions or lack clarity.
- 5 : An attempt has been made to give an asymptotic lower bound on the worst-case time complexity of the recursive algorithm from Q2(a) in terms of parameter k, and justify it, however it contains either a major mistake or many mistakes, gives an unreasonably loose lower bound, or is not clearly justified by giving and solving a correct (lower bound) recurrence derived from your algorithm.
- 0 : Work with little or no academic merit.

- **Question 2 (c) (30 marks)**

  Given that your implementation satisfies the requirements of the question (i.e. it is a bottom-up dynamic programming (not memoised) solution that runs in polynomial time in terms of k), your implementation will be evaluated for correctness and efficiency by executing our own set of junit test cases.

  - 30 : All of our tests pass
  - 24 : at least 80% of our tests pass
  - 18 : at least 60% of our tests pass
  - 12 : at least 40% of our tests pass
  - 6 : at least 20% of our tests pass
  - 0 : less than 20% of our test pass or work with little or no academic merit

  Note: Code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks. A Java 8 compiler will be used to compile and test the code.

  Implementations that do not satisfy the assignment requirements will receive 0 marks even if they pass some of the test cases.

  The Dynamic class will be tested in isolation from the Recursive class.

- **Question 2 (d) (10 marks)**

  For this part of the question, the analysis should be no more than 1/2 of a page using minimum 11pt font. Longer solutions will receive 0 marks. Also, if a plausible, neat, legible and simple to understand solution to Q2(c) has not been given, this question will receive 0 marks. Otherwise the following marking criteria applies.

10 : A correct asymptotic upper bound on the worst-case time complexity of the algorithm from Q2(c) is given in terms of parameter k. The upper bound, which should be polynomial in k, should be as tight as reasonably possible for the algorithm at hand. The time-complexity given should be clearly justified with respect to the algorithm. Any assumptions made in the analysis are reasonable and clearly stated. Asymptotic notation should be used correctly and the asymptotic time complexity given has been simplified to remove lower order terms and unnecessary constant factors.

7 : A correct asymptotic upper bound on the worst-case time complexity the algorithm from Q2(c) is given in terms of parameter k. The upper bound should be polynomial in k. The time-complexity given should be mostly clearly justified with respect to the algorithm. Any assumptions made in the analysis are mostly reasonable and clearly stated.

5 : A reasonable attempt has been made to give a tight asymptotic upper bound on the worst-case time complexity of the algorithm from Q2(c) in terms of parameter k, and to justify it, however the analysis or justification may contain minor mistakes or omissions or lack clarity.

3 : An attempt has been made to give an asymptotic upper bound on the worst-case time complexity of the algorithm from Q2(c) in terms of parameter k, and justify it, however it contains either a major mistake or many mistakes, gives an unreasonably loose lower bound, or is not clearly justified.

0 : Work with little or no academic merit.

- **Question 2 (e) (10 marks)**

  Given that your implementation satisfies the requirements of the question (i.e. it is a bottom-up dynamic programming (not memoised) solution that runs in polynomial time in terms of k), your implementation will be evaluated for correctness and efficiency by executing our own set of junit test cases.

  10 : All of our tests pass

  8 : at least 80% of our tests pass

  6 : at least 60% of our tests pass

  4 : at least 40% of our tests pass

  2 : at least 20% of our tests pass

  0 : less than 20% of our test pass or work with little or no academic merit

  Note: Code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks. A Java 8 compiler will be used to compile and test the code.

  Implementations that do not satisfy the assignment requirements will receive 0 marks even if they pass some of the test cases.

  The Dynamic class will be tested in isolation from the Recursive class.