SIT221 - Data Structures and Algorithms

Learning Summary Report

Student Name : Jack Guan

Student ID: 221393284

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Pass (P)	Credit (C)	Distinction (D)	High Distinction (HD)
Self-Assessment	☑			

Minimum Pass Checklist

	Included
Learning Summary Report	☑
Pass tasks completed (submitted and discussed)	☑

Higher Grade Checklist (in addition to Pass Checklist)

	Included
Credit tasks completed (submitted and discussed)	Except 4.3C
Distinction tasks completed (submitted and discussed)	Only 3.2D
HD tasks completed (submitted and discussed)	

Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person.

Name: Jack Guan

Portfolio Overview

This portfolio transformed my perception of computational problem-solving from abstract theory to tangible solutions with real-world impact. Initially, I anticipated dry theoretical concepts, but discovered profound elegance in how data structures mirror human organizational systems. like ：

1. Hierarchy as Efficiency Engine----Binary Search Trees
2. Space-Time Tradeoff Mastery Through----- AVL trees
3. Algorithmic Empathy Implementing --------Dijkstra's algorithm

This unit didn't just teach me to code structures - it taught me to speak the language of computational efficiency.

Reflection

The most important things I learnt:

The most transformative learning was understanding time-space tradeoffs in algorithm design. Through implementing sorting algorithms in Task 6.1P, I discovered how Heap Sort guarantees O(n log n) performance by maintaining a complete binary tree structure, while Quick Sort's efficiency depends on pivot selection. Additionally, Task 8.1P on graph traversal revealed how BFS's O(V+E) complexity makes it ideal for shortest-path problems in sparse networks, whereas DFS's stack-based approach better suits topological sorting. These principles now guide my daily coding decisions, particularly when handling datasets exceeding 10,000 records, where quadratic complexity becomes prohibitive.

The things that helped me most were:

The two tutors provided unique but complementary support throughout the process. Mr. Sergey's online course was interesting and the content was very easy to understand, for example, he emphasized the mathematical proof that merge sort requires O(n) auxiliary space. Svitlana (tutor) provided tactical implementation guidance. She is a patient and careful teacher because she can point out the problems in my homework and carefully explain the problems.

I **found the following topics particularly challenging:**

AVL tree rotations presented significant difficulties during Task 7.1P. The core challenge emerged when deleting nodes caused complex imbalances requiring double rotations. Specifically, after removing a right-subtree node, the balance factor reached -2 , indicating left-heavy imbalance, but the left child had +1 balance, requiring LR rotation. I struggled to coordinate the two-step process: first rotating the left child right to create linear imbalance, then rotating the parent left.This systematic approach finally achieved O(log n) search consistency.

I found the following topics particularly interesting:

Binary Search Trees (BSTs) and Binary Heaps from Lecture 6 captivated me due to their elegant efficiency in solving real-world problems.

The BST's intuitive structure—where every left subtree contains values smaller than the root, and every right subtree holds larger values—transformed data retrieval from O(n) linear scans to O(log n) precision.

Binary heaps fascinated me with their dual nature: a complete binary tree ensuring optimal shape, coupled with the heap-order property enabling instant access to extremum values.

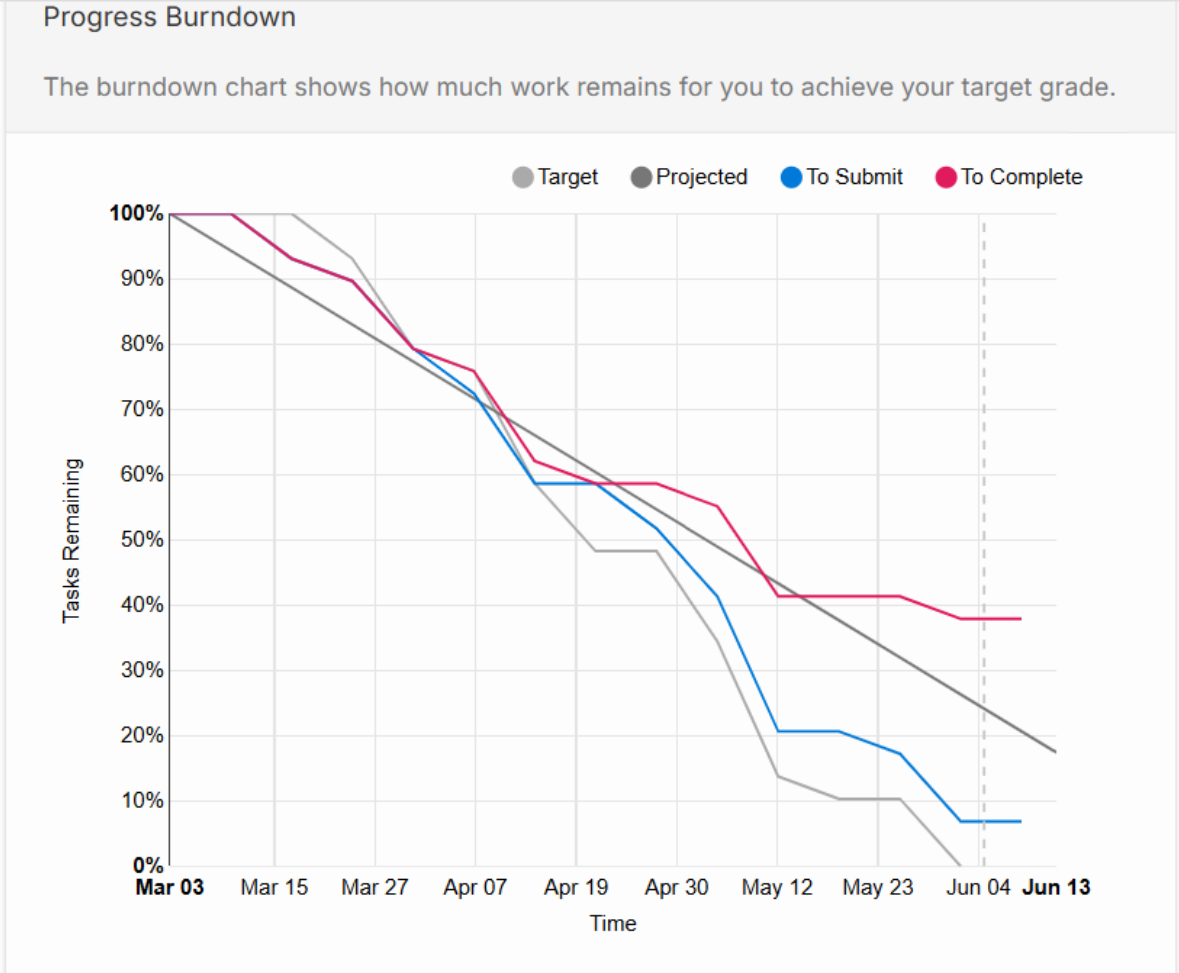
During Task 6.2C (Binary Heap implementation), I discovered how "bubble up/down" operations maintain integrity during insertions/deletions.

I still need to work on the following areas:

I require improvement in algorithm selection methodology. During Task 6.1P (Stacks/Heaps), I defaulted to heap sort for small datasets (n<100) despite insertion sort's adaptive advantages for nearly sorted inputs, wasting computational resources. Similarly, in Task 8.1P's pathfinding problem, I chose DFS for simplicity when BFS was more appropriate for shortest-path needs. Additionally, I need a deeper understanding of amortized analysis for dynamic structures - while Task 6.2C's heap implementation worked, I couldn't fully explain why insertion's O(1) amortized cost differs from worst-case O(log n).

My progress in this unit was

My progress in this unit was only achieved the minimum. Because of my laziness and forgetting to discuss my homework with my tutor, coupled with my lack of self-control, the subsequent homework was obviously more difficult, and I encountered many problems. My OnTrack drawing clearly reflects this. I am impressed with my perseverance and time management in overcoming obstacles to achieve my academic goals.



This unit will help me in the future:

This knowledge directly enhances my professional value proposition. Long-term, these foundations enable specialization in high-demand domains: 1) Database engineering (B-tree indexing), 2) Network systems (Dijkstra optimizations), 3) Blockchain (Merkle trees).

Critically, complexity theory provides the framework for evaluating emerging technologies. This unit's greatest gift is the ability to quantitatively justify architectural decisions to stakeholders.

If I did this unit again, I would do the following things differently:

If I were to do this unit again, I'd implement three fundamental changes to maximize learning. First, I'd tackle Distinction tasks early.

Second, I'd establish a cross-functional study group within Week 1, meeting biweekly to debug challenges like Task 7.1P's rotation edge cases collectively rather than struggling solo for weeks. Most crucially, I'd seek Sergey's guidance on mathematical proofs earlier, recognizing that formal complexity analysis (like recurrence relation solving) ultimately saves implementation time. These adjustments would transform competent completion into mastery.

Deakin University

Data Structures and Algorithms

Yi Guan

Portfolio Submission

Submitted By:
Yi Guan
ppv

Tutor:
Sergey Polyakovskiy

June 5, 2025

Contents

2 Overall Task Status

Task	Status	Times Assessed
Computing the minimum number of deques: A problem-solving task	Not Started	
Implementation of the Doubly Linked List	Complete	1
Enrolment Form	Complete	1
Vector: A simple dynamic collection of elements	Complete	2
Helping your peers	Needs Improvement	1
Enabling sorting of generic data	Complete	1
Basic questions on algorithmic complexity	Complete	1
Advanced questions on algorithmic complexity	Complete	1
Implementation of simple sorting algorithms	Complete	2
Implementation of advanced sorting algorithms	Complete	1
Implementation of the Binary Search algorithm	Complete	1
Exploring the Skip-List data structure	Complete	4
Stacks, heaps, and Elon Musk	Complete	1
AVL trees	Complete	1
Graphs, paths, and Miss Marple	Complete	2
Rescuing Mario's black cat	Not Started	0
Implementation of the Binary Heap	Complete	1
A few advanced questions on complexity analysis and algorithm design	Discuss	2
Helping Alex with treasure hunting: A problem-solving task	Discuss	1
Determining the sorting rule: A problem-solving task	Not Started	0
Special coins: A problem-solving task	Not Started	

3 Learning Outcomes

3.1 Complexity

Evaluate the memory usage and computational complexity of different solution strategies and use this to provide recommendations in terms of solution direction for given problem scenarios.

Task	Rating	Status	Times Assessed
Enrolment Form	♦♦♦♦♦	Complete	1
Implementation of the Doubly Linked List	♦♦♦♦♦	Complete	1
Vector: A simple dynamic collection of elements	♦♦♦♦♦	Complete	2
Helping your peers	♦♦♦◊◊	Needs Improvement	1
Enabling sorting of generic data	♦♦♦♦♦	Complete	1
Basic questions on algorithmic complexity	♦♦♦♦♦	Complete	1
Advanced questions on algorithmic complexity	♦♦♦♦♦	Complete	1
Implementation of simple sorting algorithms	♦♦♦♦♦	Complete	2
Implementation of advanced sorting algorithms	♦♦♦♦♦	Complete	1
Implementation of the Binary Search algorithm	♦♦♦♦♦	Complete	1
Exploring the Skip-List data structure	♦♦♦♦♦	Complete	4
Stacks, heaps, and Elon Musk	♦♦♦♦♦	Complete	1
AVL trees	♦♦♦♦♦	Complete	1
Graphs, paths, and Miss Marple	♦♦♦♦♦	Complete	2
Implementation of the Binary Heap	♦♦♦♦♦	Complete	1
A few advanced questions on complexity analysis and algorithm design	♦♦♦♦♦	Discuss	2
Helping Alex with treasure hunting: A problem-solving task	♦♦♦♦♦	Discuss	1

3.2 Implement Solutions

Create and use a range of data structures and algorithms to design solutions and implement programs that address specified requirements and constraints

Task	Rating	Status	Times Assessed
Enrolment Form	♦♦♦♦♦	Complete	1
Implementation of the Doubly Linked List	♦♦♦♦♦	Complete	1
Vector: A simple dynamic collection of elements	♦♦♦♦♦	Complete	2
Helping your peers	♦♦♦◊◊	Needs Improvement	1
Enabling sorting of generic data	♦♦♦♦♦	Complete	1
Basic questions on algorithmic complexity	♦♦♦♦♦	Complete	1
Advanced questions on algorithmic complexity	♦♦♦♦♦	Complete	1
Implementation of simple sorting algorithms	♦♦♦♦♦	Complete	2
Implementation of advanced sorting algorithms	♦♦♦♦♦	Complete	1
Implementation of the Binary Search algorithm	♦♦♦♦♦	Complete	1
Exploring the Skip-List data structure	♦♦♦♦♦	Complete	4
Stacks, heaps, and Elon Musk	♦♦♦♦♦	Complete	1
AVL trees	♦♦♦♦♦	Complete	1
Graphs, paths, and Miss Marple	♦♦♦♦♦	Complete	2
Implementation of the Binary Heap	♦♦♦♦♦	Complete	1
A few advanced questions on complexity analysis and algorithm design	♦♦♦♦♦	Discuss	2
Helping Alex with treasure hunting: A problem-solving task	♦♦♦♦♦	Discuss	1

3.3 Document solutions

Document problem and solution constraints, design decisions, and trade-offs involved in creating software solutions for a given problem.

Task	Rating	Status	Times Assessed
Enrolment Form	♦♦♦♦♦	Complete	1
Implementation of the Doubly Linked List	♦♦♦♦♦	Complete	1
Vector: A simple dynamic collection of elements	♦♦♦♦♦	Complete	2
Helping your peers	♦♦♦◊◊	Needs Improvement	1
Enabling sorting of generic data	♦♦♦♦♦	Complete	1
Basic questions on algorithmic complexity	♦♦♦♦♦	Complete	1
Advanced questions on algorithmic complexity	♦♦♦♦♦	Complete	1
Implementation of simple sorting algorithms	♦♦♦♦♦	Complete	2
Implementation of advanced sorting algorithms	♦♦♦♦♦	Complete	1
Implementation of the Binary Search algorithm	♦♦♦♦♦	Complete	1
Exploring the Skip-List data structure	♦♦♦♦♦	Complete	4
Stacks, heaps, and Elon Musk	♦♦♦♦♦	Complete	1
AVL trees	♦♦♦♦♦	Complete	1
Graphs, paths, and Miss Marple	♦♦♦♦♦	Complete	2
Implementation of the Binary Heap	♦♦♦♦♦	Complete	1
A few advanced questions on complexity analysis and algorithm design	♦♦♦♦♦	Discuss	2
Helping Alex with treasure hunting: A problem-solving task	♦♦♦♦♦	Discuss	1

4 Additional Files

1. VIDEO_LINK_PDF.pdf

Video link for my learning summary report:

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=5e9e34ef-af7b-41ed-9284-b2f20116604f

5 Enrolment Form

After reading the enclosed document, this task requires you to complete the attached form and submit it to OnTrack. Teaching staff will not consider any of your submissions until this pre-pass task is completed.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/03/13 00:13	Yi Guan	Ready to Mark
2025/03/13 00:13	Yi Guan	hihi =w=
2025/03/17 14:00	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Enrolment Form

Submitted By:
Yi Guan
ppv
2025/03/13 00:13

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦♦♦
Implement Solutions	♦♦♦♦♦
Document solutions	♦♦♦♦♦

alg

March 13, 2025

Practical Task 0.1

(Pass Task)

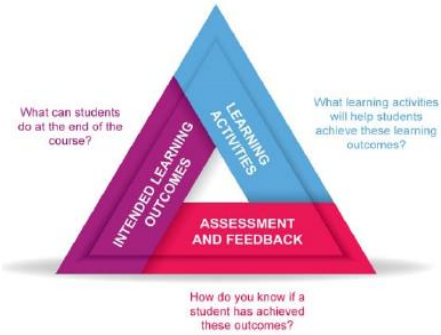
Submission deadline: Monday, March 17

General Instructions

The objective of this task is to ensure that you are familiar with the assessment protocols and processes being used this trimester during SIT221 Data Structures and Algorithms. You must read this document and fill the form at the end, then submit the signed form to OnTrack.

Assessment Model

SIT221 Data Structures and Algorithms has a reputation for being a challenging and enjoyable unit. To help students tailor this unit to achieve the outcomes they are after, SIT221 uses a teaching approach referred to as Constructive Alignment. Constructivism views knowledge as being constructed in the mind of the learner. Therefore, it is important for students to actively perform the required tasks to learn. As a result, the role of the educator changes from someone who "teaches" to someone who "facilitates" learning.



In this unit, this Constructive Alignment is achieved via OnTrack. OnTrack is an in-house developed online software platform that facilitates this formative learning approach via a task-oriented portfolio assessment. The strategy is to organise learning activities across multiple tasks structured around grade outcomes. You need to select the grade you are aiming to achieve. Your decision can be based on your prior experience in the domain, your aptitude towards the topic, your time commitment, or the requirements of your career objective. This choice allows you to scaffold your own learning to achieve greater depth than you previously may have thought was possible.

In general, the graded tasks will provide the following challenge levels:

- **Pass**. This offers scaffolded tasks to help achieve the minimum acceptable standard. Students will be able to implement object-oriented solutions that make use of abstraction, encapsulation, inheritance, and polymorphism if given a detailed design to work from.
- **Credit**. Students will apply what they have learnt in the pass tasks to new problems with less guidance. Students will be able to implement object-oriented designs and propose changes to designs to meet changing requirements.
- **Distinction**. Students will apply their advanced knowledge to design and build solutions to more challenging real-world scenarios. Students will be able to design and develop object-oriented programs to ill-defined problems.
- **High Distinction**. Students will extend their understanding to demonstrate greater technical ability, more complex solution structures, advanced algorithms, or in other ways exceed the expectations of the unit.

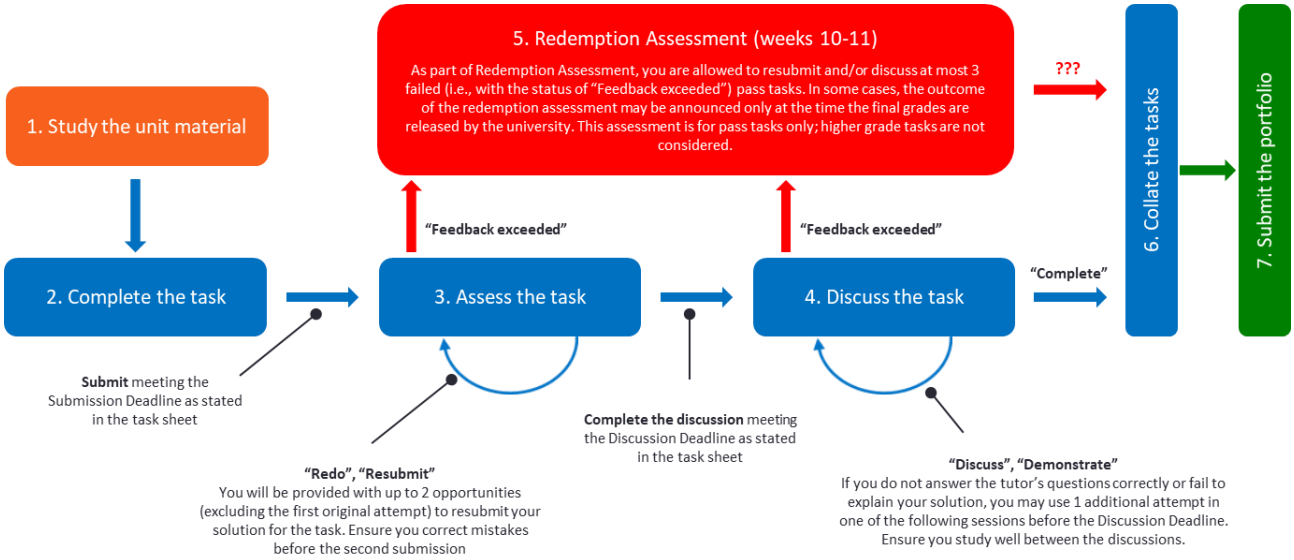
For this unit, the following will set the minimum standard for each grade:

- **Pass**: Complete (and **discuss** with the teaching staff) all pass tasks to **guarantee** a pass in this unit.
- **Credit**: Complete all pass and credit tasks.
- **Distinction**: Complete all pass, credit, and distinction tasks.
- **High Distinction**: Complete all pass, credit, distinction, and at least one high distinction task.

Your final grade is based on the number of tasks that you complete at each task level. Students aiming for higher grades will need to succeed with more tasks. It is recommended that you seriously and honestly consider your capabilities. Genuinely advanced students will find pass tasks quick and easy to complete. If you find pass tasks difficult and/or too time-consuming, then you should adjust your expectations. If you do not complete required tasks at a level, we may accept higher grade tasks as a replacement for lower grade ones where such tasks meet the requirements and expected learning outcomes.

Assessment Process

Knowledge and skills in this unit continuously build on those learnt the weeks before. Therefore, if you fall behind, it becomes impossible to understand the subsequent content. Tasks are spread throughout the trimester and need to be done in sequence and on time. To complete a task, students must follow the formative learning process illustrated in the process diagram below. This is designed to ensure your educators can provide a reasonable level of support to all students.



The following describes the steps to be followed during this unit.

1. **Study material**: Ensure you attend/watch lectures and explore the associated unit materials, including the "Further Notes" and "Submission Instructions" sections of each task sheet.
2. **Complete the task**: During your lecture (lab) time, your lecturer (tutor) will provide some general advice and answer questions to clarify the task's requirements. They will not provide solutions, but may give you some valuable hints; thus, attending lectures and practical sessions regularly is important to get insights on how to solve the assessment tasks. You may also seek peer support by posting questions in the Online Peer-to-Peer Study Group in MS Teams (Click to navigate). Once you believe you have a correct solution, you can submit it via OnTrack (https://ontrack.deakin.edu.au/) for assessment.
3. **Get the solution assessed**: Your tutor will check your submission and provide feedback via the OnTrack chat facility. This will either inform you that your solution is correct or will identify where it requires additional work. If it is correct, it will be flagged as "Demonstrate" or "Discuss" and you can go to step 4. If it requires additional work, the tutor will mark the task as either "Redo" or "Resubmit". You will usually get one additional week to revise your submission to incorporate the tutor's feedback. Once you are sure you have addressed the tutor's concerns, you can resubmit. The tutor is only expected to accept **2 resubmissions**. Then, the tutor may decide (i.e., it is up to the tutor's discretion and is not a must) to give you a very limited time to make minor changes and submit again to meet the expected standards. If the tutor does not accept your submission, then it will be marked as "Feedback Exceeded". If a task is marked as "Feedback Exceeded", you have to proceed to step 5 in weeks 10-11.

Note that **solutions submitted after the submission deadline** (and thus getting the status of "Time exceeded") **will never be considered**. If you fail the submission deadline, you fail the task and this reduces the chance to pass the unit. Keep in mind that all deadlines are strict to guarantee smooth and on-time work throughout the unit.

Solutions submitted as blank / empty files will be marked as failed submissions immediately. Plagiarised tasks or purchased solutions will be reported and investigated by the academic integrity committee.

4. **Discuss (or Demonstrate) the solution**: If your solution has been accepted as correct in step 3, you must also demonstrate to the tutor your individual learning and understanding of the associated concepts through a one-on-one interview. Before the discussion deadline (see the header of the respective task sheet or check the effective due dates in SIT221 MS Teams channel), you must either:

   - Attend the practical class either via Microsoft Teams (cloud mode) or in person (on-campus mode) and have a one-on-one conversation with your tutor. In the former case, ensure you have a working camera and a microphone (Teams has a mobile application available if you do not have a webcam). When you attend the practical class, inform the tutor via a chat message that you are ready for an interview and wait for the tutor to get to you.
   - If the task permits, you may record a short (5-10 minutes) video explaining your work and solution to the task. Upload the video to one of accessible resources (e.g., Youtube or Microsoft Sharepoint) and refer to it for the purpose of marking. You must provide a working link to the video to the tutor in the respective textbox in OnTrack. A video recording, when it is allowed to replace a live discussion of the task, must be made in the **camera on mode**; that is, the tutor must see both the presenter and the shared screen. Then, your tutor may record several audio questions. You may use the Intelligent Discussion Facility in OnTrack to answer the additional questions. When you click on these, OnTrack will record your response live. You must answer straight away in your own words. As this is a live response, you should ensure you understand the solution to the task you submitted. If the tutor finds your video recording and/or online explanation unsatisfactory, he/she may still ask you to attend a practical class to pass a one-on-one interview.

   Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this **compulsory interview part**. Please, come prepared so that the class time is used efficiently and fairly for all students in it. You should start your interview as soon as possible as if your answers are wrong or incomplete, you may have to pass another interview, still before the deadline. Use available attempts properly. The tutor is only expected to give you **2 opportunities to discuss your task** (including the original attempt). If eventually the tutor does not accept your answers, the task will be marked as "Feedback Exceeded"; see step 5 for the possible actions.

   Again, the deadlines are strict. No solutions will be generally discussed after the discussion deadline. Uncompleted solutions will be marked as "Feedback Exceeded". If you fail the deadline, you fail the task, that reduces the chance to pass the unit.

   If the tutor is satisfied with your answers, the task will be marked as "Complete".

5. **Redemption assessment**: If your pass task is marked as "Feedback Exceeded" or "Time Exceeded" (and **only if** the first submission is made before the submission deadline), you can still obtain a pass in the unit by passing an additional assessment for this task during the weeks 10 and 11. This option is only provided for pass tasks. If the redemption assessment is possible, your tutor will indicate this via a comment in the chat box in OnTrack. The redemption assessment can have different forms and depends on your particular situation; this will be decided by your tutor and the unit chair case by case. You can be given an additional opportunity to discuss the failed tasks or may need to pass a special interview covering the topics related to such tasks. In the latter case, all further video communications will be recorded for a reference and the outcome will be announced only with the final grade for the unit. You will be allowed to pass this assessment for **no more than 3 failed pass tasks** regardless the total number of failed tasks.

6. **Collate all tasks**: At the end of week 11, you will need to prepare your final portfolio for submission. This will involve filling in a cover sheet, indicating the number of tasks you have completed at each level, and justifying the learning that you have accomplished. The aim of this is to complete a self-reflection on your study and argue for the grade you believe you should obtain.

Possible Extensions

The following lists the facts about possible extensions in the unit.

1. Only individual extensions are available to students dealing with short-term, unexpected setbacks (e.g., medical conditions, hardship or trauma, which might be caused, for example, by a sudden change in employment circumstances, severe disruption to domestic arrangements, or when the student is the victim of a serious crime). Every application must be accompanied with evidence supporting it (e.g., a medical certificate or purchased tickets to/from your home country).
2. The full-time employment status is not seen as a proper ground for an extension by the university. Therefore, the unit chair is not allowed to grant extra time.
3. The unit chair is authorised to grant extensions of up to 7 days long (14 days, if the student has a suitable DRC access plan). Any longer extensions can only be obtained through the special consideration process (https://www.deakin.edu.au/students/study-support/assessments-and-examinations/special-consideration).

Submission of This Task

Having read the above discussion, this task requires you to complete the following form and submit it to OnTrack. Once submitted, your tutor will initiate an Intelligent Discussion with you which you will need to provide a response. This discussion will only be asking you to introduce yourself to the tutor. If this pre-pass task is not completed, we will not be accepting any future task submissions.

Student ID: 221393284

Name: Jack Guan

☐ Tick to indicate you have read this task sheet, asked your tutor anything you need clarified, and that you understand what you need to do to pass this unit.

Signed: Jack Guan

Date: 11/03/2025

6 Vector: A simple dynamic collection of elements

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Complexity	♦♦♦♦♦

Not hard, but not easy either. :(

Outcome	**Weight**
Implement Solutions	♦♦♦♦♦

Not hard, but not easy either. :(

Outcome	**Weight**
Document solutions	♦♦♦♦♦

Not hard, but not easy either. :(

Date	Author	Comment
2025/03/13 23:52	Yi Guan	Ready to Mark
2025/03/13 23:52	Yi Guan	hihi
2025/03/16 00:06	Svitlana Prypoten	Hi Yi,Good start, but some modifications are needed.In Clear() you do not remove elements you just set Count to 0. Please refer tohttps://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.clear?view=net-9.0#system-collections-generic-list-1-clearNot only count should be set to 0, but any references held by the collection'selements are cleared. Same issue with Count– in RemoveAt() Please resubmit once fixed
2025/03/16 00:06	Svitlana Prypoten	Fix and Resubmit
2025/03/16 14:57	Yi Guan	Got it Sir, I'll do it later today. :)
2025/03/16 16:28	Yi Guan	Ready to Mark
2025/03/16 16:28	Yi Guan	Hi, Sir. I have fixed my code.
2025/03/16 16:28	Yi Guan	image comment
2025/03/16 16:28	Yi Guan	image comment
2025/03/16 16:29	Yi Guan	:sweat_smile:Hope this time will be all right
2025/03/16 23:10	Svitlana Prypoten	You have correct fix for RemoveAt(), but not right for Clear(). Anyway, happy to discuss on our session, but be prepared to explain what should be the fix for Clear()
2025/03/16 23:10	Svitlana Prypoten	Discuss
2025/03/17 01:08	Yi Guan	Okay..... What time will be the discuss session running this week? Is it on Monday 18 pm?
2025/03/18 23:09	Svitlana Prypoten	Yes it's Monday 18.00-20.00pm
2025/03/18 23:10	Svitlana Prypoten	You also can upload a video, but please make sure you cover fixes for Clear()
2025/03/19 19:35	Yi Guan	cool, and here is the video link for it. https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=da 2168-40f6-ba01-b2a5008b5d2b
2025/03/19 20:19	Svitlana Prypoten	I don't have access to your video
2025/03/20 14:50	Yi Guan	Sorry, I forgot to enable the share setting to public, this time should be all right . https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=da 2168-40f6-ba01-b2a5008b5d2b
2025/03/26 12:25	Svitlana Prypoten	Hi Jack, I will accept video this time, but for future, please make sure you have you camera on, as this is one of requirements.Overall all good. You have an issue in removeAt() around sequence of actions. You should set value to default first and t hen do Count–My question for you: Is it necessary to have if(index==Count) in Insert() and what is the benefit or drawback to have it.
2025/03/27 11:47	Yi Guan	I would say it might not be necessary, but it can give clear instructions on adding a new element, and it's easier to read and understand. It also uses the same logic as the Add() method, which reduces duplication.
2025/03/27 11:48	Yi Guan	:sweat_smile:Thank you Miss, i got it
2025/03/29 09:50	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Vector: A simple dynamic collection of elements

Submitted By:
Yi Guan
ppv
2025/03/16 16:28

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

Not hard, but not easy either. :(

March 16, 2025

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Vector
8  {
9      public class Vector<T>
10     {
11         // This constant determines the default number of elements in a newly
           ↪  created vector.
12         // It is also used to extended the capacity of the existing vector
13         private const int DEFAULT_CAPACITY = 10;
14
15         // This array represents the internal data structure wrapped by the vector
           ↪  class.
16         // In fact, all the elements are to be stored in this private  array.
17         // You will just write extra functionality (methods) to make the work with
           ↪  the array more convenient for the user.
18         private T[] data;
19
20         // This property represents the number of elements in the vector
21         public int Count { get; private set; } = 0;
22
23         // This property represents the maximum number of elements (capacity) in the
           ↪  vector
24         public int Capacity { get; private set; } = 0;
25
26         // This is an overloaded constructor
27         public Vector(int capacity)
28         {
29             data = new T[capacity];
30         }
31
32         // This is the implementation of the default constructor
33         public Vector() : this(DEFAULT_CAPACITY) { }
34
35         // An Indexer is a special type of property that allows a class or structure
           ↪  to be accessed the same way as array for its internal collection.
36         // For example, introducing the following indexer you may address an element
           ↪  of the vector as vector[i] or vector[0] or ...
37         public T this[int index]
38         {
39             get
40             {
41                 if (index >= Count || index < 0) throw new
                   ↪  IndexOutOfRangeException();
42                 return data[index];
43             }
```

```
44              set
45              {
46                  if (index >= Count || index < 0) throw new
                    ↪  IndexOutOfRangeException();
47                  data[index] = value;
48              }
49          }
50
51          // This private method allows extension of the existing capacity of the
            ↪  vector by another 'extraCapacity' elements.
52          // The new capacity is equal to the existing one plus 'extraCapacity'.
53          // It copies the elements of 'data' (the existing array) to 'newData' (the
            ↪  new array), and then makes data pointing to 'newData'.
54          private void ExtendData(int extraCapacity)
55          {
56              T[] newData = new T[data.Length + extraCapacity];
57              for (int i = 0; i < Count; i++) newData[i] = data[i];
58              data = newData;
59          }
60
61          // This method adds a new element to the existing array.
62          // If the internal array is out of capacity, its capacity is first extended
            ↪  to fit the new element.
63          public void Add(T element)
64          {
65              if (Count == data.Length) ExtendData(DEFAULT_CAPACITY);
66              data[Count++] = element;
67          }
68
69          // This method searches for the specified object and returns the zero-based
            ↪  index of the first occurrence within the entire data structure.
70          // This method performs a linear search; therefore, this method is an O(n)
            ↪  runtime complexity operation.
71          // If occurrence is not found, then the method returns -1.
72          // Note that Equals is the proper method to compare two objects for
            ↪  equality, you must not use operator '=' for this purpose.
73          public int IndexOf(T element)
74          {
75              for (var i = 0; i < Count; i++)
76              {
77                  if (data[i].Equals(element)) return i;
78              }
79              return -1;
80          }
81
82          // TODO:**************************************************************
            ↪  ************************
83          // TODO: Your task is to implement all the remaining methods.
84          // Read the instruction carefully, study the code examples from above as
            ↪  they should help you to write the rest of the code.
85          public void Insert(int index, T element)
86          {
87              if (index > Count || index < 0) throw new IndexOutOfRangeException();
88              if (Count == Capacity) ExtendData(DEFAULT_CAPACITY);
```

```
89              if (index == Count)
90              {
91                  Add(element);
92                  return;
93              }
94              for (int i = Count; i > index; i--)
95              {
96                  data[i] = data[i - 1];
97              }
98              data[index] = element;
99              Count++;
100         }
101
102         public void Clear()
103         {
104             //Count = 0;
105             //data = new T[DEFAULT_CAPACITY];
106
107             Array.Clear(data, 0, Count);// this code will clean the references in
                ↪  array
108             Count = 0;  // set the count to 0
109
110         }
111
112         public bool Contains(T element)
113         {
114             return IndexOf(element) != -1;
115         }
116
117         public bool Remove(T element)
118         {
119             int index = IndexOf(element);
120             if (index != -1)
121             {
122                 RemoveAt(index);
123                 return true;
124             }
125             return false;
126         }
127
128         public void RemoveAt(int index)
129         {
130             if (index >= Count || index < 0) throw new IndexOutOfRangeException();
131             for (int i = index; i < Count - 1; i++)
132             {
133                 data[i] = data[i + 1];
134             }
135             Count--;
136
137             data[Count] = default(T);
138             // To clear the redundant references, recycle the data that no longer
                ↪  used.
139
140         }
```

```
141
142         public override string ToString()
143         {
144             StringBuilder sb = new StringBuilder();
145             sb.Append("[");
146             for (int i = 0; i < Count; i++)
147             {
148                 sb.Append(data[i]?.ToString());
149                 if (i < Count - 1) sb.Append(", ");
150             }
151             sb.Append("]");
152             return sb.ToString();
153         }
154
155
156
157
158     }
159     class Tester
160     {
161
162         private static bool CheckIntSequence(int[] certificate, Vector<int> vector)
163         {
164             if (certificate.Length != vector.Count) return false;
165             for (int i = 0; i < certificate.Length; i++)
166             {
167                 if (certificate[i] != vector[i]) return false;
168             }
169             return true;
170         }
171
172         static void Main(string[] args)
173         {
174             Vector<int> vector = null;
175             string result = "";
176
177             // test 1
178             try
179             {
180                 Console.WriteLine("\nTest A: Create a new vector by calling
                    ↪  'Vector<int> vector = new Vector<int>(50);'");
181                 vector = new Vector<int>(50);
182                 Console.WriteLine(" :: SUCCESS");
183                 result = result + "A";
184             }
185             catch (Exception exception)
186             {
187                 Console.WriteLine(" :: FAIL");
188                 Console.WriteLine(exception.ToString());
189                 result = result + "-";
190             }
191
192             // test 2
193             try
```

```
194             {
195                 Console.WriteLine("\nTest B: Add a sequence of numbers 2, 6, 8, 5,
                    ↪  5, 1, 8, 5, 3, 5");
196                 vector.Add(2); vector.Add(6); vector.Add(8); vector.Add(5);
                    ↪  vector.Add(5); vector.Add(1); vector.Add(8); vector.Add(5);
                    ↪  vector.Add(3); vector.Ad
197 d(5);
198                 if (!CheckIntSequence(new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5 },
                    ↪  vector)) throw new Exception("Vector stores incorrect sequence
                    ↪  of integers");
199                 Console.WriteLine(" :: SUCCESS");
200                 result = result + "B";
201             }
202             catch (Exception exception)
203             {
204                 Console.WriteLine(" :: FAIL");
205                 Console.WriteLine(exception.ToString());
206                 result = result + "-";
207             }
208 
209             // test 3
210             try
211             {
212                 Console.WriteLine("\nTest C: Remove number 3, 7, and then 6");
213                 bool check = vector.Remove(3) && (!vector.Remove(7)) &&
                    ↪  vector.Remove(6);
214                 if (!CheckIntSequence(new int[] { 2, 8, 5, 5, 1, 8, 5, 5 },
                    ↪  vector)) throw new Exception("Vector stores incorrect sequence
                    ↪  of integers");
215                 Console.WriteLine(" :: SUCCESS");
216                 Console.WriteLine(vector.ToString());
217                 result = result + "C";
218             }
219             catch (Exception exception)
220             {
221                 Console.WriteLine(" :: FAIL");
222                 Console.WriteLine(exception.ToString());
223                 result = result + "-";
224             }
225 
226             // test 4
227             try
228             {
229                 Console.WriteLine("\nTest D: Insert number 50 at index 6, then
                    ↪  number 0 at index 0, then number 60 at index 'vector.Count-1',
                    ↪  then number 70 at
230 index 'vector.Count'");
231                 vector.Insert(6, 50); vector.Insert(0, 0);
                    ↪  vector.Insert(vector.Count - 1, 60);
                    ↪  vector.Insert(vector.Count, 70);
232                 if (!CheckIntSequence(new int[] { 0, 2, 8, 5, 5, 1, 8, 50, 5, 60,
                    ↪  5, 70 }, vector)) throw new Exception("Vector stores incorrect
                    ↪  sequence of int
233 egers");
```

```
234                 Console.WriteLine(" :: SUCCESS");
235                 Console.WriteLine(vector.ToString());
236                 result = result + "D";
237             }
238             catch (Exception exception)
239             {
240                 Console.WriteLine(" :: FAIL");
241                 Console.WriteLine(exception.ToString());
242                 result = result + "-";
243             }
244 
245 
246             // test 5
247             try
248             {
249                 Console.WriteLine("\nTest E: Insert number -1 at index
                    ↪  'vector.Count+1'");
250                 vector.Insert(vector.Count + 1, -1);
251 
252             }
253             catch (IndexOutOfRangeException exception)
254             {
255                 Console.WriteLine(" :: SUCCESS");
256                 result = result + "E";
257             }
258             catch (Exception exception)
259             {
260                 Console.WriteLine(" :: FAIL");
261                 Console.WriteLine("Last operation is invalid and must throw
                    ↪  IndexOutOfRangeException. Your solution does not match
                    ↪  specification.");
262                 result = result + "-";
263             }
264             Console.WriteLine(vector);
265 
266             // test 6
267             try
268             {
269                 Console.WriteLine("\nTest F: Remove number at index 4, then number
                    ↪  index 0, and then number at index 'vector.Count-1'");
270                 vector.RemoveAt(4); vector.RemoveAt(0);
                    ↪  vector.RemoveAt(vector.Count - 1);
271                 if (!CheckIntSequence(new int[] { 2, 8, 5, 1, 8, 50, 5, 60, 5 },
                    ↪  vector)) throw new Exception("Vector stores incorrect sequence
                    ↪  of integers");
272                 Console.WriteLine(" :: SUCCESS");
273                 result = result + "F";
274             }
275             catch (Exception exception)
276             {
277                 Console.WriteLine(" :: FAIL");
278                 Console.WriteLine(exception.ToString());
279                 result = result + "-";
280             }
```

```
281             Console.WriteLine(vector);
282
283             // test 7
284             try
285             {
286                 Console.WriteLine("\nTest G: Remove number at index
                    ↪  'vector.Count'");
287                 vector.RemoveAt(vector.Count);
288             }
289             catch (IndexOutOfRangeException exception)
290             {
291                 Console.WriteLine(" :: SUCCESS");
292                 result = result + "G";
293             }
294             catch (Exception exception)
295             {
296                 Console.WriteLine(" :: FAIL");
297                 Console.WriteLine("Last operation is invalid and must throw
                    ↪  IndexOutOfRangeException. Your solution does not match
                    ↪  specification.");
298                 result = result + "-";
299             }
300
301             // test 8
302             try
303             {
304                 Console.WriteLine("\nTest H: Run a sequence of operations: ");
305
306                 Console.WriteLine("vector.Contains(1);");
307                 if (vector.Contains(1)) Console.WriteLine(" :: SUCCESS");
308                 else throw new Exception("1 must be in the vector");
309
310                 Console.WriteLine("vector.Contains(2);");
311                 if (vector.Contains(2)) Console.WriteLine(" :: SUCCESS");
312                 else throw new Exception("2 must be in the vector");
313
314                 Console.WriteLine("vector.Contains(4);");
315                 if (!vector.Contains(4)) Console.WriteLine(" :: SUCCESS");
316                 else throw new Exception("4 must not be in the vector");
317
318                 Console.WriteLine("vector.Add(4); vector.Contains(4);");
319                 vector.Add(4);
320                 if (vector.Contains(4)) Console.WriteLine(" :: SUCCESS");
321                 else throw new Exception("4 must be in the vector");
322
323                 result = result + "H";
324             }
325             catch (Exception exception)
326             {
327                 Console.WriteLine(" :: FAIL");
328                 Console.WriteLine(exception.ToString());
329                 result = result + "-";
330             }
331
```

```
332             // test 9
333             try
334             {
335                 Console.WriteLine("\nTest I: Print the content of the vector via
                    ↪  calling vector.ToString();");
336                 Console.WriteLine(vector.ToString());
337                 Console.WriteLine(" :: SUCCESS");
338                 result = result + "I";
339             }
340             catch (Exception exception)
341             {
342                 Console.WriteLine(" :: FAIL");
343                 Console.WriteLine(exception.ToString());
344                 result = result + "-";
345             }
346 
347             // test 10
348             try
349             {
350                 Console.WriteLine("\nTest J: Clear the content of the vector via
                    ↪  calling vector.Clear();");
351                 vector.Clear();
352                 if (!CheckIntSequence(new int[] { }, vector)) throw new
                    ↪  Exception("Vector stores incorrect data. It must be empty.");
353                 Console.WriteLine(" :: SUCCESS");
354                 result = result + "J";
355             }
356             catch (Exception exception)
357             {
358                 Console.WriteLine(" :: FAIL");
359                 Console.WriteLine(exception.ToString());
360                 result = result + "-";
361             }
362 
363             Console.WriteLine("\n\n ------------------- SUMMARY -------------------
                ↪  ");
364             Console.WriteLine("Tests passed: " + result);
365             Console.ReadKey();
366         }
367     }
368 }
```

7 Enabling sorting of generic data

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/03/19 20:16	Yi Guan	Ready to Mark
2025/03/19 20:16	Yi Guan	hihi
2025/03/21 21:58	Svitlana Prypoten	Happy to discuss or waiting for video
2025/03/21 21:58	Svitlana Prypoten	Discuss
2025/03/31 19:49	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Enabling sorting of generic data

Submitted By:
Yi Guan
ppv
2025/03/19 20:16

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

March 19, 2025

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vector
{
    public class Vector<T>
    {
        private const int DEFAULT_CAPACITY = 10;
        private T[] data;

        public int Count { get; private set; } = 0;

        public int Capacity { get; private set; } = 0;

        public Vector(int capacity)
        {
            data = new T[capacity];
            Capacity = capacity;
        }
        public Vector() : this(DEFAULT_CAPACITY) { }
        public T this[int index]
        {
            get
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                return data[index];
            }
            set
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }
        private void ExtendData(int extraCapacity)
        {
            T[] newData = new T[data.Length + extraCapacity];
            for (int i = 0; i < Count; i++)
                newData[i] = data[i];
            data = newData;
            Capacity = data.Length;
        }
        public void Add(T element)
        {
            if (Count == data.Length)
                ExtendData(DEFAULT_CAPACITY);
            data[Count++] = element;
        }
        public int IndexOf(T element)
        {
```

```
55              for (int i = 0; i < Count; i++)
56              {
57                  if (data[i].Equals(element))
58                      return i;
59              }
60              return -1;
61          }
62
63
64          public void Insert(int index, T element)
65          {
66              if (index < 0 || index > Count)
67                  throw new IndexOutOfRangeException();
68              if (Count == data.Length)
69                  ExtendData(DEFAULT_CAPACITY);
70              if (index == Count)
71              {
72                  Add(element);
73                  return;
74              }
75              for (int i = Count; i > index; i--)
76              {
77                  data[i] = data[i - 1];
78              }
79              data[index] = element;
80              Count++;
81          }
82
83
84          public void Clear()
85          {
86              Array.Clear(data, 0, Count);
87              Count = 0;
88          }
89
90
91          public bool Contains(T element)
92          {
93              return IndexOf(element) != -1;
94          }
95
96
97          public bool Remove(T element)
98          {
99              int index = IndexOf(element);
100             if (index != -1)
101             {
102                 RemoveAt(index);
103                 return true;
104             }
105             return false;
106         }
107
108
```

```
109         public void RemoveAt(int index)
110         {
111             if (index < 0 || index >= Count)
112                 throw new IndexOutOfRangeException();
113             for (int i = index; i < Count - 1; i++)
114             {
115                 data[i] = data[i + 1];
116             }
117             Count--;
118             data[Count] = default(T);
119         }
120 
121 
122         public override string ToString()
123         {
124             StringBuilder sb = new StringBuilder();
125             sb.Append("[");
126             for (int i = 0; i < Count; i++)
127             {
128                 sb.Append(data[i]?.ToString());
129                 if (i < Count - 1)
130                     sb.Append(", ");
131             }
132             sb.Append("]");
133             return sb.ToString();
134         }
135 
136 
137         public void Sort()
138         {
139             Array.Sort(data, 0, Count);
140         }
141 
142 
143         public void Sort(IComparer<T> comparer)
144         {
145             if (comparer == null)
146                 throw new ArgumentNullException(nameof(comparer));
147             Array.Sort(data, 0, Count, comparer);
148         }
149     }
150     public class AscendingIntComparer : IComparer<int>
151     {
152         public int Compare(int A, int B)
153         {
154             return A - B;
155         }
156     }
157 
158 
159     public class DescendingIntComparer : IComparer<int>
160     {
161         public int Compare(int A, int B)
162         {
```

```
163             return B - A;
164         }
165     }
166 
167 
168     public class EvenNumberFirstComparer : IComparer<int>
169     {
170         public int Compare(int A, int B)
171         {
172             return A % 2 - B % 2;
173         }
174     }
175 
176 
177     public class Student : IComparable<Student>
178     {
179         public string Name { get; set; }
180         public int Id { get; set; }
181 
182         public override string ToString()
183         {
184             return $"{Id}[{Name}]";
185         }
186 
187         public int CompareTo(Student other)
188         {
189             if (other == null) return 1;
190             return Id.CompareTo(other.Id);
191         }
192     }
193 
194 
195     public class AscendingIDComparer : IComparer<Student>
196     {
197         public int Compare(Student x, Student y)
198         {
199             if (x == null || y == null)
200                 throw new ArgumentNullException();
201             return x.Id.CompareTo(y.Id);
202         }
203     }
204 
205 
206     public class DescendingNameDescendingIdComparer : IComparer<Student>
207     {
208         public int Compare(Student x, Student y)
209         {
210             if (x == null || y == null)
211                 throw new ArgumentNullException();
212             int nameCompare = string.Compare(y.Name, x.Name,
                ↪ StringComparison.Ordinal);
213             if (nameCompare != 0)
214                 return nameCompare;
215             return y.Id.CompareTo(x.Id);
```

```
216             }
217         }
218 }
```

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Vector;
7
8  namespace _1._2P
9  {
10     class Tester
11     {
12         private static bool CheckIntSequence(int[] certificate, Vector<int> vector)
13         {
14             if (certificate.Length != vector.Count) return false;
15             for (int i = 0; i < certificate.Length; i++)
16             {
17                 if (certificate[i] != vector[i]) return false;
18             }
19             return true;
20         }
21
22         static void Main(string[] args)
23         {
24             Vector<int> vector = null;
25             string result = "";
26
27             // test 1
28             try
29             {
30                 Console.WriteLine("\nTest A: Run a sequence of operations: ");
31                 Console.WriteLine("Create a new vector by calling 'Vector<int>
                   ↪  vector = new Vector<int>(50);'");
32                 vector = new Vector<int>(50);
33                 Console.WriteLine("Add a sequence of numbers 2, 6, 8, 5, 5, 1, 8, 5,
                   ↪  3, 5, 7, 1, 4, 9");
34                 vector.Add(2); vector.Add(6); vector.Add(8); vector.Add(5);
                   ↪  vector.Add(5); vector.Add(1); vector.Add(8); vector.Add(5);
35                 vector.Add(3); vector.Add(5); vector.Add(7); vector.Add(1);
                   ↪  vector.Add(4); vector.Add(9);
36                 if (!CheckIntSequence(new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7,
                   ↪  1, 4, 9 }, vector)) throw new Exception("Vector stores an
                   ↪  incorrect sequence
37 of integers after adding new elements");
38                 Console.WriteLine("Sort the integers in the default order defined by
                   ↪  the native CompareTo() method");
39                 //uncomment the line to activate the test
40                 vector.Sort();
41                 int[] array = new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7, 1, 4, 9
                   ↪  };
42                 Array.Sort(array, 0, 14);
```

```
43                  Console.WriteLine("Resulting order: " + vector.ToString());
44                  if (!CheckIntSequence(array, vector)) throw new Exception("Vector
                    ↪  stores an incorrect sequence of integers after sorting them");
45                  Console.WriteLine(" :: SUCCESS");
46                  result = result + "A";
47              }
48              catch (Exception exception)
49              {
50                  Console.WriteLine(" :: FAIL");
51                  Console.WriteLine(exception.ToString());
52                  result = result + "-";
53              }
54
55              // test 2
56              try
57              {
58                  Console.WriteLine("\nTest B: Run a sequence of operations: ");
59                  Console.WriteLine("Create a new vector by calling 'Vector<int>
                    ↪  vector = new Vector<int>(50);'");
60                  vector = new Vector<int>(50);
61                  Console.WriteLine("Add a sequence of numbers 2, 6, 8, 5, 5, 1, 8, 5,
                    ↪  3, 5, 7, 1, 4, 9");
62                  vector.Add(2); vector.Add(6); vector.Add(8); vector.Add(5);
                    ↪  vector.Add(5); vector.Add(1); vector.Add(8);
63                  vector.Add(5); vector.Add(3); vector.Add(5); vector.Add(7);
                    ↪  vector.Add(1); vector.Add(4); vector.Add(9);
64                  if (!CheckIntSequence(new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7,
                    ↪  1, 4, 9 }, vector)) throw new Exception("Vector stores an
                    ↪  incorrect sequence
65  of integers after adding new elements");
66                  Console.WriteLine("Sort the integers in the order defined by the
                    ↪  AscendingIntComparer class");
67                  //uncomment the line to activate the test
68                  vector.Sort(new AscendingIntComparer());
69                  int[] array = new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7, 1, 4, 9
                    ↪  };
70                  Array.Sort(array, 0, 14, new AscendingIntComparer());
71                  Console.WriteLine("Resulting order: " + vector.ToString());
72                  if (!CheckIntSequence(array, vector)) throw new Exception("Vector
                    ↪  stores an incorrect sequence of integers after sorting them");
73                  Console.WriteLine(" :: SUCCESS");
74                  result = result + "B";
75              }
76              catch (Exception exception)
77              {
78                  Console.WriteLine(" :: FAIL");
79                  Console.WriteLine(exception.ToString());
80                  result = result + "-";
81              }
82
83              // test 3
84              try
85              {
86                  Console.WriteLine("\nTest C: Run a sequence of operations: ");
```

```
87                  Console.WriteLine("Create a new vector by calling 'Vector<int>
                    ↪  vector = new Vector<int>(50);'");
88                  vector = new Vector<int>(50);
89                  Console.WriteLine("Add a sequence of numbers 2, 6, 8, 5, 5, 1, 8, 5,
                    ↪  3, 5, 7, 1, 4, 9");
90                  vector.Add(2); vector.Add(6); vector.Add(8); vector.Add(5);
                    ↪  vector.Add(5); vector.Add(1); vector.Add(8);
91                  vector.Add(5); vector.Add(3); vector.Add(5); vector.Add(7);
                    ↪  vector.Add(1); vector.Add(4); vector.Add(9);
92                  if (!CheckIntSequence(new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7,
                    ↪  1, 4, 9 }, vector)) throw new Exception("Vector stores an
                    ↪  incorrect sequence
93  of integers after adding new elements");
94                  Console.WriteLine("Sort the integers in the order defined by the
                    ↪  DescendingIntComparer class");
95                  //uncomment the line to activate the test
96                  vector.Sort(new DescendingIntComparer());
97                  int[] array = new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7, 1, 4, 9
                    ↪  };
98                  Array.Sort(array, 0, 14, new DescendingIntComparer());
99                  Console.WriteLine("Resulting order: " + vector.ToString());
100                 if (!CheckIntSequence(array, vector)) throw new Exception("Vector
                    ↪  stores an incorrect sequence of integers after sorting them");
101                 Console.WriteLine(" :: SUCCESS");
102                 result = result + "C";
103             }
104             catch (Exception exception)
105             {
106                 Console.WriteLine(" :: FAIL");
107                 Console.WriteLine(exception.ToString());
108                 result = result + "-";
109             }
110
111             // test 4
112             try
113             {
114                 Console.WriteLine("\nTest D: Run a sequence of operations: ");
115                 Console.WriteLine("Create a new vector by calling 'Vector<int>
                    ↪  vector = new Vector<int>(50);'");
116                 vector = new Vector<int>(50);
117                 Console.WriteLine("Add a sequence of numbers 2, 6, 8, 5, 5, 1, 8, 5,
                    ↪  3, 5, 7, 1, 4, 9");
118                 vector.Add(2); vector.Add(6); vector.Add(8); vector.Add(5);
                    ↪  vector.Add(5); vector.Add(1); vector.Add(8);
119                 vector.Add(5); vector.Add(3); vector.Add(5); vector.Add(7);
                    ↪  vector.Add(1); vector.Add(4); vector.Add(9);
120                 if (!CheckIntSequence(new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7,
                    ↪  1, 4, 9 }, vector)) throw new Exception("Vector stores an
                    ↪  incorrect sequence
121 of integers after adding new elements");
122                 Console.WriteLine("Sort the integers in the order defined by the
                    ↪  EvenNumberFirstComparer class");
123                 //uncomment the line to activate the test
124                 vector.Sort(new EvenNumberFirstComparer());
```

```
125             int[] array = new int[] { 2, 6, 8, 5, 5, 1, 8, 5, 3, 5, 7, 1, 4, 9
                ↪  };
126             Array.Sort(array, 0, 14, new EvenNumberFirstComparer());
127             Console.WriteLine("Resulting order: " + vector.ToString());
128             if (!CheckIntSequence(array, vector)) throw new Exception("Vector
                ↪  stores an incorrect sequence of integers after sorting them");
129             Console.WriteLine(" :: SUCCESS");
130             result = result + "D";
131         }
132         catch (Exception exception)
133         {
134             Console.WriteLine(" :: FAIL");
135             Console.WriteLine(exception.ToString());
136             result = result + "-";
137         }
138 
139         // test 5
140         try
141         {
142             string[] names = new string[] { "Kelly", "Cindy", "John", "Andrew",
                ↪  "Richard", "Michael", "Guy", "Elicia", "Tom", "Iman", "Simon",
                ↪  "Vicky" };
143             Random random = new Random(100);
144             Console.WriteLine("\nTest E: Run a sequence of operations: ");
145             Console.WriteLine("Create a new vector of Student objects by calling
                ↪  'Vector<Student> students = new Vector<Student>();'");
146             Vector<Student> students = new Vector<Student>();
147             for (int i = 0; i < 14; i++)
148             {
149                 Student student = new Student() { Name = names[random.Next(0,
                    ↪  names.Length)], Id = i };
150                 Console.WriteLine("Add student with record: " +
                    ↪  student.ToString());
151                 students.Add(student);
152             }
153             Console.WriteLine("Sort the students in the default order defined by
                ↪  the native CompareTo() method");
154             //uncomment the line to activate the test
155             students.Sort();
156             Console.WriteLine("Print the vector of students via
                ↪  students.ToString();");
157             Console.WriteLine(students.ToString());
158 
159             Console.WriteLine(" :: SUCCESS");
160             result = result + "E";
161         }
162         catch (Exception exception)
163         {
164             Console.WriteLine(" :: FAIL");
165             Console.WriteLine(exception.ToString());
166             result = result + "-";
167         }
168 
169         // test 6
```

```
170             try
171             {
172                 string[] names = new string[] { "Kelly", "Cindy", "John", "Andrew",
                    ↪  "Richard", "Michael", "Guy", "Elicia", "Tom", "Iman", "Simon",
                    ↪  "Vicky" };
173                 Random random = new Random(100);
174                 Console.WriteLine("\nTest F: Run a sequence of operations: ");
175                 Console.WriteLine("Create a new vector of Student objects by calling
                    ↪  'Vector<Student> students = new Vector<Student>();'");
176                 Vector<Student> students = new Vector<Student>();
177                 for (int i = 0; i < 14; i++)
178                 {
179                     Student student = new Student() { Name = names[random.Next(0,
                        ↪  names.Length)], Id = i };
180                     Console.WriteLine("Add student with record: " +
                        ↪  student.ToString());
181                     students.Add(student);
182                 }
183                 Console.WriteLine("Sort the students in the order defined by the
                    ↪  AscendingIDComparer class");
184                 //uncomment the line to activate the test
185                 students.Sort(new AscendingIDComparer());
186                 Console.WriteLine("Print the vector of students via
                    ↪  students.ToString();");
187                 Console.WriteLine(students.ToString());
188
189                 Console.WriteLine(" :: SUCCESS");
190                 result = result + "F";
191             }
192             catch (Exception exception)
193             {
194                 Console.WriteLine(" :: FAIL");
195                 Console.WriteLine(exception.ToString());
196                 result = result + "-";
197             }
198
199             // test 7
200             try
201             {
202                 string[] names = new string[] { "Kelly", "Cindy", "John", "Andrew",
                    ↪  "Richard", "Michael", "Guy", "Elicia", "Tom", "Iman", "Simon",
                    ↪  "Vicky" };
203                 Random random = new Random(100);
204                 Console.WriteLine("\nTest G: Run a sequence of operations: ");
205                 Console.WriteLine("Create a new vector of Student objects by calling
                    ↪  'Vector<Student> students = new Vector<Student>();'");
206                 Vector<Student> students = new Vector<Student>();
207                 for (int i = 0; i < 14; i++)
208                 {
209                     Student student = new Student() { Name = names[random.Next(0,
                        ↪  names.Length)], Id = i };
210                     Console.WriteLine("Add student with record: " +
                        ↪  student.ToString());
211                     students.Add(student);
```

```
212                 }
213                 Console.WriteLine("Sort the students in the order defined by the
                    ↪  DescendingNameDescendingIdComparer class");
214                 //uncomment the line to activate the test
215                 students.Sort(new DescendingNameDescendingIdComparer());
216                 Console.WriteLine("Print the vector of students via
                    ↪  students.ToString();");
217                 Console.WriteLine(students.ToString());
218 
219                 Console.WriteLine(" :: SUCCESS");
220                 result = result + "G";
221             }
222             catch (Exception exception)
223             {
224                 Console.WriteLine(" :: FAIL");
225                 Console.WriteLine(exception.ToString());
226                 result = result + "-";
227             }
228 
229             Console.WriteLine("\n\n ------------------- SUMMARY -------------------
                ↪  ");
230             Console.WriteLine("Tests passed: " + result);
231             Console.ReadKey();
232         }
233     }
234 }
```

8 Basic questions on algorithmic complexity

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/03/25 13:46	Yi Guan	Ready to Mark
2025/03/25 13:46	Yi Guan	hihi
2025/03/26 17:02	Svitlana Prypoten	Hi Jack, you have 1 incorrect in section 3.Anyway happy to discuss or waiting for video.
2025/03/26 17:02	Svitlana Prypoten	Discuss
2025/03/27 11:49	Yi Guan	okay, I will double check it
2025/04/07 19:05	Svitlana Prypoten	Complete
2025/04/07 19:09	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Basic questions on algorithmic complexity

Submitted By:
Yi Guan
ppv
2025/03/25 13:46

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

March 25, 2025

Study the Introspective Sort, which is a default sorting algorithm implemented in the Array. Sort method. You must address the following questions:

1. What is special about the Introspective Sort?

   A： Introspective Sort combines QuickSort, HeapSort, and Insertion Sort. It starts with QuickSort, switches to HeapSort if recursion depth exceeds O(log n), and uses Insertion Sort for small subarrays for efficiency.

2. Does this method result in an unstable sorting; that is, if two elements are seen equal, might their order be not preserved after the data collection is sorted? Or does it perform a stable sorting, which preserves the order of elements that are seen equal?

   A： QuickSort and HeapSort are both unstable, so Introsort does not preserve the order of equal elements.

3. What is the time complexity of this sorting algorithm?

   A：
   Best: O(nlogn)
   Average: O(nlogn)
   Worst-case: O(nlogn)

In Task 1.1P on the Vector<T> class, you have completed/been provided with a number of methods (and properties), such as Count, Capacity, Add, IndexOf, Insert, Clear, Contains, Remove, and RemoveAt.Answer the following questions:

1. What is the time complexity of each of these operations?

   A:
   Count/Capacity: O(1)

   Add: O(1)

   IndexOf/Contains: O(n).

Insert/Remove/RemoveAt: O(n)

Clear: O(1)

2.Does your implementation match the complexity of the equivalent operations provided by the Microsoft .NET Framework for its List<T> collection?

A: Yes they are matches, because both use dynamic arrays with identical amortized complexities.

Answer and explain whether the following statements are right or wrong.

3.a. A $\theta(n^3)$ algorithm **always** takes longer to run than a $\theta(\log n)$ algorithm.

3.b. The best-case time complexity of the Bubble Sort algorithm is always $O(n)$.

3.c. The worst-case time complexity of the Insertion Sort algorithm is always $O(n^2)$.

3.d. The Selection Sort is an in-place sorting algorithm.

3.a: It's wrong, it only true for large n, not always.
3.b: It's right sorted input requires O(n) comparisons.
3.c: It's right, reversed input causes O(n^2) swaps.
3.d: It's right because it uses O(1) extra space.

3.e. The worst-case space complexity of the Insertion Sort algorithm is $O(1)$.

3.f. $2 + 2 = O(1)$

3.g. $n^3 + 10^6 n = O(n^3)$

3.h. $n log\, n = O(n)$

3.i. $log\, n = o(n)$

3.j. $log\, n + 2^{100} = \Theta(log\, n)$

3.k. $n\, log\, n = \Omega(n)$

3.l. $n + 2^n = O(n)$

3.m. $n + \frac{100n}{\log n} = o(n)$

3.n. $n! = O(n)$

3.e: It's right because the worst case will be Sorts in place.
3.f: It's right because the time will be constant, and there is no other condition.
3.g: It's wrong because 10^6grows much faster, with n than n does and cannot be reduced to O(n).
3. h. It's wrong because n log(n) grows faster.
3.i. It's right because log n is getting smaller and smaller.

3.j. It's right because log n will eventually exceed any constant term as long as n is large enough.
3.k. It's right because (n log n) grows at least as fast as n.
3.l. It's wrong because it will be more relay on 2^n.
3.m. It's wrong because both n have the same increase rate.
3.n. It's wrong because(n!) grows faster than any polynomial.

Give your best- and worst-case asymptotic runtime analysis to the following code snippets.

4.1. Let flag be a random Boolean variable, whose value is either true or false.

```
int count = 0;
for (int i = 0; i < n; i++)
{
   if (flag)
   {
       for (j = i+1; i < n; j++)
       {
           count = count + i + j;
       }
   }
}
```

A: Best-case: O(n) (if flag is false).

Worst-case: O(n^2) (if flag is true).

4.2. Let random() be a function producing a real random number in the range [0,1).

```
int count = 0;
for (int i = 0; i < n; i++)
{
   int num = random();
   if( num < 0.01 )
   {
       count = count + 1;
   }
}
int num = count;
for (int j = 0; j < num; j++)
{
   count = count + j;
}
```

A: Best-case: O(n).

Worst-case: O(n).

4.3. Let Compare(x, y) be a method with the time complexity of $\Theta(1)$.

```
for (int i = 0; i < n; i++)
{
   for (int j = 0; j < n-i-1; j++)
   {
       if (a[j] > a[j+1])
       {
           Compare(a[j], a[j + 1]);
       }
   }
}
```

A: Best-case: O(n^2).

Worst-case: O(n^2).

Similar to bubble sort

9 Advanced questions on algorithmic complexity

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/03/25 13:47	Yi Guan	Ready to Mark
2025/03/25 13:47	Yi Guan	hihi
2025/03/26 17:12	Svitlana Prypoten	Happy to discuss or waiting for video
2025/03/26 17:12	Svitlana Prypoten	Discuss
2025/03/27 11:49	Yi Guan	:grinning:
2025/04/07 20:53	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=7d 197d-486d-8fbb-b2b800aefba3
2025/04/07 20:53	Yi Guan	Miss this is my Video for this task.
2025/04/07 20:54	Yi Guan	:smile:
2025/04/11 20:27	Svitlana Prypoten	I don't have access to the video
2025/04/12 00:42	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=7d 197d-486d-8fbb-b2b800aefba3
2025/04/12 00:43	Yi Guan	Sorry Miss, this time should working.
2025/04/12 00:43	Yi Guan	:smiling_face_with_tear:
2025/04/14 16:14	Svitlana Prypoten	Not all explanation of how you calculate limits are mathematically correct.
2025/04/14 16:14	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Advanced questions on algorithmic complexity

Submitted By:
Yi Guan
ppv
2025/03/25 13:47

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

March 25, 2025

1. f is a function that satisfies the following:
   - f is in $O(n)$,
   - f is in $\Omega(1)$,
   - f is neither in $\theta(1)$ nor in $\theta(n)$.

   Can you give an example of such a function f? Show that the function you name indeed satisfies all of the above.

A:

Function : $f(n)$= sqrt(n)

1. If there are existence constant like x = 1 and n1 = 1 , let n > n1 ,

so we will have sqrt (n) <= n. So f = O(n)

2. If there are existence constant like x = 1 and n1 = 1, let n > n1 ,

So we will have sqrt(n) >= 1 . So f = Omega(1)

3. If f is not in theta(1), let x,y >0 , then let x <= sqrt(n) <= y to satisfies bigger enough value of n . But sqrt(n) will be increase infinity. So it does not make sense.

   If f is not in theta(n), let x , y > 0, then x*n <= sqrt(n) <= y*n. But sqrt(n) / n = 1/sqrt(n) => 0. therefore it won't satisfies x*n <= sqrt(n). So f is not in theta(n).

2. For each pair of functions given below, point out the asymptotic relationships that apply: $f = O(g)$, $f = \theta(g)$, and $f = \Omega(g)$.
   - $f(n) = \sqrt{n}$ and $g(n) = log\ n$
   - $f(n) = 10$ and $g(n) = 11$
   - $f(n) = 100 \cdot 3^n$ and $g(n) = 4^n$
   - $f(n) = 4^{n+2}$ and $g(n) = 2^{2n+3}$
   - $f(n) = 7n \cdot log\ n$ and $g(n) = n \cdot log\ 7n$
   - $f(n) = n!$ and $g(n) = (n+1)!$

A :

1.The asymptotic relationship is f = Omega(g)

Because, based on polynomial growth is faster than logarithmic growth, we can calculate the limit of it. So it will be infinity : sqrt(n) / log n

2. The asymptotic relationship is f = theta(g)

Because 10/11 will be a constant and won't be 0.

3. The asymptotic relationship is f = O(g)

Because lim n-> infty 100*3^n / 4^n = 100* (3/4)^n = 0. so f(n)=O(g(n)).

4. The asymptotic relationship is f = theta(g)

Because 4^(n+2) = 2 ^2(n+2) , and g(n) = 2^(2n+3), therefore 2 ^2(n+2) /2^(2n+3) will be a constant number.

5. The asymptotic relationship is f = theta(g)

Because g(n) = n(log 7 + log n), when n can be grow infinity, therefore f(n) / g(n) < 7. Which is a constant number.

6. The asymptotic relationship is f = O(g)

Because g(n) = (n+1) * n!, so the function will be n! / (n+1) ! = 1/n+1 = 0.

10 Implementation of simple sorting algorithms

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	**Weight**
Complexity	♦♦♦♦♦

bit hard be honest

Outcome	**Weight**
Implement Solutions	♦♦♦♦♦

bit hard be honest

Outcome	**Weight**
Document solutions	♦♦♦♦♦

bit hard be honest

Date	Author	Comment
2025/03/29 18:13	Yi Guan	Ready to Mark
2025/03/29 18:13	Yi Guan	hihi
2025/03/30 16:56	Svitlana Prypoten	Hi Jack, You have some issues in you code:1. Missing validation2. BubbleSort - incorrect contrition for outer loopPlease resubmit once fixed
2025/03/30 16:56	Svitlana Prypoten	Fix and Resubmit
2025/04/05 18:58	Yi Guan	Ready to Mark
2025/04/05 18:58	Svitlana Prypoten	Build failed. Please address the issues raised, fix, and resubmit.
2025/04/05 18:59	Yi Guan	Hi, I just fixed my BubbleSort function and added validation in to my three Sort methods.
2025/04/05 19:00	Yi Guan	this is for the check methods.
2025/04/05 19:01	Yi Guan	image comment
2025/04/05 19:02	Yi Guan	For the BubbleSort Method the number of outer loops is changed to num - 1 times to avoid redundant loops.
2025/04/05 19:03	Yi Guan	image comment
2025/04/05 19:03	Yi Guan	and program run smoothly.
2025/04/05 19:04	Yi Guan	image comment
2025/04/06 20:14	Yi Guan	Ready to Mark
2025/04/06 20:14	Svitlana Prypoten	The program built without error.Run succeeded!
2025/04/06 20:15	Yi Guan	The ISorter interface is defined repeatedly in the Vector namespace. In order to test the code, I simply put the code in the ISorter document into Vector.
2025/04/06 20:15	Yi Guan	image comment
2025/04/06 20:18	Yi Guan	This time should run successfully, I just commented for duplicate code.
2025/04/12 10:23	Svitlana Prypoten	Happy to discuss or waiting for video.The video demonstration for this task should include running1 test for each method in debug mode to sort array partially in descending or ascendingorder and explain what happening on each iteration for each method.
2025/04/12 10:23	Svitlana Prypoten	Discuss
2025/04/14 17:09	Yi Guan	Miss
2025/04/14 17:10	Yi Guan	this is the video link for my task 3.1P
2025/04/14 17:10	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=28 799a-4be4-b8ac-b2bf006c223a
2025/04/17 20:12	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Implementation of simple sorting algorithms

Submitted By:
Yi Guan
ppv
2025/04/06 20:14

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

bit hard be honest

April 6, 2025

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Vector;

namespace Vector
{
    public class BubbleSort : ISorter
    {
        public void Sort<K>(K[] array, int index, int num, IComparer<K> comparer)
        ↪  where K : IComparable<K>
        {
            if (array == null)
                throw new ArgumentNullException(nameof(array));
            if (index < 0)
                throw new ArgumentOutOfRangeException(nameof(index));
            if (num < 0)
                throw new ArgumentOutOfRangeException(nameof(num));
            if (index + num > array.Length)
                throw new ArgumentException("index and num exceed array bounds");


            IComparer<K> comp = comparer ?? Comparer<K>.Default;

            for (int i = 0; i < num -1 ; i++)
            {
                bool swapped = false;
                for (int j = index; j < index + num - i - 1; j++)
                {
                    if (comp.Compare(array[j], array[j + 1]) > 0)
                    {
                        (array[j], array[j + 1]) = (array[j + 1], array[j]);
                        swapped = true;
                    }
                }
                if (!swapped) break;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Vector;

namespace Vector
{
    public class InsertionSort : ISorter
    {
        public void Sort<K>(K[] array, int index, int num, IComparer<K> comparer)
        ↪  where K : IComparable<K>
        {
            if (array == null)
                throw new ArgumentNullException(nameof(array));
            if (index < 0)
                throw new ArgumentOutOfRangeException(nameof(index));
            if (num < 0)
                throw new ArgumentOutOfRangeException(nameof(num));
            if (index + num > array.Length)
                throw new ArgumentException("index and num exceed array bounds");


            IComparer<K> comp = comparer ?? Comparer<K>.Default;

            for (int i = index + 1; i < index + num; i++)
            {
                K current = array[i];
                int j = i - 1;
                while (j >= index && comp.Compare(array[j], current) > 0)
                {
                    array[j + 1] = array[j];
                    j--;
                }
                array[j + 1] = current;
            }
        }
    }
}

```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Vector;

namespace Vector
{
    public class SelectionSort : ISorter
    {
        public void Sort<K>(K[] array, int index, int num, IComparer<K> comparer)
        ↪  where K : IComparable<K>
        {
            IComparer<K> comp = comparer ?? Comparer<K>.Default;

            if (array == null)
                throw new ArgumentNullException(nameof(array));
            if (index < 0)
                throw new ArgumentOutOfRangeException(nameof(index));
            if (num < 0)
                throw new ArgumentOutOfRangeException(nameof(num));
            if (index + num > array.Length)
                throw new ArgumentException("index and num exceed array bounds");

            for (int i = index; i < index + num - 1; i++)
            {
                int minIdx = i;
                for (int j = i + 1; j < index + num; j++)
                {
                    if (comp.Compare(array[j], array[minIdx]) < 0)
                        minIdx = j;
                }
                (array[i], array[minIdx]) = (array[minIdx], array[i]);
            }
        }
    }
}
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vector
{
    //public interface ISorter
    //{
    //    void Sort<K>(K[] array, int index, int num, IComparer<K> comparer) where K : IComparable<K>;
    //}
    public class Vector<T> where T : IComparable<T>
    {
        private const int DEFAULT_CAPACITY = 10;
        private T[] data;

        public int Count { get; private set; } = 0;

        public int Capacity { get; private set; } = 0;

        public Vector(int capacity)
        {
            data = new T[capacity];
            Capacity = capacity;
        }
        public Vector() : this(DEFAULT_CAPACITY) { }
        public T this[int index]
        {
            get
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                return data[index];
            }
            set
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }
        private void ExtendData(int extraCapacity)
        {
            T[] newData = new T[data.Length + extraCapacity];
            for (int i = 0; i < Count; i++)
                newData[i] = data[i];
            data = newData;
            Capacity = data.Length;
        }
        public void Add(T element)
        {
            if (Count == data.Length)
```

```
54                  ExtendData(DEFAULT_CAPACITY);
55              data[Count++] = element;
56          }
57          public int IndexOf(T element)
58          {
59              for (int i = 0; i < Count; i++)
60              {
61                  if (data[i].Equals(element))
62                      return i;
63              }
64              return -1;
65          }
66
67
68          public void Insert(int index, T element)
69          {
70              if (index < 0 || index > Count)
71                  throw new IndexOutOfRangeException();
72              if (Count == data.Length)
73                  ExtendData(DEFAULT_CAPACITY);
74              if (index == Count)
75              {
76                  Add(element);
77                  return;
78              }
79              for (int i = Count; i > index; i--)
80              {
81                  data[i] = data[i - 1];
82              }
83              data[index] = element;
84              Count++;
85          }
86
87
88          public void Clear()
89          {
90              Array.Clear(data, 0, Count);
91              Count = 0;
92          }
93
94
95          public bool Contains(T element)
96          {
97              return IndexOf(element) != -1;
98          }
99
100
101         public bool Remove(T element)
102         {
103             int index = IndexOf(element);
104             if (index != -1)
105             {
106                 RemoveAt(index);
107                 return true;
```

```
108             }
109             return false;
110         }
111 
112 
113         public void RemoveAt(int index)
114         {
115             if (index < 0 || index >= Count)
116                 throw new IndexOutOfRangeException();
117             for (int i = index; i < Count - 1; i++)
118             {
119                 data[i] = data[i + 1];
120             }
121             Count--;
122             data[Count] = default(T);
123         }
124 
125 
126         public override string ToString()
127         {
128             StringBuilder sb = new StringBuilder();
129             sb.Append("[");
130             for (int i = 0; i < Count; i++)
131             {
132                 sb.Append(data[i]?.ToString());
133                 if (i < Count - 1)
134                     sb.Append(", ");
135             }
136             sb.Append("]");
137             return sb.ToString();
138         }
139         //public void Sort()
140         //{
141         //    Array.Sort(data, 0, Count);
142         //}
143         //public void Sort(IComparer<T> comparer)
144         //{
145         //    // if (comparer == null)
146         //    //  throw new argumentnullexception(nameof(comparer));
147         //    Array.Sort(data, 0, Count, comparer);
148         //}
149         public void Sort(ISorter algorithm, IComparer<T> comparer)
150         {
151             if (algorithm == null)
152             {
153                 Array.Sort(data, 0, Count, comparer ?? Comparer<T>.Default);
154             }
155             else
156             {
157                 algorithm.Sort<T>(data, 0, Count, comparer ?? Comparer<T>.Default);
158             }
159         }
160         public class Student : IComparable<Student>
161         {
```

```
162             public string Name { get; set; }
163             public int Id { get; set; }
164 
165             public override string ToString()
166             {
167                 return $"{Id}[{Name}]";
168             }
169 
170             public int CompareTo(Student other)
171             {
172                 if (other == null) return 1;
173                 return Id.CompareTo(other.Id);
174             }
175         }
176 
177     }
178 }
179 
180 
```

11 Implementation of the Binary Search algorithm

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/04/05 19:23	Yi Guan	Ready to Mark
2025/04/05 19:23	Yi Guan	hihi
2025/04/06 00:16	Svitlana Prypoten	Happy to discuss or waiting for video.The video demonstration for this task should include demonstrationon example how you code works, logic and result for each iteration. You can usedebugging mode. As well explain your choice of iterative or recursive versions of the Binary Search Algorithm.
2025/04/06 00:16	Svitlana Prypoten	Discuss
2025/05/01 19:46	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=1a 4aac-439e-bbc2-b2cf012c6b01
2025/05/01 19:46	Yi Guan	Hi, this is my video link for this task
2025/05/03 09:00	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Implementation of the Binary Search algorithm

Submitted By:
Yi Guan
ppv
2025/04/05 19:23

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

April 5, 2025

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SIT221_4._1P
{
    public interface ISorter
    {
        void Sort<K>(K[] array, int index, int num, IComparer<K> comparer) where K
        ↪  : IComparable<K>;
    }
    public class Vector<T> where T : IComparable<T>
    {
        private const int DEFAULT_CAPACITY = 10;
        private T[] data;

        public int Count { get; private set; } = 0;

        public int Capacity { get; private set; } = 0;

        public Vector(int capacity)
        {
            data = new T[capacity];
            Capacity = capacity;
        }
        public Vector() : this(DEFAULT_CAPACITY) { }
        public T this[int index]
        {
            get
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                return data[index];
            }
            set
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }
        private void ExtendData(int extraCapacity)
        {
            T[] newData = new T[data.Length + extraCapacity];
            for (int i = 0; i < Count; i++)
                newData[i] = data[i];
            data = newData;
            Capacity = data.Length;
        }
        public void Add(T element)
        {
            if (Count == data.Length)
```

```
54                  ExtendData(DEFAULT_CAPACITY);
55              data[Count++] = element;
56          }
57          public int IndexOf(T element)
58          {
59              for (int i = 0; i < Count; i++)
60              {
61                  if (data[i].Equals(element))
62                      return i;
63              }
64              return -1;
65          }
66
67
68          public void Insert(int index, T element)
69          {
70              if (index < 0 || index > Count)
71                  throw new IndexOutOfRangeException();
72              if (Count == data.Length)
73                  ExtendData(DEFAULT_CAPACITY);
74              if (index == Count)
75              {
76                  Add(element);
77                  return;
78              }
79              for (int i = Count; i > index; i--)
80              {
81                  data[i] = data[i - 1];
82              }
83              data[index] = element;
84              Count++;
85          }
86
87
88          public void Clear()
89          {
90              Array.Clear(data, 0, Count);
91              Count = 0;
92          }
93
94
95          public bool Contains(T element)
96          {
97              return IndexOf(element) != -1;
98          }
99
100
101         public bool Remove(T element)
102         {
103             int index = IndexOf(element);
104             if (index != -1)
105             {
106                 RemoveAt(index);
107                 return true;
```

```
108             }
109             return false;
110         }
111 
112 
113         public void RemoveAt(int index)
114         {
115             if (index < 0 || index >= Count)
116                 throw new IndexOutOfRangeException();
117             for (int i = index; i < Count - 1; i++)
118             {
119                 data[i] = data[i + 1];
120             }
121             Count--;
122             data[Count] = default(T);
123         }
124 
125 
126         public override string ToString()
127         {
128             StringBuilder sb = new StringBuilder();
129             sb.Append("[");
130             for (int i = 0; i < Count; i++)
131             {
132                 sb.Append(data[i]?.ToString());
133                 if (i < Count - 1)
134                     sb.Append(", ");
135             }
136             sb.Append("]");
137             return sb.ToString();
138         }
139         //public void Sort()
140         //{
141         //    Array.Sort(data, 0, Count);
142         //}
143         //public void Sort(IComparer<T> comparer)
144         //{
145         //    // if (comparer == null)
146         //    //  throw new argumentnullexception(nameof(comparer));
147         //    Array.Sort(data, 0, Count, comparer);
148         //}
149         public void Sort(ISorter algorithm, IComparer<T> comparer)
150         {
151             if (algorithm == null)
152             {
153                 Array.Sort(data, 0, Count, comparer ?? Comparer<T>.Default);
154             }
155             else
156             {
157                 algorithm.Sort<T>(data, 0, Count, comparer ?? Comparer<T>.Default);
158             }
159         }
160         public class Student : IComparable<Student>
161         {
```

```
162             public string Name { get; set; }
163             public int Id { get; set; }
164 
165             public override string ToString()
166             {
167                 return $"{Id}[{Name}]";
168             }
169 
170             public int CompareTo(Student other)
171             {
172                 if (other == null) return 1;
173                 return Id.CompareTo(other.Id);
174             }
175         }
176         public int BinarySearch(T item, IComparer<T> comparer)
177         {
178             IComparer<T> comp = comparer ?? Comparer<T>.Default;
179             int left = 0;
180             int right = Count - 1;
181 
182             while (left <= right)
183             {
184                 int mid = left + (right - left) / 2;
185                 int comparison = comp.Compare(data[mid], item);
186 
187                 if (comparison == 0)
188                     return mid;
189                 else if (comparison < 0)
190                     left = mid + 1;
191                 else
192                     right = mid - 1;
193             }
194             return -1;
195         }
196     }
197 
198 }
199 
200 
```

12 Implementation of advanced sorting algorithms

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

So hard :(

Outcome	Weight
Implement Solutions	♦♦♦♦♦

So hard :(

Outcome	Weight
Document solutions	♦♦♦♦♦

So hard :(

Date	Author	Comment
2025/04/09 20:48	Yi Guan	Ready to Mark
2025/04/09 20:48	Svitlana Prypoten	The program built without error.Run succeeded!
2025/04/09 20:48	Yi Guan	hihi
2025/04/12 10:25	Svitlana Prypoten	Happy to discuss or waiting for video.The video demonstration for this task should include running1 test for each method in debug mode to sort array partially in descending or ascendingorder and explain what happening on each iteration for each method.
2025/04/12 10:25	Svitlana Prypoten	Discuss
2025/05/01 19:46	Yi Guan	hi, this is my video link: https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=93 b646-4086-aa60-b2d000951bb0
2025/05/03 09:13	Svitlana Prypoten	What is the time complexity of Randomized Quick-Sort?
2025/05/03 23:47	Yi Guan	for the Quick Sort the time complexity is O(n log n) and it's for average time taken.
2025/05/03 23:47	Yi Guan	worse case will be O(n^2)
2025/05/03 23:49	Yi Guan	But Merge sort and Heap sort, they have the same time complexity as O(n log n) both good case and worse case.
2025/05/18 09:39	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Implementation of advanced sorting algorithms

Submitted By:
Yi Guan
ppv
2025/04/09 20:48

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

So hard :(

April 9, 2025

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vector
{
    public class RandomizedQuickSort : ISorter
    {
        private Random random = new Random();

        public void Sort<K>(K[] array, int index, int num, IComparer<K> comparer)
        ↪  where K : IComparable<K>
        {
            //Limit sort range.

            QuickSort(array, index, index + num - 1, comparer);
        }

        private void QuickSort<K>(K[] array, int left, int right, IComparer<K>
        ↪  comparer) where K : IComparable<K>
        {
            //Recursion termination condition.

            if (left < right)
            {
                int pivotIndex = Partition(array, left, right, comparer);
                //Recursively process the left half.

                QuickSort(array, left, pivotIndex - 1, comparer);

                //Recursively process the right half.

                QuickSort(array, pivotIndex + 1, right, comparer);
            }
        }

        private int Partition<K>(K[] array, int left, int right, IComparer<K>
        ↪  comparer) where K : IComparable<K>
        {
            int randomPivot = random.Next(left, right + 1);

            //Swap the pivot to the end of the subarray.
            Swap(array, randomPivot, right);

            K pivot = array[right]; // Current value.

            int i = left - 1; // Record value of right.

            for (int j = left; j < right; j++)
            {
                //If it's current value is equal or smaller, so swap to the left
                ↪  side.
```

```
51
52                  if (comparer.Compare(array[j], pivot) <= 0)
53                  {
54                      i++;
55                      Swap(array, i, j);
56                  }
57              }
58              Swap(array, i + 1, right);
59
60              return i + 1;
61          }
62
63          private void Swap<K>(K[] array, int i, int j)
64          {
65              K temp = array[i];
66              array[i] = array[j];
67              array[j] = temp;
68          }
69      }
70  }
71
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vector
{
    public class MergeSortTopDown : ISorter
    {
        public void Sort<K>(K[] array, int index, int num, IComparer<K> comparer)
        ↪  where K : IComparable<K>
        {
            //Limit sort range.

            MergeSort(array, index, index + num - 1, comparer);
        }

        private void MergeSort<K>(K[] array, int left, int right, IComparer<K>
        ↪  comparer) where K : IComparable<K>
        {
            //Recursion termination condition.
            if (left < right)
            {
                int mid = (left + right) / 2; // Mid point.
                MergeSort(array, left, mid, comparer); //Sorting left side.
                MergeSort(array, mid + 1, right, comparer);//Sorting right side.
                Merge(array, left, mid, right, comparer);// All together.
            }
        }

        private void Merge<K>(K[] array, int left, int mid, int right, IComparer<K>
        ↪  comparer)
        {
            // Create a temporary array to store data.
            K[] temp = new K[right - left + 1];
            int i = left, j = mid + 1, k = 0;

            while (i <= mid && j <= right)
            {
                // Compare of the two elements , put samll one int to temporary
                ↪  array.
                if (comparer.Compare(array[i], array[j]) <= 0)
                    temp[k++] = array[i++];
                else
                    temp[k++] = array[j++];
            }

            // Get the rest of the value in left.
            while (i <= mid) temp[k++] = array[i++];

            // Get the rest of the value in right.
            while (j <= right) temp[k++] = array[j++];

```

```
51              //Copy the temporary array in to orignal array.
52              Array.Copy(temp, 0, array, left, temp.Length);
53          }
54      }
55  }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vector
{
    public class MergeSortBottomUp : ISorter
    {

        public void Sort<K>(K[] array, int index, int num, IComparer<K> comparer)
        ↪  where K : IComparable<K>
        {
            int n = index + num; //Get the position of the last element to be
            ↪  sorted.
            for (int size = 1; size < n; size *= 2)// Outer loop
            {
                for (int left = index; left < n; left += 2 * size)
                {
                    //Determine the middle and right limits of the current two
                    ↪  subsequences respectively.
                    int mid = Math.Min(left + size - 1, n - 1);
                    int right = Math.Min(left + 2 * size - 1, n - 1);
                    //Merge two adjacent subarrays.
                    Merge(array, left, mid, right, comparer);
                }
            }
        }

        private void Merge<K>(K[] array, int left, int mid, int right, IComparer<K>
        ↪  comparer)
        {
            // Create a temporary array to store data.
            K[] temp = new K[right - left + 1];
            int i = left, j = mid + 1, k = 0;

            while (i <= mid && j <= right)
            {
                // Compare of the two elements , put samll one int to temporary
                ↪  array.
                if (comparer.Compare(array[i], array[j]) <= 0)
                    temp[k++] = array[i++];
                else
                    temp[k++] = array[j++];
            }

            // Get the rest of the value in left.
            while (i <= mid) temp[k++] = array[i++];

            // Get the rest of the value in right.
            while (j <= right) temp[k++] = array[j++];

            //Copy the temporary array in to orignal array.
```

```
50              Array.Copy(temp, 0, array, left, temp.Length);
51          }
52      }
53  }
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace Vector
{
    //public interface isorter
    //{
    //    void sort<k>(k[] array, int index, int num, icomparer<k> comparer) where k : icomparable<k>;
    //}
    public class Vector<T> where T : IComparable<T>
    {
        private const int DEFAULT_CAPACITY = 10;
        private T[] data;

        public int Count { get; private set; } = 0;

        public int Capacity { get; private set; } = 0;

        public Vector(int capacity)
        {
            data = new T[capacity];
            Capacity = capacity;
        }
        public Vector() : this(DEFAULT_CAPACITY) { }
        public T this[int index]
        {
            get
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                return data[index];
            }
            set
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }
        private void ExtendData(int extraCapacity)
        {
            T[] newData = new T[data.Length + extraCapacity];
            for (int i = 0; i < Count; i++)
                newData[i] = data[i];
            data = newData;
            Capacity = data.Length;
        }
        public void Add(T element)
        {
```

```
54              if (Count == data.Length)
55                  ExtendData(DEFAULT_CAPACITY);
56              data[Count++] = element;
57          }
58          public int IndexOf(T element)
59          {
60              for (int i = 0; i < Count; i++)
61              {
62                  if (data[i].Equals(element))
63                      return i;
64              }
65              return -1;
66          }
67
68
69          public void Insert(int index, T element)
70          {
71              if (index < 0 || index > Count)
72                  throw new IndexOutOfRangeException();
73              if (Count == data.Length)
74                  ExtendData(DEFAULT_CAPACITY);
75              if (index == Count)
76              {
77                  Add(element);
78                  return;
79              }
80              for (int i = Count; i > index; i--)
81              {
82                  data[i] = data[i - 1];
83              }
84              data[index] = element;
85              Count++;
86          }
87
88
89          public void Clear()
90          {
91              Array.Clear(data, 0, Count);
92              Count = 0;
93          }
94
95
96          public bool Contains(T element)
97          {
98              return IndexOf(element) != -1;
99          }
100
101
102         public bool Remove(T element)
103         {
104             int index = IndexOf(element);
105             if (index != -1)
106             {
107                 RemoveAt(index);
```

```
108                 return true;
109             }
110             return false;
111         }
112 
113 
114         public void RemoveAt(int index)
115         {
116             if (index < 0 || index >= Count)
117                 throw new IndexOutOfRangeException();
118             for (int i = index; i < Count - 1; i++)
119             {
120                 data[i] = data[i + 1];
121             }
122             Count--;
123             data[Count] = default(T);
124         }
125 
126 
127         public override string ToString()
128         {
129             StringBuilder sb = new StringBuilder();
130             sb.Append("[");
131             for (int i = 0; i < Count; i++)
132             {
133                 sb.Append(data[i]?.ToString());
134                 if (i < Count - 1)
135                     sb.Append(", ");
136             }
137             sb.Append("]");
138             return sb.ToString();
139         }
140         //public void Sort()
141         //{
142         //    Array.Sort(data, 0, Count);
143         //}
144         //public void Sort(IComparer<T> comparer)
145         //{
146         //    // if (comparer == null)
147         //    //  throw new argumentnullexception(nameof(comparer));
148         //    Array.Sort(data, 0, Count, comparer);
149         //}
150         public void Sort(ISorter algorithm, IComparer<T> comparer)
151         {
152             if (algorithm == null)
153             {
154                 Array.Sort(data, 0, Count, comparer ?? Comparer<T>.Default);
155             }
156             else
157             {
158                 algorithm.Sort<T>(data, 0, Count, comparer ?? Comparer<T>.Default);
159             }
160         }
161         public class Student : IComparable<Student>
```

```
        {
            public string Name { get; set; }
            public int Id { get; set; }

            public override string ToString()
            {
                return $"{Id}[{Name}]";
            }

            public int CompareTo(Student other)
            {
                if (other == null) return 1;
                return Id.CompareTo(other.Id);
            }
        }

    }
}
```

13 Exploring the Skip-List data structure

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/04/09 02:43	Yi Guan	Ready to Mark
2025/04/09 02:43	Yi Guan	hihi
2025/04/14 16:56	Svitlana Prypoten	Your skip-list is incorrect, please double check andresubmit once fixed.
2025/04/14 16:56	Svitlana Prypoten	Fix and Resubmit
2025/04/14 17:49	Yi Guan	Hi, Miss. I thought the element 0 also in layer 3, I just deleted that 0 from layer 3. the skip-list should be all correct now
2025/04/14 17:49	Yi Guan	Ready to Mark
2025/04/17 19:00	Svitlana Prypoten	In step 1 you said 1 is in level 1, why you put it in all 3 levels in this case? and why you miss 4o from level 2?
2025/04/17 19:00	Svitlana Prypoten	Fix and Resubmit
2025/04/24 00:08	Yi Guan	Ready to Mark
2025/04/24 00:11	Yi Guan	Miss, When initially inserted, the level of element 1 should indeed be level 1. In the subsequent insertion steps, when inserting 85 (level 3) and 8 (level 3), I mistakenly promoted 1 to a higher level. In the actual skip list rule, the level of an element is determined by the random level when it is inserted, and subsequent insertions will not modify the level of an existing element.
2025/04/24 00:13	Yi Guan	I also corrected the problem about missing 40 in layer 2. I thought it went directly to 85, so 40 was not needed. It was my mistake.
2025/04/24 00:13	Yi Guan	The final result should be like this:Layer 3: 1 -> 8 -> 85Layer 2: 0 -> 1 -> 5 -> 8 -> 40 -> 85Layer 1: 0 -> 1 -> 5 -> 8 -> 11 -> 40 -> 85 -> 86
2025/04/28 17:37	Svitlana Prypoten	You are writing that element 1 should be on level 1, but still have it on level 2 and 3
2025/04/28 17:37	Svitlana Prypoten	Fix and Resubmit
2025/04/28 20:52	Yi Guan	Ready to Mark
2025/04/28 20:52	Yi Guan	Got it Miss, I have fixed it.
2025/05/03 23:54	Yi Guan	Miss, for this task, can I still submit the video for this assignment or discuss it this Monday? Because it is waiting for feedback, I did not submit the assignment video without permission. Although I know it has been revised many times before, if the above content cannot be discussed, can I discuss it at the end of the semester? I remember there were 3 opportunities.
2025/05/03 23:55	Yi Guan	:scream:
2025/05/05 19:01	Svitlana Prypoten	Happy to discuss or waiting for video.The video demonstration for this task should include explanation for theory questions,detail explanation how you create your skip-list and example how you will dosearch for some value in your skip list.
2025/05/05 19:07	Svitlana Prypoten	Discuss
2025/05/07 18:26	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=d1 94e8-4960-9192-b2d60088e23a
2025/05/07 18:26	Yi Guan	Hi Miss, this is the video link for my task 4.2P
2025/05/07 18:26	Yi Guan	:grinning:
2025/05/09 18:20	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Exploring the Skip-List data structure

Submitted By:
Yi Guan
ppv
2025/04/28 20:52

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

April 28, 2025

1. - How exactly is the skip-list constructed from layers of linked-lists? What does each of the linked-lists constituting the skip-list store?

A： Skip-List consists of multiple layers of ordered linked lists. The bottom layer of linked lists contains all elements, and each upper layer of linked lists is a subset of the lower layer, with the element spacing increasing gradually. Each layer of linked lists stores references to some elements, and the elements in the higher layer of linked lists serve as "jumping points" to speed up the search.

2. What is the height of a skip-list? How can we determine the height when adding new elements?

A：

The height is the maximum number of layers in the Skip-List.

When inserting an element, the number of layers is determined by "flipping a coin". For example, if the coin flip sequence is HHT, the number of layers is 3.

3. What is the runtime complexity for searching, insertion, and removal operations on the skip-list? What does it mean when we say that the complexity of these operations is '***expected***'? What is the difference between an ***expected* bound** on runtime complexity and a ***worst-case* bound**?

A：

The expected time complexity of searching, inserting, and deleting is O(log n).

"Expected" refers to the performance in the average case, which depends on randomness; the worst case may be O(n), that is, all elements are concentrated in the same layer.

The expected bound is the average probability, and the worst case bound considers all possible inputs.

4. Compare the skip-list to other data-structures that you already know, e.g., a list collection, singly and doubly linked lists. What are the advantages and disadvantages of the skip-list? Compare the complexities of the basic operations offered by the skip-list to those of the mentioned data structures.

A：

Data structure	Advantages	Disadvantages	Complexity
Skip-List	O(log n) expected time/ Simple to use	Additional space overhead	O(log n)
Singly linked list	Simple, low memory	Linear time operation	O(n)

Balanced tree	O(log n) worst-case time	Hard to achieve	O(log n)

5.

How exactly does each of the standard operations work? Here, you should be ready to explain the sequence of actions required to perform searching, insertion, and removal from the skip-list.

A :

Search: Start from the highest level, go right to find the largest node that is smaller than the target, sink to the next level, and continue until the bottom level.

Insert: Determine the number of new node levels, insert each level, and update the pointer.

Delete: Find the position of the node in each level, delete each level, and repair the pointer.

2. Solve the following numeric example. Given the sequence of ***coin tosses***, where H stands for ***head*** and T stands for ***tails***:

T H T T H H T T H T H T H H T H H T T H H T H T T H T T T H T H T H T T H T H H T T.

Build a skip-list containing the following values:

1 40 11 85 86 5 0 8.

Show the full state of the skip-list after each insertion. Note that *you must explicitly state the meaning you attach* to *heads* and *tails* at the start of your answer. Remember that you must start tossing the coin from left side of the sequence and may not need to use all the tosses to build the skip-list. You may assume as the height resolution policy that we allow a tower to grow as long as heads (or tails) keep getting returned from the given sequence, which emulates a random number generator.

A : Based on the insertion order: 1 -> 40 -> 11 -> 85 -> 86 -> 5 -> 0 -> 8

Element	Coin	Number of Layers
1	T	1
40	H -> T	2
11	T	1
85	H-> H -> T	3
86	T	1
5	H -> T	2
0	H -> T	2
8	H-> H-> T	3

Insertion process:

1. Insert 1 (Layer 1)

Layer 1: 1

2. Insert 40 (Layer 2)
   Layer 2: 40
   Layer 1: 1 -> 40

3. Insert 11 (Layer 1)
   Layer 2: 40
   Layer 1: 1 -> 11 -> 40

4. Insert 85 (Layer 3)
   Layer 3: 85
   Layer 2: 40 -> 85
   Layer 1: 1 -> 11 -> 40 -> 85

5. Insert 86 (Layer 1)
   Layer 3: 85
   Layer 2: 40 -> 85
   Layer 1: 1 -> 11 -> 40 -> 85 -> 86

6. Insert 5 (Layer 2)
   Layer 3: 85
   Layer 2: 5 -> 40 -> 85
   Layer 1: 1 -> 5 -> 11 -> 40 -> 85 -> 86

7. Insert 0 (Layer 2)
   Layer 3: 85
   Layer 2: 0 -> 5 -> 40 -> 85
   Layer 1: 0 -> 1 -> 5 -> 11 -> 40 -> 85 -> 86

8. Insert 8 (Layer 3)
   Layer 3: 8 -> 85
   Layer 2: 0 -> 5 -> 8 -> 40 -> 85
   Layer 1: 0 -> 1 -> 5 -> 8 -> 11 -> 40 -> 85 -> 86

Final Skip-List status:

Layer 3:8 -> 85

Layer 2: 0 ->5 -> 8 -> 40 -> 85

Layer 1: 0 -> 1 -> 5 -> 8 -> 11 -> 40 -> 85 -> 86

14 Implementation of the Doubly Linked List

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/04/14 12:49	Yi Guan	Ready to Mark
2025/04/14 12:49	Svitlana Prypoten	The program built without error.Run succeeded!
2025/04/14 12:49	Yi Guan	hihi
2025/04/17 20:45	Svitlana Prypoten	Happyto discuss or waiting for video.
2025/04/17 20:45	Svitlana Prypoten	Discuss
2025/05/05 19:14	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=b1 b902-4cbd-accd-b2d4009091e8
2025/05/05 19:14	Yi Guan	Hi, Miss this is my video link for this task
2025/05/05 20:36	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Implementation of the Doubly Linked List

Submitted By:
Yi Guan
ppv
2025/04/14 12:49

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

April 14, 2025

```
using System;
using System.Text;

namespace DoublyLinkedList
{
    public class DoublyLinkedList<T>
    {

        // Here is the the nested Node<K> class
        private class Node<K> : INode<K>
        {
            public K Value { get; set; }
            public Node<K> Next { get; set; }
            public Node<K> Previous { get; set; }

            public Node(K value, Node<K> previous, Node<K> next)
            {
                Value = value;
                Previous = previous;
                Next = next;
            }

            // This is a ToString() method for the Node<K>
            // It represents a node as a tuple {'the previous node's value'-(the node's value)-'the next node's value')}.
            // 'XXX' is used when the current node matches the First or the Last of the DoublyLinkedList<T>
            public override string ToString()
            {
                StringBuilder s = new StringBuilder();
                s.Append("{");
                s.Append(Previous.Previous == null ? "XXX" : Previous.Value.ToString());
                s.Append("-(");
                s.Append(Value);
                s.Append(")-");
                s.Append(Next.Next == null ? "XXX" : Next.Value.ToString());
                s.Append("}");
                return s.ToString();
            }

        }

        // Here is where the description of the methods and attributes of the DoublyLinkedList<T> class starts

        // An important aspect of the DoublyLinkedList<T> is the use of two auxiliary nodes: the Head and the Tail.
```

```
44          // The both are introduced in order to significantly simplify the
            ↪  implementation of the class and make insertion functionality reduced
            ↪  just to a AddBetw
45  een(...)
46          // These properties are private, thus are invisible to a user of the data
            ↪  structure, but are always maintained in it, even when the
            ↪  DoublyLinkedList<T>
47  is formally empty.
48          // Remember about this crucial fact when you design and code other functions
            ↪  of the DoublyLinkedList<T> in this task.
49          private Node<T> Head { get; set; }
50          private Node<T> Tail { get; set; }
51          public int Count { get; private set; } = 0;
52
53          public DoublyLinkedList()
54          {
55              Head = new Node<T>(default(T), null, null);
56              Tail = new Node<T>(default(T), Head, null);
57              Head.Next = Tail;
58          }
59
60          public INode<T> First
61          {
62              get
63              {
64                  if (Count == 0) return null;
65                  else return Head.Next;
66              }
67          }
68
69          public INode<T> Last
70          {
71              get
72              {
73                  if (Count == 0) return null;
74                  else return Tail.Previous;
75              }
76          }
77
78          public INode<T> After(INode<T> node)
79          {
80              if (node == null) throw new NullReferenceException();
81              Node<T> node_current = node as Node<T>;
82              if (node_current.Previous == null || node_current.Next == null) throw
                ↪  new InvalidOperationException("The node referred as 'before' is no
                ↪  longer in t
83  he list");
84              if (node_current.Next.Equals(Tail)) return null;
85              else return node_current.Next;
86          }
87
88          public INode<T> AddLast(T value)
89          {
90              return AddBetween(value, Tail.Previous, Tail);
```

```
91          }
92
93          // This is a private method that creates a new node and inserts it in
            ↪  between the two given nodes referred as the previous and the next.
94          // Use it when you wish to insert a new value (node) into the
            ↪  DoublyLinkedList<T>
95          private Node<T> AddBetween(T value, Node<T> previous, Node<T> next)
96          {
97              Node<T> node = new Node<T>(value, previous, next);
98              previous.Next = node;
99              next.Previous = node;
100             Count++;
101             return node;
102         }
103
104         public INode<T> Find(T value)
105         {
106             Node<T> node = Head.Next;
107             while (!node.Equals(Tail))
108             {
109                 if (node.Value.Equals(value)) return node;
110                 node = node.Next;
111             }
112             return null;
113         }
114
115         public override string ToString()
116         {
117             if (Count == 0) return "[]";
118             StringBuilder s = new StringBuilder();
119             s.Append("[");
120             int k = 0;
121             Node<T> node = Head.Next;
122             while (!node.Equals(Tail))
123             {
124                 s.Append(node.ToString());
125                 node = node.Next;
126                 if (k < Count - 1) s.Append(",");
127                 k++;
128             }
129             s.Append("]");
130             return s.ToString();
131         }
132
133         // TODO: Your task is to implement all the remaining methods.
134         // Read the instruction carefully, study the code examples from above as
            ↪  they should help you to write the rest of the code.
135
136         public INode<T> Before(INode<T> node)
137         {
138             if (node == null)
139                 throw new ArgumentNullException(nameof(node));
140             Node<T> current = node as Node<T>;
141             if (current == null || current.Previous == null || current.Next == null)
```

```
142                 throw new InvalidOperationException("The node is not in the current
                    ↪  list");
143             if (current.Previous == Head)
144                 return null;
145             return current.Previous;
146         }
147 
148         public INode<T> AddFirst(T value)
149         {
150             return AddBetween(value, Head, Head.Next);
151         }
152 
153         public INode<T> AddBefore(INode<T> before, T value)
154         {
155             if (before == null)
156                 throw new ArgumentNullException(nameof(before));
157             Node<T> nodeBefore = before as Node<T>;
158             if (nodeBefore == null || nodeBefore.Previous == null ||
                ↪  nodeBefore.Next == null)
159                 throw new InvalidOperationException("Node is not in the list");
160             return AddBetween(value, nodeBefore.Previous, nodeBefore);
161         }
162 
163         public INode<T> AddAfter(INode<T> after, T value)
164         {
165             if (after == null)
166                 throw new ArgumentNullException(nameof(after));
167             Node<T> nodeAfter = after as Node<T>;
168             if (nodeAfter == null || nodeAfter.Previous == null || nodeAfter.Next
                ↪  == null)
169                 throw new InvalidOperationException("Node is not in the list");
170             return AddBetween(value, nodeAfter, nodeAfter.Next);
171         }
172 
173         public void Clear()
174         {
175             Node<T> current = Head.Next;
176             while (current != Tail)
177             {
178                 Node<T> nextNode = current.Next;
179                 current.Previous = null;
180                 current.Next = null;
181                 current = nextNode;
182             }
183             Head.Next = Tail;
184             Tail.Previous = Head;
185             Count = 0;
186         }
187 
188         public void Remove(INode<T> node)
189         {
190             if (node == null)
191                 throw new ArgumentNullException(nameof(node));
192             Node<T> nodeToRemove = node as Node<T>;
```

```
193             if (nodeToRemove == null || nodeToRemove.Previous == null ||
                ↪  nodeToRemove.Next == null)
194                 throw new InvalidOperationException("Node is not in the list");
195 
196             nodeToRemove.Previous.Next = nodeToRemove.Next;
197             nodeToRemove.Next.Previous = nodeToRemove.Previous;
198             nodeToRemove.Previous = null;
199             nodeToRemove.Next = null;
200             Count--;
201         }
202 
203         public void RemoveFirst()
204         {
205             if (Count == 0)
206                 throw new InvalidOperationException("The list is empty");
207             Remove(Head.Next);
208         }
209 
210         public void RemoveLast()
211         {
212             if (Count == 0)
213                 throw new InvalidOperationException("The list is empty");
214             Remove(Tail.Previous);
215         }
216 
217     }
218 }
```

15 Stacks, heaps, and Elon Musk

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

alg

Outcome	Weight
Implement Solutions	♦♦♦♦♦

alg

Outcome	Weight
Document solutions	♦♦♦♦♦

alg

Date	Author	Comment
2025/04/28 20:51	Yi Guan	Ready to Mark
2025/04/28 20:51	Yi Guan	hihi
2025/05/07 21:47	Svitlana Prypoten	The video should include explanation of each part of the task and answers onfollowing questions:1. What is binary heap?2. What are properties for max binary heap?3. What are DownHeap and UpHeap procedures?
2025/05/07 21:47	Svitlana Prypoten	Discuss
2025/05/15 22:28	Yi Guan	Hi Miss, this is the video for this task.
2025/05/15 22:28	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=93d4d7-4206-8f19-b2de00c7b97d
2025/05/15 22:28	Yi Guan	:grinning:
2025/05/18 09:18	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Stacks, heaps, and Elon Musk

Submitted By:
Yi Guan
ppv
2025/04/28 20:51

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

April 28, 2025

- **Part A**

The Stack data structure provides **Push** and **Pop**, the two operations to write and read data, respectively. Your task is to extend this data structure so that, in addition to these two operations, it also provides the **Min** operation to return the smallest element in the stack. Remember that the new data structure must operate in a constant $O(1)$ time for all three aforementioned operations.

A： The main stack stores all elements, and the auxiliary stack stores the current minimum value of each step. When pushing an element, if the new element is ≤ the top of the auxiliary stack, it is also pushed into the auxiliary stack; when popping an element, if the element popped is equal to the top of the auxiliary stack, the auxiliary stack is popped out simultaneously. In this way, when taking the minimum value, you only need to return to the top of the auxiliary stack, and the operations of pushing, popping, and taking the minimum value can all be completed in O(1) time.

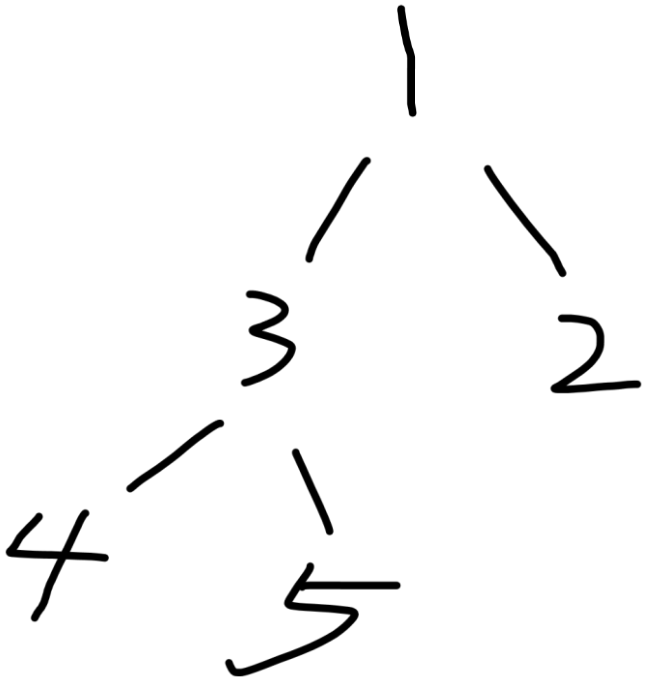
- **Part B**

After reading about the Binary Max-Heap data structure, Elon Musk decided to implement it as part of his supper-duper algorithm. However, as he was not paying enough attention in the class, his implementation of the **DownHeap** procedure (also known as the **Max-Heapify**) is not correct. He gave us the following pseudocode of his implementation.

```
Function DownHeap( index ):
    If ( H[Left( index )].key > H[index].key )
    {
        Swap elements H[index] and H[Left( index )]
        DownHeap ( Left( index ) )
    }
    Else If ( H[Right( index )].key > H[index].key )
    {
        Swap elements H[index] and H[Right( index )]
        DownHeap ( Right( index ) )
    }
```

Your task is to give a numeric example of a heap H, for which the call of the above **DownHeap** procedure leads to a wrong structure of H as a max-heap. Specifically, you need to propose a heap where the root H[1] is the only node breaking the essential **Max-Heap Property** so that running the above **DownHeap(1)** (i.e., index = 1) does NOT result in a valid max-heap. You must draw the heap H before and after running **DownHeap(1)**. Note that your instance can be small.

A： Let heap where the root (index 1) violates the max-heap property (before downheap):

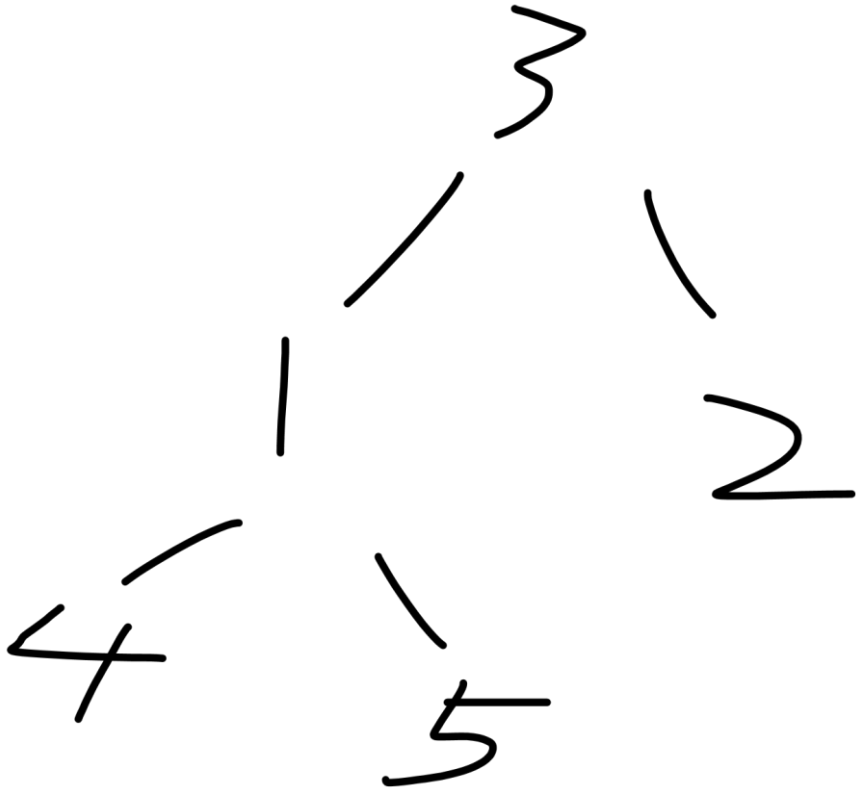


The array will be : [1,3,2,4,5]

Compare left child (3) with root (1). Since 3>1, swap them.

Recursively call downheap on the left child (now index 2).

The new structure becomes:



But the Elon’s code does not check this subtree again.

If with the downheap(1):

The subtree rooted at index 2 still violates the max-heap property. The correct DownHeap should recursively compare and swap with the largest child, but Elon’s code terminates prematurely.

- **Part C**

Assume now that Elon Musk has eventually fixed the problem above and developed a correct implementation of the Max-Heap that provides all the standard operations such as:

- **DeleteMax**, **Insert(key)**, **IncreaseKey(index, key)** in a logarithmic $O(\log n)$ time, and
- **Max** in a constant $O(1)$ time,

where n is the total number of stored elements. Elon Musk wants to add a new capability to his max-heap. Specifically, he would like to implement a **DeleteElement(index)** operation that removes element H[index] from the max-heap H consisting of n elements in a logarithmic $O(\log n)$ time. Note that the H[index] does

NOT need to be the maximum-key element in H. Also, naturally, H must maintain the Max-Heap Property after deletion of H[index].

Your task is to help Elon Musk by providing a pseudocode of such a **DeleteElement(index)** operation accompanied with a brief analysis of its running time as a function of n elements.

A: Firstly, we can have a out of bounds check method, make sure the target index is in the valid range. For example : 1 < index < H_size.

Then exchange the target element with the last element of the heap. Delete the last element. For example: H_size – 1 => H_size ; index / 2 => parent .

Lastly, If the new element is larger or smaller than the parent node, recursively exchange it with the parent node until the heap order is restored.

For example : if index > 1 > parent ; shiftup (H,index)

Else: maxheap(H,index)

16 A few advanced questions on complexity analysis and algorithm design

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

so hard ....

Outcome	Weight
Implement Solutions	♦♦♦♦♦

so hard ....

Outcome	Weight
Document solutions	♦♦♦♦♦

so hard ....

Date	Author	Comment
2025/04/30 18:34	Yi Guan	Ready to Mark
2025/04/30 18:34	Yi Guan	hihi
2025/05/12 10:16	Svitlana Prypoten	Part B is not correct, check the rules, this time T goes to the next level, and H stop.Part C required more details:- Clarify indexing: Instead of Current.left, use index-based access (e.g., 2 * index, 2 * index + 1).- Add bounds checks: Ensure left and right child indices are within the heap size before inserting.- Specify what's stored in Aux_heap: Store both the value and index as a tuple so you can access children later.
2025/05/12 10:16	Svitlana Prypoten	Fix and Resubmit
2025/05/12 20:38	Yi Guan	Ready to Mark
2025/05/12 20:38	Svitlana Prypoten	Time Exceeded
2025/05/12 20:38	Yi Guan	Hi Miss, I have fixed questions B and C and added more explanations to support my pseudocode.
2025/05/14 12:30	Svitlana Prypoten	Happy to discuss or waiting for video
2025/05/14 12:30	Svitlana Prypoten	Discuss

Deakin University

Data Structures and Algorithms

OnTrack Submission

A few advanced questions on complexity analysis and algorithm design

Submitted By:
Yi Guan
ppv
2025/05/12 20:38

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

so hard ....

May 12, 2025

- **Part A**

Tim and Te are two students studying algorithms and data structures. They are working on a web-system that allows selection of a time interval in history to obtain a daily maximum air temperature during the specified interval. Currently, Te stores the information about the temperature in a list sorted chronologically. Thus, if there are n days, then $t_1, \dots, t_n$ represents the list of temperatures, each associated with a particular date. A user's query consists of a pair (i, j) with $1 \leq i < j \leq n$, and the system is supposed to return $\max(t_i, \dots, t_j)$. Assume for simplicity that n is a power of 2, i.e., $n = 2^k$ for $k = 1, 2, 3, \dots$.

Tim believes that they should minimize the amount of computation per query since there will likely be a lot of traffic incoming to the system. His **first solution** is to precompute and store $\max(t_i, \dots, t_j)$ for every possible input (i, j). By doing so, he can respond to users' queries quicker. However, his concern is that such a solution needs storage of all pairs (values); thus, it requires a bit too much memory.

To decrease the memory consumption, Tim's **second solution** allows a small amount of computation per query. His idea is, rather than just store the $\max(e_i, \dots, e_j)$ for every (i, j), calculate and record values precomputed in such a way that, for any user query, the server can retrieve two precomputed values and take the maximum of the two to return the final answer. Specifically, for a given list $t_1, \dots, t_n$, he proposes to split it into two sub-lists $t_1, \dots, t_{\frac{n}{2}}$ and $t_{\frac{n}{2}+1}, \dots, t_n$, and record

- $\max\left(t_i, \dots, t_{\frac{n}{2}}\right)$ for each $1 \leq i \leq \frac{n}{2}$, and
- $\max\left(t_{\frac{n}{2}+1}, \dots, t_j\right)$ for each $\frac{n}{2} < j \leq n$.

He then recursively splits the two lists into two equal parts and stops the recursion when the list's size becomes equal to 1.

Your task is to evaluate the (best- and worst-case) time and space complexity for all three solutions: the Te's current approach, the first and the second solution proposed by Tim. You need to follow their ideas about the data structures and show how values need to be precomputed and stored. Your answer can be in the form of a formal proof or a written explanation. You should make small numeric examples that will also help you to grasp the data structures.

A : For Te's system, the list ti.....tj for each query(i,j).

The time complexity will be O(j – i + 1) = O(n) per query. For his best/worst case.

The space complexity : O(1).

For Tim's first solution, precompute and store the maximum for all possible pairs (i,j).

Total pairs is n(n-1) / 2.

Precompute using dynamic: max(i,j) = max(max(i,j –1) , tj).

Time complexity will be : precomputation O(n^2).

Query : O(1).

Space complexity: O(n^2).

For Tim's second solution, let recursively split the list and store partial maxima.

Precomputation:

Each recursion level splits the list into halves.

At level l, store 2^l intervals of size n/2^l.

Total nodes : 2n-1.

Space complexity : O(n log n).

Time complexity : O(log n) per query.

- **Part B**

Based on your previous research about the Skip-List data structure, you should know that the insertion into a skip-list asks for flipping a coin until a head occurs. This rule determines the height for a tower representing a new element. So, for example, if we flipped two tails before a head, then we would insert an element of height 3. In all the examples you have seen before, the probability of flipping a head was exactly 1/2.

Now assume that the coin is biased, so the probability of flipping a tail on a given throw is 1/10. Derive an expression for the probability of producing an element of height h with this biased coin. Describe the consequence of this bias in terms of insertion into the skip-list and its structure.

A:

Probability of Height h

Original skip-List: Height h occurs with probability (1/2)^h

Biased Coin:

Continue to increase the number of layers each time you throw a T and stop when you throw an H.

Probability of tail = 1/10, head= 9/10 .

To generate height h, you need to throw T （h−1） times in succession, followed by 1 throw of H.

P(h)=(1/10)^h-1 * (9/10).

Consequences of Bias:

1. Higher Average Height: Elements are more likely to have taller towers.

Compared to a standard skip list (expected height 2), the biased coin results in a significantly lower average element height.

2. Reduced space overhead: Fewer layers mean fewer pointers per element, reducing memory usage.

3. Search efficiency may decrease: The number of high-level nodes decreases, resulting in lower-level nodes to be traversed during search, and the time complexity may approach O(n).

- **Part C**

After watching Lecture 6, you likely know that the maximum element of a binary max-heap can be obtained by means of the Max operation in a constant $O(1)$ time. However, there is an approach to find the k^{th} maximum element in $O(k \log k)$ time, where $1 \leq k \leq n$. Keep in mind that both insertion and removal in a binary max-heap of size n takes a $O(\log n)$ time. Your task is to design such an algorithm and, as the answer, provide its pseudocode complemented by your explanation.

A:

Data structure: The main heap is stored in an array, with indexes starting from 1, satisfying: The left child of node i is 2i. The right child of node i is 2i + 1.

Use the auxiliary maximum heap (AuxHeap) to store tuples (value, index), sorted by value.

Initialization: Insert the root node of the main heap (index 1, value heap[1]) into AuxHeap.

for i = 1 to k:

if AuxHeap is empty:

break

(current_val, current_idx) = AuxHeap.extract_max()

result.append(current_val)

Loop k times:

a. Extract the maximum value: Extract the current maximum value from AuxHeap.

b. Insert child nodes: Calculate the left child node index 2i and the right child node index 2i + 1. If the index does not exceed the heap size n, insert the value and index of the child node into AuxHeap.

Bounds check: Before inserting a child node, verify whether the index is valid.

// insert left （if exist）

left_idx = 2 * current_idx

if left_idx <= n:

AuxHeap.insert((heap[left_idx], left_idx))

// insert right （if exist）

right_idx = 2 * current_idx + 1

if right_idx <= n:

AuxHeap.insert((heap[right_idx], right_idx))

```
return result[k-1]
```

Each operation: extract_max and insert has a time complexity of O(log K), because the size of the auxiliary heap is at most O(k).

Total time: O(K log K).

References:

1. https://www.geeksforgeeks.org/segment-tree-efficient-implementation/
2. https://mitpress.mit.edu/9780262530910/introduction-to-algorithms/

17 Implementation of the Binary Heap

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

hard as usual.....

Outcome	Weight
Implement Solutions	♦♦♦♦♦

hard as usual.....

Outcome	Weight
Document solutions	♦♦♦♦♦

hard as usual.....

Date	Author	Comment
2025/05/05 18:56	Yi Guan	Ready to Mark
2025/05/05 18:56	Svitlana Prypoten	Automated Assessment Started
2025/05/05 18:56	Yi Guan	hihi
2025/05/09 19:40	Svitlana Prypoten	Discuss
2025/05/09 19:41	Svitlana Prypoten	Happy to discuss or waiting for video
2025/05/16 03:06	Yi Guan	Hi Miss this is the video for this task.
2025/05/16 03:06	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=70 8ac0-4e8f-99ff-b2de01149037
2025/05/16 03:06	Yi Guan	:grinning:
2025/05/18 09:44	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Implementation of the Binary Heap

Submitted By:
Yi Guan
ppv
2025/05/05 18:56

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

hard as usual.....

May 5, 2025

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Heap
8  {
9      public class Heap<K, D> where K : IComparable<K>
10     {
11
12         // This is a nested Node class whose purpose is to represent a node of a
           ↪  heap.
13         private class Node : IHeapifyable<K, D>
14         {
15             // The Data field represents a payload.
16             public D Data { get; set; }
17             // The Key field is used to order elements with regard to the Binary Min
               ↪  (Max) Heap Policy, i.e. the key of the parent node is smaller
               ↪  (larger) than
18  the key of its children.
19             public K Key { get; set; }
20             // The Position field reflects the location (index) of the node in the
               ↪  array-based internal data structure.
21             public int Position { get; set; }
22
23             public Node(K key, D value, int position)
24             {
25                 Data = value;
26                 Key = key;
27                 Position = position;
28             }
29
30             // This is a ToString() method of the Node class.
31             // It prints out a node as a tuple ('key value','payload','index')}.
32             public override string ToString()
33             {
34                 return "(" + Key.ToString() + "," + Data.ToString() + "," +
                   ↪  Position + ")";
35             }
36         }
37
38         // ------------------------------------------------------------------------
           ↪  ---------
39         // Here the description of the methods and attributes of the Heap<K, D>
           ↪  class starts
40
41         public int Count { get; private set; }
42
```

```
43          // The data nodes of the Heap<K, D> are stored internally in the List
            ↪ collection.
44          // Note that the element with index 0 is a dummy node.
45          // The top-most element of the heap returned to the user via Min() is
            ↪ indexed as 1.
46          private List<Node> data = new List<Node>();
47
48          // We refer to a given comparer to order elements in the heap.
49          // Depending on the comparer, we may get either a binary Min-Heap or a
            ↪ binary Max-Heap.
50          // In the former case, the comparer must order elements in the ascending
            ↪ order of the keys, and does this in the descending order in the latter
            ↪ case.
51          private IComparer<K> comparer;
52
53          // We expect the user to specify the comparer via the given argument.
54          public Heap(IComparer<K> comparer)
55          {
56              this.comparer = comparer;
57
58              // We use a default comparer when the user is unable to provide one.
59              // This implies the restriction on type K such as 'where K :
                ↪ IComparable<K>' in the class declaration.
60              if (this.comparer == null) this.comparer = Comparer<K>.Default;
61
62              // We simplify the implementation of the Heap<K, D> by creating a dummy
                ↪ node at position 0.
63              // This allows to achieve the following property:
64              // The children of a node with index i have indices 2*i and 2*i+1 (if
                ↪ they exist).
65              data.Add(new Node(default(K), default(D), 0));
66          }
67
68          // This method returns the top-most (either a minimum or a maximum) of the
            ↪ heap.
69          // It does not delete the element, just returns the node casted to the
            ↪ IHeapifyable<K, D> interface.
70          public IHeapifyable<K, D> Min()
71          {
72              if (Count == 0) throw new InvalidOperationException("The heap is
                ↪ empty.");
73              return data[1];
74          }
75
76          // Insertion to the Heap<K, D> is based on the private UpHeap() method
77          public IHeapifyable<K, D> Insert(K key, D value)
78          {
79              Count++;
80              Node node = new Node(key, value, Count);
81              data.Add(node);
82              UpHeap(Count);
83              return node;
84          }
85
```

```
86          private void UpHeap(int start)
87          {
88              int position = start;
89              while (position != 1)
90              {
91                  if (comparer.Compare(data[position].Key, data[position / 2].Key) <
                    ↪  0) Swap(position, position / 2);
92                  position = position / 2;
93              }
94          }
95
96          // This method swaps two elements in the list representing the heap.
97          // Use it when you need to swap nodes in your solution, e.g. in DownHeap()
                ↪  that you will need to develop.
98          private void Swap(int from, int to)
99          {
100             Node temp = data[from];
101             data[from] = data[to];
102             data[to] = temp;
103             data[to].Position = to;
104             data[from].Position = from;
105         }
106
107         public void Clear()
108         {
109             for (int i = 0; i <= Count; i++) data[i].Position = -1;
110             data.Clear();
111             data.Add(new Node(default(K), default(D), 0));
112             Count = 0;
113         }
114
115         public override string ToString()
116         {
117             if (Count == 0) return "[]";
118             StringBuilder s = new StringBuilder();
119             s.Append("[");
120             for (int i = 0; i < Count; i++)
121             {
122                 s.Append(data[i + 1]);
123                 if (i + 1 < Count) s.Append(",");
124             }
125             s.Append("]");
126             return s.ToString();
127         }
128
129         // TODO: Your task is to implement all the remaining methods.
130         // Read the instruction carefully, study the code examples from above as
                ↪  they should help you to write the rest of the code.
131         public IHeapifyable<K, D> Delete()
132         {
133             if (Count == 0) throw new InvalidOperationException("The heap is
                    ↪  empty.");
134             IHeapifyable<K, D> min = data[1];
135             data[1] = data[Count];
```

```
136             data[1].Position = 1;
137             data.RemoveAt(Count);
138             Count--;
139             if (Count > 0) DownHeap(1);
140             return min;
141         }
142 
143         // Builds a minimum binary heap using the specified data according to the
            ↪  bottom-up approach.
144         private void DownHeap(int start)
145         {
146             int current = start;
147             while (true)
148             {
149                 int left = 2 * current;
150                 int right = 2 * current + 1;
151                 int smallest = current;
152 
153                 if (left <= Count && comparer.Compare(data[left].Key,
                    ↪  data[smallest].Key) < 0)
154                     smallest = left;
155                 if (right <= Count && comparer.Compare(data[right].Key,
                    ↪  data[smallest].Key) < 0)
156                     smallest = right;
157 
158                 if (smallest == current) break;
159 
160                 Swap(current, smallest);
161                 current = smallest;
162             }
163         }
164         public IHeapifyable<K, D>[] BuildHeap(K[] keys, D[] data)
165         {
166             if (Count != 0) throw new InvalidOperationException("Heap is not
                ↪  empty.");
167             if (keys.Length != data.Length) throw new ArgumentException("Keys and
                ↪  data arrays must be of the same length.");
168 
169             Count = keys.Length;
170             List<Node> newNodes = new List<Node>(keys.Select((k, i) => new Node(k,
                ↪  data[i], i + 1)));
171             this.data = new List<Node> { new Node(default(K), default(D), 0) };
172             this.data.AddRange(newNodes);
173 
174             for (int i = Count / 2; i >= 1; i--)
175                 DownHeap(i);
176 
177             return newNodes.Cast<IHeapifyable<K, D>>().ToArray();
178         }
179 
180         public void DecreaseKey(IHeapifyable<K, D> element, K new_key)
181         {
182             Node node = element as Node;
```

```
183             if (node == null || node.Position > Count || data[node.Position] !=
                ↪  node)
184                 throw new InvalidOperationException("Element is inconsistent with
                    ↪  the heap.");
185
186             if (comparer.Compare(new_key, node.Key) >= 0)
187                 throw new ArgumentException("New key is not smaller than the current
                    ↪  key.");
188
189             node.Key = new_key;
190             UpHeap(node.Position);
191         }
192
193     }
194 }
```

18 AVL trees

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

using a lot of references

Outcome	Weight
Implement Solutions	♦♦♦♦♦

using a lot of references

Outcome	Weight
Document solutions	♦♦♦♦♦

using a lot of references

Date	Author	Comment
2025/05/05 18:57	Yi Guan	Ready to Mark
2025/05/05 18:57	Yi Guan	hihi
2025/05/12 21:44	Svitlana Prypoten	Discuss
2025/05/12 21:44	Svitlana Prypoten	Happy to discuss or waiting for video
2025/05/23 01:32	Yi Guan	Hi.Miss this is the video link for this task:https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?i 6167-4321-aad6-b2e500fc529c
2025/05/24 09:34	Svitlana Prypoten	Please update your video. You do a lot of mistakes explaining how you build AVL tree. Make sure you review AVL tree rules.
2025/05/24 09:39	Svitlana Prypoten	Your explanation of pseudocode is not convinced me you understand how it works and why it's time complexity O(n). Please redo.
2025/05/24 14:18	Yi Guan	Got it Miss, I'll be upload the video later today.
2025/05/24 14:19	Yi Guan	:sweat_smile:
2025/05/24 17:11	Yi Guan	Hi Miss this is the redo video link for this task.
2025/05/24 17:11	Yi Guan	https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=c7 5891-49f0-8cb8-b2e7006e6003
2025/05/24 17:12	Yi Guan	:sweat_smile:hopefully this time I explained everything for it.
2025/05/24 18:37	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

AVL trees

Submitted By:
Yi Guan
ppv
2025/05/05 18:57

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

using a lot of references

May 5, 2025

1. Draw a series of figures demonstrating the insertion of the values

   20, 9, 3, 7, 5, 8, 25, 30, 15, 6, 17

   into an initially ***empty*** AVL tree. Insert the values in the order they appear in the given sequence. You must:

   - Show the resulting AVL tree immediately before and after each insertion step that causes the tree's rebalancing.
   - Calculate the ***balance degree*** (as the difference between the heights of the left and the right subtrees rooted at a node) and label each node of the AVL tree before and after the necessary rebalancing.
   - Clearly indicate the node(s) at which the rotation is performed. Here, let v be the first node you encounter in going up from the newly added node, say z, towards the root of the AVL tree T such that v is unbalanced. Then, let x be the child of v such that x is the ancestor of z. Determine whether the subtree of T rooted at v is ***right*** or ***left heavy***. Similarly, indicate whether the subtree of T rooted at x is ***right*** or ***left heavy***.
   - Performing a ***single*** or a ***double rotation*** as a rebalancing (repairing) operation on the AVL tree, specify the type of the rotation that you apply: ***Single Left Rotation***, ***Single Right Rotation***, ***Left-Right Rotation***, or ***Right-Left Rotation***.
   - Each time a new value is added, ensure that both the ***Binary Search Tree Property*** and the ***Height-Balance (AVL Tree) Property*** are maintained.

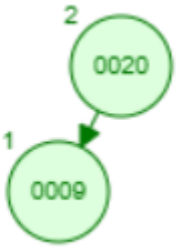
A :

BF : balance factor.

1. insert 20 :

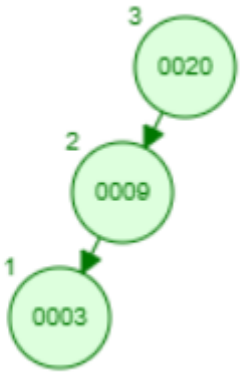
   BF = 0

2. insert 9



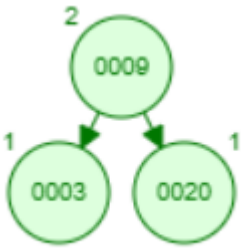
9 < 20, shift it to the left subtree . 20 BF = 1 ; 9 BF = 0

3. insert 3



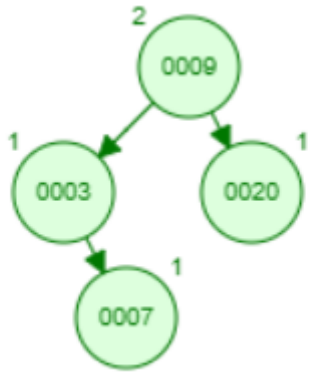
3 < 9 < 20, shift to the left subtree.

Because 20 BF = 2 , the left subtree of the left subtree is too high, so it need to single right rotation with 9 as axis. Now the BF of 9 and 20 are 0 .



Because 20 BF = 2 , the left subtree of the left subtree is too high, so it need to single right rotation with 9 as axis. Now the BF of 9 and 20 are 0 .

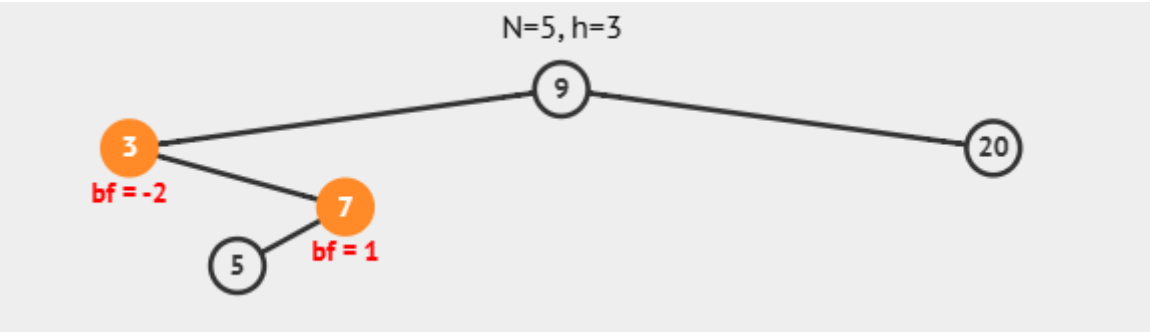
4. insert 7



Because 7 < 9 ; 7 > 3. So it will be on the right subtree of the 3.

BF of 9 is 2-1 = 1 so it's ok,

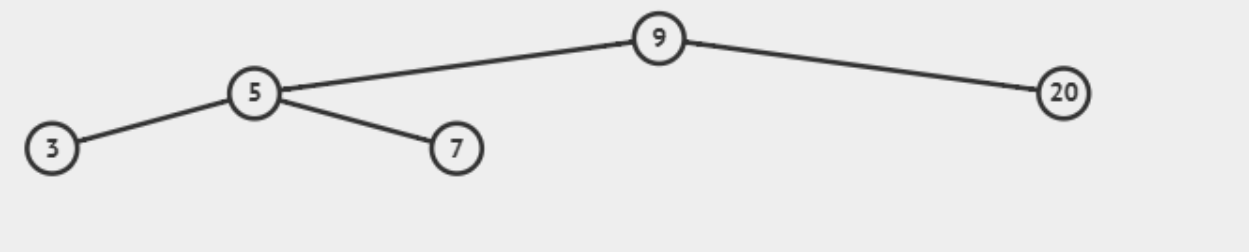
5. insert 5



Because 3 < 5 < 7 , it will belong to left subtree of 7.

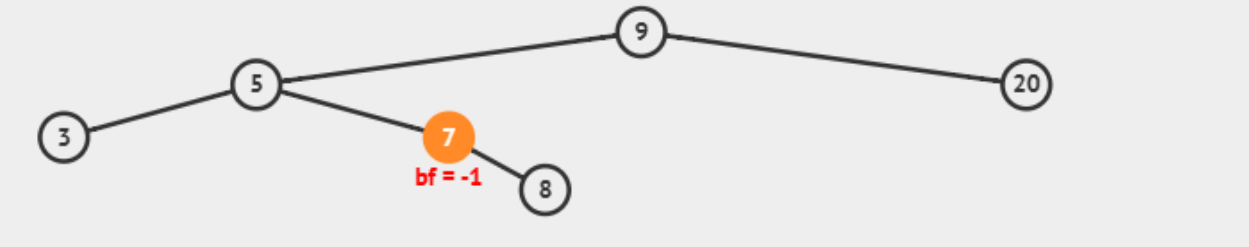
But the BF of 3 will be 0-2=-2, which is not balanced. BF of 7 is 1 .

Do a single right rotate with 7 axis, then do another single left rotation of 3 with 5 axis.



Now the BF of 5 is 0 and BF of 9 is 1.

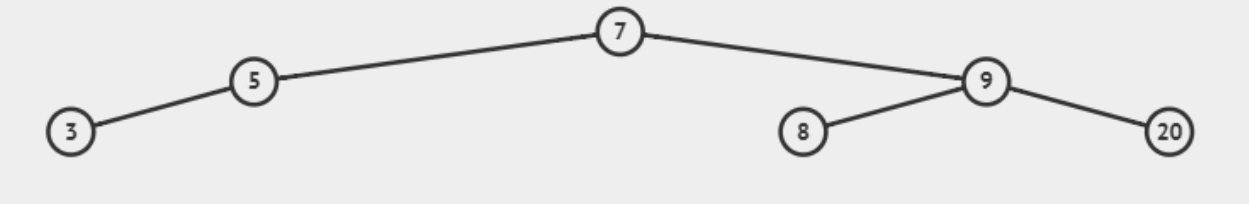
6. insert 8



Because 8 < 9 ; 8 > 5 , 7 ; so it belongs to right subtree of 7.

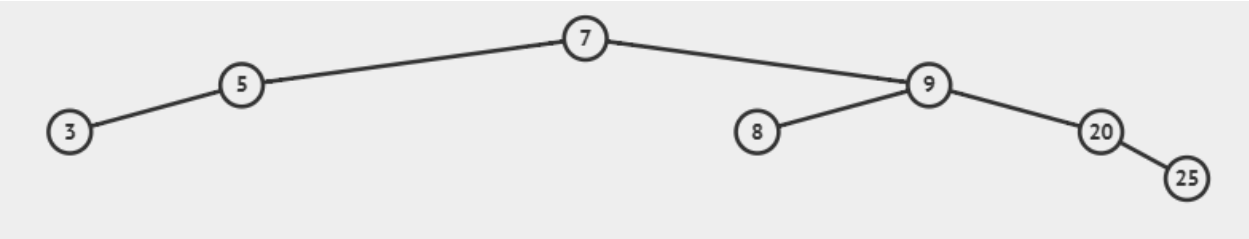
But the BF of 7 and 5 are 0-1 = -1 ; BF of 9 is 3-1=2 which is not balanced.

Do a single left rotation of 7 with 5 axis , then do another single right rotation of 7 with 9 axis.



Now the BF of 7 is 0, balanced.

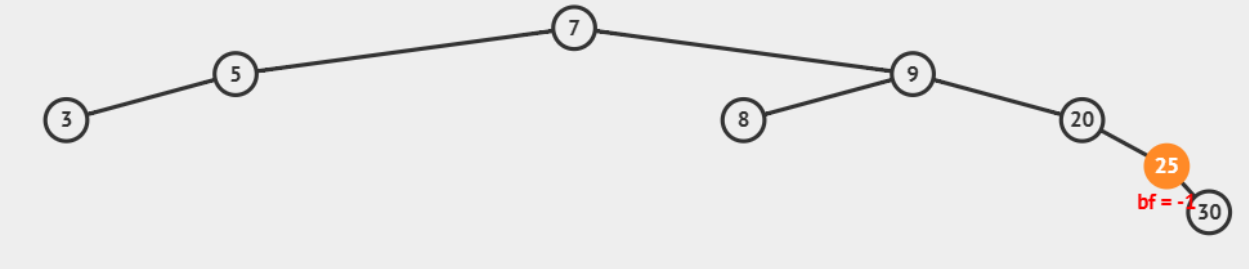
7. insert 25



Because 25 is the largest so far, so it will be on the right subtree of 20.

Because the BF of 7 is 2 – 3 = -1, it's balanced.

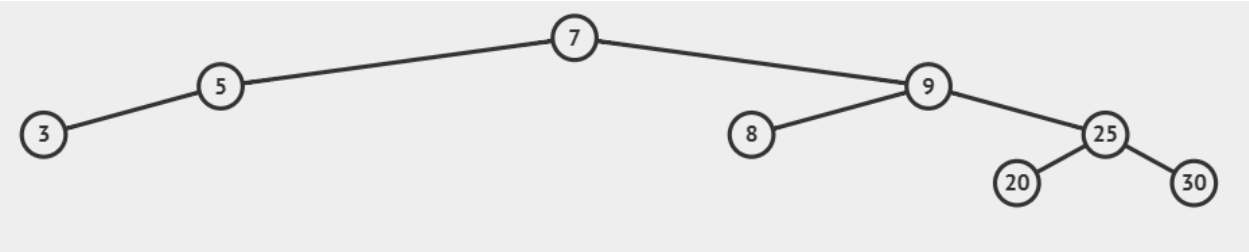
8. insert 30



Because 30 > 25 , so it will be on the right subtree of 25.

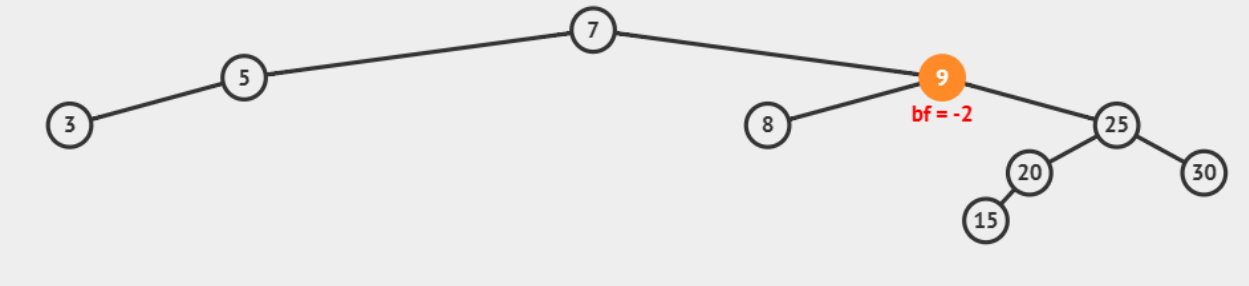
But the BF of 20 is 0 – 2 = -2 which is not balance.

So we need to do a single left rotation for 20 with 25 axis.



Now the BF of 25 is 0 ; BF of 9 is –1 ; BF of 7 is –1 . Balanced.

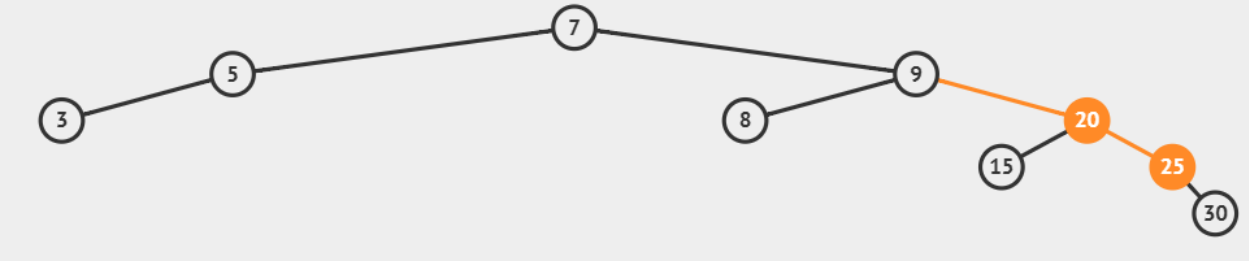
9. insert 15



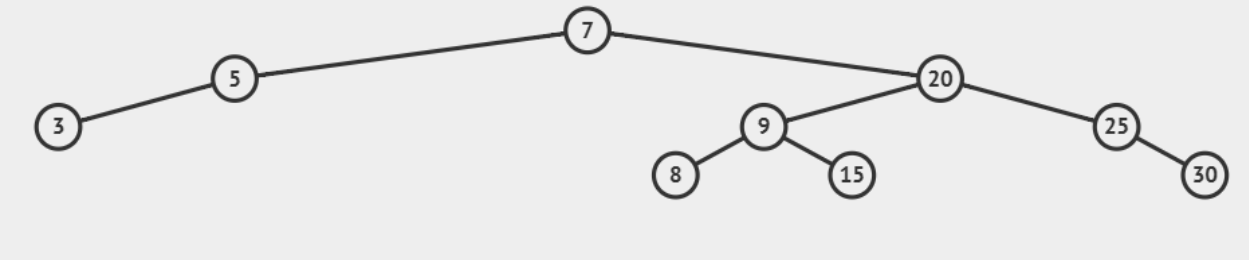
Because 20 , 25 >15 > 9 , so it will belong to left subtree of 20.

But the BF of 9 is 1 – 3 = -2 which is not balanced.

So we need to do a single right rotation of 20 with 25 axis .

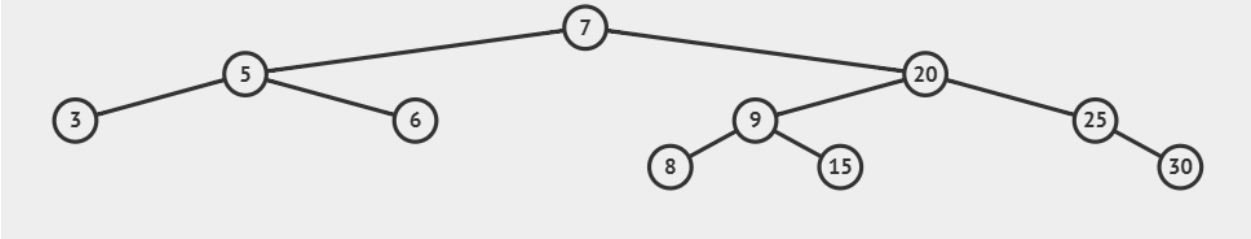


Then we need to do another left rotation of 9 with 20 axis.



Now the BF of 25 is –1 ; BF of 20 is 0; BF of 7 is –1. Balanced.

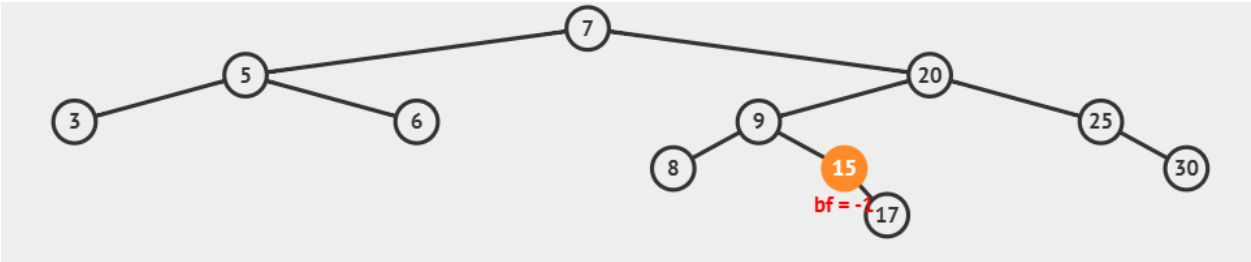
10. insert 6



Because 7 > 6 > 5, so it will be on the right subtree of 5.

The BF of 5 is 0 and BF of 7 is –1. Balanced .

11. insert 17

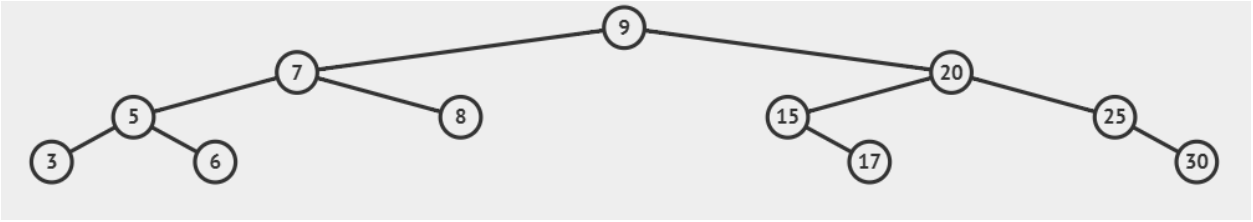


Because 20 < 15 > 9, so it belongs to right subtree of 15.

But the BF of 7 is –2 , not balanced.

Make it single right rotation of 9 with 20 axis.

Then do another single left rotation of 9 with 7 axis.



The BF of 9 ; 5 ;20 are 0 .

BF of 7 is 1; BF of 25 is –1. Balanced.

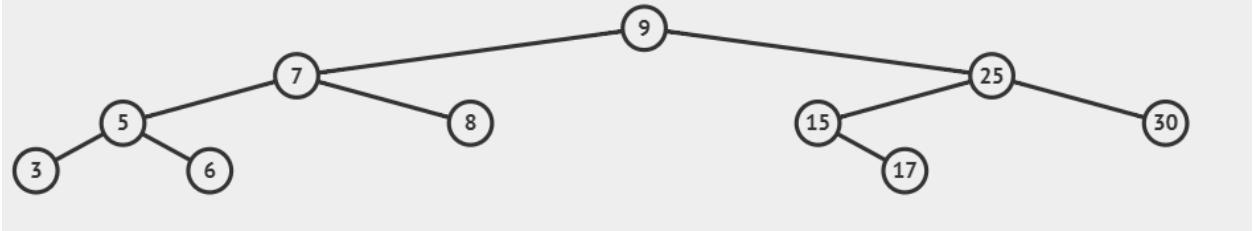
2. Now, draw a series of figures showing the deletion of the values:

   20, 15, 8, 25, 30, 9, 17, 5, 6, 3, 7

   from the AVL tree built in the previous part of the task. Delete the values in the order they appear in the given sequence. Draw the AVL tree after each deletion and rotation (if any) and complement each of the figures with all the aforementioned details.

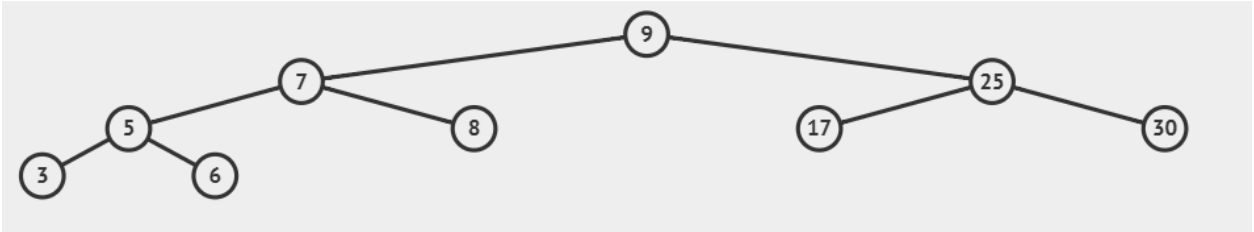
   When you delete a node with two children, you must always replace it with ***'the largest element smaller than the key of the deleted node'*** (as opposed to the other possible rule: 'the smallest element larger than the key of the deleted node'). **Following this rule is important** to let us check your solution quickly.

1. Delete 20



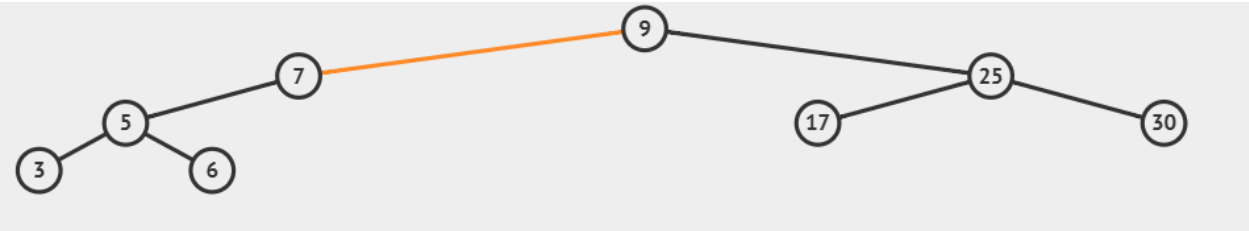
The BF of 9 , 25 , 7 are 1; Balanced.

2. Delete 15



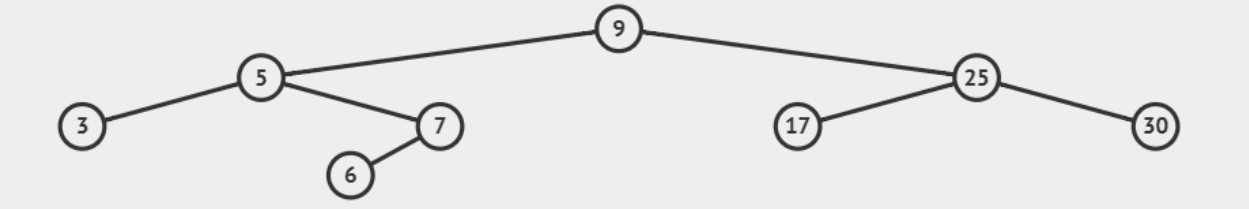
BF of 9 is 1 ; BF of 25 is 0. Balanced.

3. Delete 8



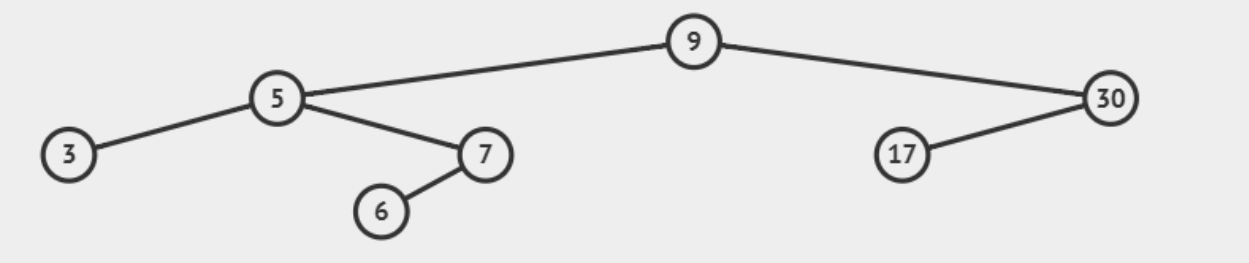
Because the BF of 7 is 2, which is not balanced.

Need to do a single right rotation of 5 with 7 axis.



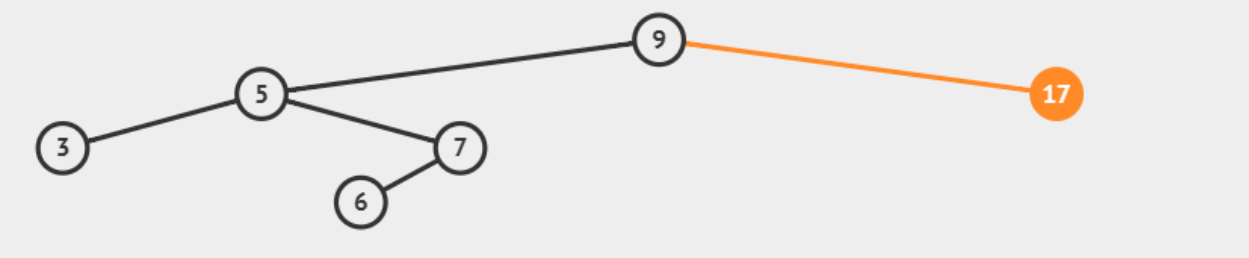
Now the BF of 5 is –1 ; BF of 9 is 1. Balanced.

4. Delete 25

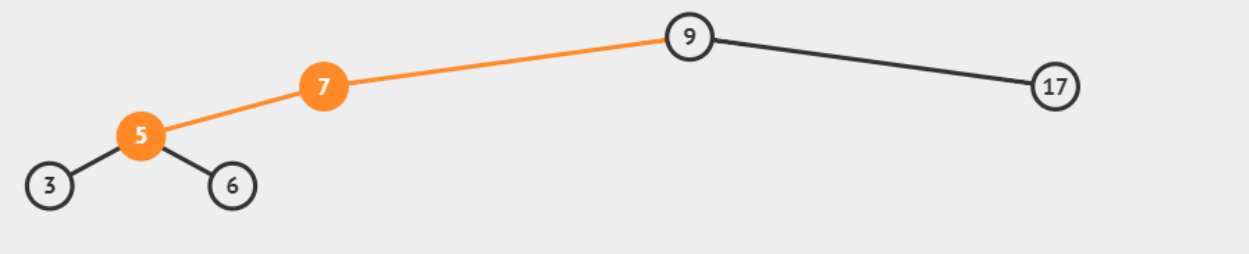


BF of 30 is 1 ; BF of 9 is 1. Balanced.

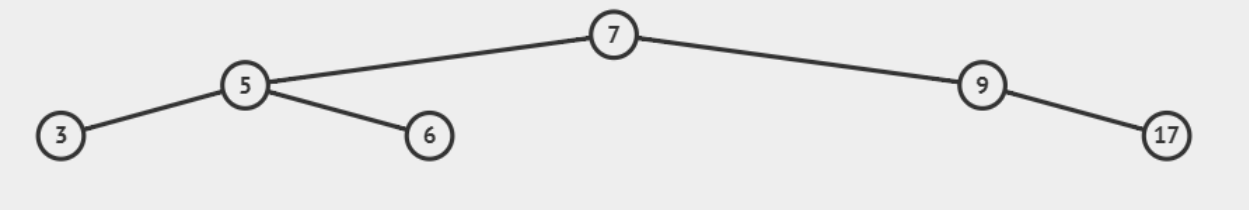
5. Delete 30



After deleted 30; the BF of 9 is 2. The BF of 5 is –1 .
Need to do a single left rotation of 7 with 5 axis.

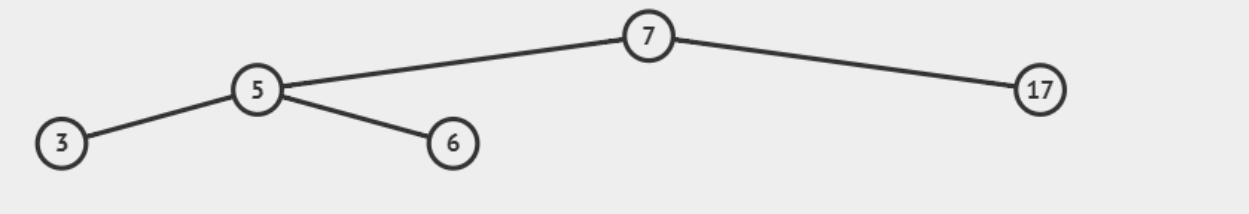


Then do another right rotation of 7 with 9 axis.



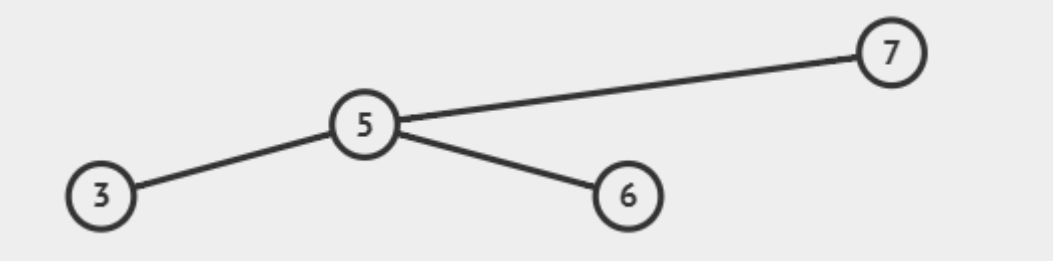
Now the BF of 7 is 0 ; BF of 9 is –1; BF of 5 is 0. Balanced .

6. Delete 9



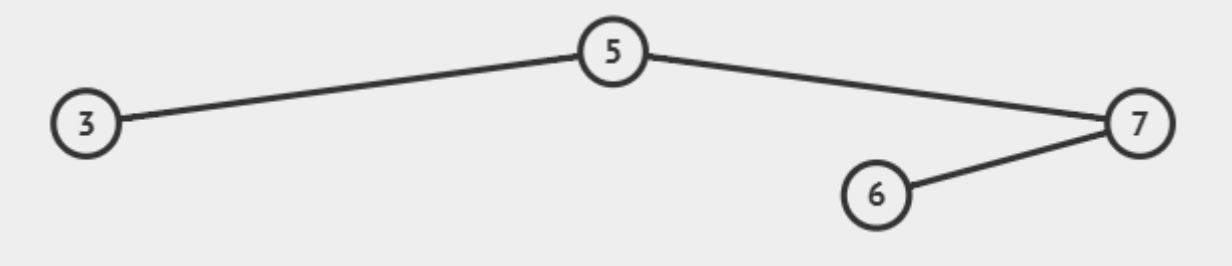
The BF of 7 is 1 ; BF of 5 is 0 ; Balanced.

7. Delete 17



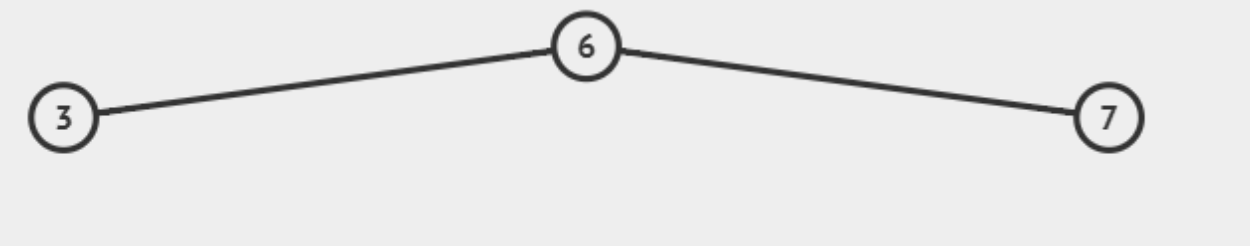
Because the BF of 7 is 2, it's not balance.

Need to do a single right rotation of 5 with 7 axis.



Now the BF of 5 is –1, which is balanced.

8. Delete 5



The BF of 6 is 0, balanced.

9. Delete 6

BF of 7 is 1, balanced.

10. Delete 3

Only 7 left , the BF is 0.

11. Delete 7

Nothing left .

3. Develop a linear $O(n)$ time algorithm that accepts an arbitrary binary search tree as an input and creates a **balanced** binary search tree whose height is $O(\log n)$. As the answer to this question, provide its pseudocode and description. Explain why you believe that your algorithm takes a linear time.

A: the pseudocode is :

Algorithm BalanceBsT(root):

// In-order traversal to get an ordered list of nodes (0(n))

nodes = InOrderTraversal(root)

// Recursively construct a balanced tree (0(n))

return BuildBalancedBsT(nodes,0, len(nodes)-1)

Function BuildBalancedBST(nodes, start,end):

if start > end:

return null

mid =(start +end)//2

root = nodes[mid]

root.left = BuildBalancedBST(nodes, start, mid-1)

root.right =BuildBalancedBST(nodes,mid +1,end)

return root

The reason is :

1. Time complexity of in-order traversal: O(n)
   In-order traversal requires visiting every node in the binary search tree once. For a tree containing several nodes, each node is visited once, and the time complexity is strictly O(n).

2. Time complexity of recursively building a balanced tree: O(n) The core of recursive construction is the divide-and-conquer strategy:

Select the middle element as the root: by calculating the midpoint of the interval mid = start+end/2, the time is O(1).

Recursively process the left and right halves: each recursion divides the array into two halves, but each node is only visited once.

For the total time complexity is O(n) + O(n) = O(n)

Reference:

1. https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
2. https://www.youtube.com/watch?v=zP2xbKerIds
3. https://tedwu1215.medium.com/avl-tree-%E9%AB%98%E5%BA%A6%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%85%83%E6%90%9C%E5%B0%8B%E6%A8%B9%E4%BB%8B%E7%B4%B9%E8%88%87%E7%AF%84%E4%BE%8B-15a82c5b778f ; Jul 4, 2023 ; 碼農思考中 - Ted;
4. https://blog.csdn.net/qq_28205153/article/details/51547495 ; 2016-05-31 17:20:44 published; TimeShatter
5. https://visualgo.net/en/bst

19 Helping Alex with treasure hunting: A problem-solving task

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

hard as usual.

Outcome	Weight
Implement Solutions	♦♦♦♦♦

hard as usual.

Outcome	Weight
Document solutions	♦♦♦♦♦

hard as usual.

Date	Author	Comment
2025/05/07 18:26	Yi Guan	Ready to Mark
2025/05/07 18:26	Svitlana Prypoten	Automated Assessment Started
2025/05/07 18:26	Yi Guan	hihi
2025/05/12 18:33	Svitlana Prypoten	Discuss
2025/05/12 18:33	Svitlana Prypoten	Happy to discuss or waiting for video
2025/05/23 01:31	Yi Guan	Hi, Miss this is the video for this task : https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=93 fd8d-4492-82da-b2e500fc567b

Deakin University

Data Structures and Algorithms

OnTrack Submission

Helping Alex with treasure hunting: A problem-solving task

Submitted By:
Yi Guan
ppv
2025/05/07 18:26

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

hard as usual.

May 7, 2025

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```
using System;
using System.Text;
using System.Text.RegularExpressions;
using System.Collections;
using System.Collections.Generic;

namespace BoxOfCoins
{
    public class BoxOfCoins
    {
        public static int Solve(int[] boxes)
        {
            int n = boxes.Length;
            int[,] dp = new int[n, n];

            for (int i = 0; i < n; i++)
            {
                dp[i, i] = boxes[i];
            }

            for (int length = 2; length <= n; length++)
            {
                for (int i = 0; i <= n - length; i++)
                {
                    int j = i + length - 1;
                    int pickLeft = boxes[i] - dp[i + 1, j];
                    int pickRight = boxes[j] - dp[i, j - 1];
                    dp[i, j] = Math.Max(pickLeft, pickRight);
                }
            }

            return dp[0, n - 1];
        }
    }
}
```

The problem exhibits optimal substructure and overlapping subproblems.

Dynamic Programming (DP) efficiently solves this by storing intermediate results.

Steps:

1. Dp Table Definition:

Dp [i , j] : Maximum net score differences the current player can achieve from boxes i to j.

2. Base Case:

For a single box (i == j), the player takes it:

Dp [i , i] = boxes[i]

Dp [i] [j] = max

(

boxes[i] - Dp [i + 1] [j], // Take left, opponent faces i+1, j

boxes[j] - Dp [i] [j - 1] // Take right, opponent faces i , j – 1 .

)

Example Input: [2, 7, 3]

1. Base Cases:

Dp [0] [0] = 2 , Dp [1] [1] = 7 , Dp [2] [2] = 32.

Length 2 Intervals:

[0,1] : max(2 – 7 , 7 – 2) = max (- 5 , 5) = 5

[1 , 2] : max(7 – 3 , 3 – 7) = 43.

Length 3 Interval ([0 , 2]) :

max(2 – 4 , 3 – 5) = max(-2 , -2) = -2

Output: -2(Alex's net score).

Time Complexity :

Time: O (n^2) due to nested loops over intervals.

Space: O (n^2) for the Dp table.

This DP approach ensures both players' optimal strategies are evaluated without redundancy. By solving smaller subproblems first, the algorithm efficiently computes the final result for the entire array.

20 Graphs, paths, and Miss Marple

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦♦♦

as hard as usually....

Outcome	Weight
Implement Solutions	♦♦♦♦♦

as hard as usually....

Outcome	Weight
Document solutions	♦♦♦♦♦

as hard as usually....

Date	Author	Comment
2025/05/12 19:47	Yi Guan	Ready to Mark
2025/05/12 19:47	Yi Guan	hihi
2025/05/21 23:04	Svitlana Prypoten	Part D is not correct.
2025/05/21 23:04	Svitlana Prypoten	Fix and Resubmit
2025/05/23 01:36	Yi Guan	got it Miss, I'll fix it later today
2025/05/23 02:32	Yi Guan	Ready to Mark
2025/05/23 02:32	Svitlana Prypoten	Time Exceeded
2025/05/23 02:32	Yi Guan	This time should be all correct.
2025/05/27 15:23	Yi Guan	Miss, should I film a video for this task or waiting for feedback ? Because the deadline is this Friday.....
2025/05/27 15:23	Yi Guan	:smiling_face_with_tear:
2025/05/28 09:53	Svitlana Prypoten	All good now, waiting for video
2025/05/28 09:53	Svitlana Prypoten	Discuss
2025/05/29 16:23	Yi Guan	Got it Miss, I'll film it later today
2025/05/30 03:32	Yi Guan	Hi Miss, this is the video link for this task :https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=a e5a8-4679-8e5b-b2ec011ecd0e
2025/05/30 03:33	Yi Guan	:sweat_smile:
2025/05/31 09:31	Svitlana Prypoten	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Graphs, paths, and Miss Marple

Submitted By:
Yi Guan
ppv
2025/05/23 02:32

Tutor:
Svitlana Prypoten

Outcome	**Weight**
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

as hard as usually....

May 23, 2025

- **Part A**

Miss Marple is at a train station in a foreign town. She wants to select a hotel that has the *maximum number of shortest paths* from the train station. She thinks that this should reduce the risk of getting lost. Suppose she gives you a city map represented via a graph $G = \langle V, E \rangle$ with $n = |V|$ locations and $m = |E|$ edges connecting the locations. Each edge connecting a pair of directly connected locations has a *unit distance*, say 1. Help Miss Marple to find a proper hotel by designing a $O(n + m)$ runtime algorithm that finds the number of shortest paths between the train station, located at node s and every hotel on the map. You need to convince her that your algorithm is correct, and it does find all possible shortest paths. Your solution must contain an explanation and a pseudocode.

A :

We need to count the number of shortest paths from the starting point (train station) to each hotel. Since the weight of all edges in the graph is 1, the shortest path is equivalent to the path with the least number of edges. Breadth-first search (BFS) is a natural fit for this problem, because BFS traverses nodes layer by layer, ensuring that the path length recorded when a node is first visited is the shortest distance.

Initialization:

Maintain two arrays:

dist[]: Record the shortest distance from the starting point to each node, the initial value is infinity (indicating unvisited).

count []: Record the number of shortest paths from the starting point to each node, the initial value is 0.

The shortest distance of the starting point is set to 0 (dist[s]=8), and the number of shortest paths is set to 1 (count[s]=1). Use a queue to manage the nodes to be processed and initially add the starting point to the queue.

BFS traversal:

Take node u from the queue and traverse all its adjacent nodes v.

Case 1: If v's shortest distance dist[v] > dist[u] + 1, it means that a shorter path has been found. At this time, update dist[v] = dist[u] + 1, and set count[v] to count[u]. Then add v to the queue.

Case 2: If dist[v]== dist[u]+1, it means that there is another shortest path. At this time, the number of paths is accumulated: count[v] += count [u].

Correctness verification:

BFS layer-by-layer traversal ensures the correctness of the shortest distance. The accumulation logic of the number of paths ensures that all possible path combinations are counted. For example, if node v can be reached through two different paths with the same shortest distance, count [v] will correctly reflect the total number.

Time Complexity

Each node and edge is visited only once, so the time complexity is O(n + m).

- **Part B**

A communication network, such as the Internet, can be modelled as an undirected graph $G = \langle V, E \rangle$. Here, the vertices of set V are the computers of the network, and the edge set E consists of one edge for each pair of computers that are directly connected. We assume that the edges of G are undirected; that is, if there is a direct connection from computer u to computer v, then there is also a direct connection from computer v to computer u.

It is highly desirable for a communication network graph to be ***connected*** so that every computer on the network can communicate, possibly through a series of relays, with any other computer. But networks can change, with some computers failing and other computers being added to the network. It is useful to have a *testing algorithm* that collects information about the current network graph (vertices and edges) at designated times and determines properties related to connectivity.

Describe, in words and a pseudocode, a ***testing algorithm*** that given an undirected graph $G = \langle V, E \rangle$ representing the current network decides whether the network is connected. A graph (the network) is ***connected*** if there is a path from any node to any other node in the graph. You may assume that G is given in the *adjacency list* format. Your algorithm must run in $O(n + m)$ time, where $n = |V|$ is the number of computers of the network, and $m = |E|$ is the number of connecting edges.

A :

The connectivity of an undirected graph requires that there is a path between any two nodes. You can verify whether all nodes are reachable from a certain starting point through a traversal .

1. Initialization:

Use the Boolean array visited[] to mark whether the node has been visited, and the initial value is all false.

Start traversal from any starting point.

2. BFS traversal:

Add the starting point to the queue and mark it as visited.

Loop through the nodes in the queue:

Take out node u and traverse all its adjacent nodes v.

If v has not been visited, mark it as visited and add it to the queue.

3. Connectivity check:

After the traversal, check the visited[] array. If all elements are true, the graph is connected, otherwise it is not connected.

Time complexity

The time complexity of BFS traversal is O(n + m), and the final check takes O(n) time, with a total complexity of O(n + m).

- **Part C**

Given a graph $G = \langle V, E \rangle$, what is to be the runtime complexity of the ***depth-first search*** algorithm, as a function of the number of nodes $n = |V|$ and edges $m = |E|$, if the input graph is represented by an ***adjacency matrix*** instead of an *adjacency list*?

A :

When the graph is represented by an adjacency matrix, the time complexity of DFS is O(n^2).

The adjacency matrix is an n * n two-dimensional array, and each node needs to traverse all possible edges.

DFS needs to visit each node once, a total of n times.

Each time a node is visited, n elements need to be traversed to find adjacent nodes, so the total time complexity is O(n × n) = O(n^2).

Compared with the adjacency list:

The adjacency list only needs to traverse the actual edges (a total of m), so the time complexity is O(n + m).

- **Part D**

Consider the directed graph $G = \langle V, E \rangle$ represented by the following cost adjacency matrix:

$$A = \begin{bmatrix} . & 3 & . & . & . & 10 & . & . & . & . \\ . & . & . & 5 & 11 & . & 7 & 1 & . & . \\ . & . & . & . & 2 & . & . & . & . & . \\ 6 & . & . & . & . & 1 & . & . & . & . \\ . & . & . & . & 0 & . & . & 0 & . & 3 \\ . & . & . & 3 & . & . & . & . & . & . \\ . & . & . & . & . & 2 & . & . & . & . \\ . & . & 10 & . & . & . & . & . & . & . \\ . & . & . & . & . & . & 5 & . & . & . \\ . & . & . & 5 & . & . & . & . & . & . \end{bmatrix}$$

Assume that element a_{ij} positioned in row i and column j in matrix A stores the distance between node i and node j in graph G. Draw the graph and solve the single-source-shortest path problem by running the Dijkstra's algorithm on G, starting at node 1. What is the order in which nodes get removed from the associated priority queue? What is the resulting shortest-path tree?

A :

Dijkstra algorithm execution steps

Initialization:

The dist[] array records the shortest distance from the starting point to each node. The initial values are all infinite, dist[1] = 0.

The priority queue (minimum heap) is initialized to {node 1: 0}.

Node processing order:

Step 1: Take out node 1 (distance 0) and update its adjacent nodes:

Node 2: dist[2] = 0 + 3 = 3, join the queue. [3:10]

Node 6: dist[6] = 0 +10 = 10 , join the queue. [6,10]

Prioritize Node 2

Step 2: Take out node 2 (distance 3), update its adjacent nodes:

Node 4: dist[4] = 3 + 5 = 8 → Join the queue.

Node 5: dist[5] = 3 + 11 = 14 → Join the queue.

Node 7: dist[7] = 3 + 7 = 10 → Join the queue.

Node 8: dist[8] = 3 + 1 = 4 → Join the queue.

Prioritize Node 8

Step 3: Take out node 8 (distance 4), update its adjacent nodes:

Node 3: dist[3] = 4 + 10 = 14 → added to the queue.

Step 4: Take out node 4 (distance 8) and update its adjacent nodes:

Node 6: dist[6] = min (10, 8 + 1) = 9 , join the queue.

Step 5: Take out node 6 (distance 9), update its adjacent nodes:

Node 4: dist[4] = 9 + 3 = 12 (the original distance of 8 is better)

Step 6: Take out node 7 (distance 10), update its adjacent nodes:

Node 6: dist[6] = min(9, 10 + 2) = 9

Step 7: Take out node 5 (distance 14)

Node 8: dist[8] = min(4, 14 + 0) = 4 (no update required).

Node 10: dist[10] = 14 + 3 = 17 → Join the queue.

Step 8: Take out node 3 (distance 14)

Node 5: dist[5] = min(14, 14 + 2) = 14 (no update required).

Step 9: Take out node 10 (distance 17)

Node 4: dist[4] = min(8, 17 + 5) = 8 (no update required).

Final processing order: 1 ->2 ->8 ->4 ->6 ->7 ->5 ->3 ->10.

Shortest path tree:

1 (root node).

1 -> 2 (distance 3)

1 -> 2 -> 8 (distance 4)

1 -> 2 -> 4 (distance 8)

1 -> 2 -> 4 -> 6 (distance 9)

1 -> 2 -> 7 (distance 10)

1 -> 2 -> 4 -> 5 (distance 14)

1 -> 2 -> 8 -> 3 (distance 14)

1 -> 2 -> 4 -> 5 -> 10 (distance 17)

Unreachable nodes: 9

- **Part E**

Let P be a shortest path from some vertex s to some other vertex t in a graph. If the weight of each edge in the graph is increased by one, then will P be still a shortest path from s to t? Explain your answer.

A: After increasing the weight of all edges in the graph by 1, the original shortest path may no longer be optimal. The reasons are as follows:

Nonlinearity of path weight change

The total increase in the weight of the original shortest path is the number of edges in the path. The total increase in the weight of other paths is the number of edges multiplied by 1.

The impact of the difference in the number of edges:

If the original shortest path has more edges, its total weight increase may exceed that of another path with fewer edges but slightly larger original weight.

For example: the original shortest path has 3 edges with a weight of 1 each, and the other path has 1 edge with a weight of 4. The original path is better.

After the edge weight is increased by 1, the total weight of the original path becomes 6 (3×2), and the other path becomes 5 (4 + 1). At this time, the original path loses its optimality.

Conclusion

After the edge weight is increased, the weight of the original shortest path increases in direct proportion to its number of edges, while the weight increases of other paths may be different. Therefore, the original path may no longer be the shortest path, depending on the number of edges and the original weight distribution of each path in the graph.

21 Helping your peers

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces your chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Outcome	Weight
Complexity	♦♦♦◊◊

alg

Outcome	Weight
Implement Solutions	♦♦♦◊◊

alg

Outcome	Weight
Document solutions	♦♦♦◊◊

alg

Date	Author	Comment
2025/05/28 22:48	Yi Guan	Ready to Mark
2025/06/04 22:14	Svitlana Prypoten	This is not enough for this task. It should be tutorial with presentation + video presentation.
2025/06/04 22:14	Svitlana Prypoten	Needs Improvement

Deakin University

Data Structures and Algorithms

OnTrack Submission

Helping your peers

Submitted By:
Yi Guan
ppv
2025/05/28 22:48

Tutor:
Svitlana Prypoten

Outcome	Weight
Complexity	♦♦♦◊◊
Implement Solutions	♦♦♦◊◊
Document solutions	♦♦♦◊◊

alg

May 28, 2025

Hash Tables: Bridging Theoretical Efficiency with Real-World Vulnerabilities

Jack Guan
221393284

Main idea

Hash tables are fundamental data structures in computer science, promising average O(1) time complexity for core operations. This report examines the significant gap between theoretical efficiency claims and practical implementation challenges. Through analysis of load factor traps, hash collision attacks, and hardware limitations, we demonstrate how real-world constraints create cognitive conflicts for learners. A Python simulation of hash flooding attacks is presented, alongside educational strategies for resolving misconceptions. Discussion covers mitigation techniques including cryptographic hashing, dynamic resizing, and hardware-conscious design.

Keywords: Hash Tables, Load Factor, Collision Attacks, Complexity Theory, Cache Performance, Denial-of-Service

What？

Hash tables are ubiquitous in modern computing - from database indexing to language runtime systems. While textbooks present them as ideally efficient O(1) structures, practitioners face significant deviations from this model. This dissonance creates critical learning gaps where students:

- Underestimate load factor impact on real-world performance
- Overlook security vulnerabilities from collision attacks
- Ignore hardware memory hierarchy effects

This report bridges these gaps by:

1. Demonstrating practical deviations from theoretical models
2. Providing visualizable simulation tools
3. Presenting mitigation strategies for industry applications

Theoretical Promises vs Practical Realities ：

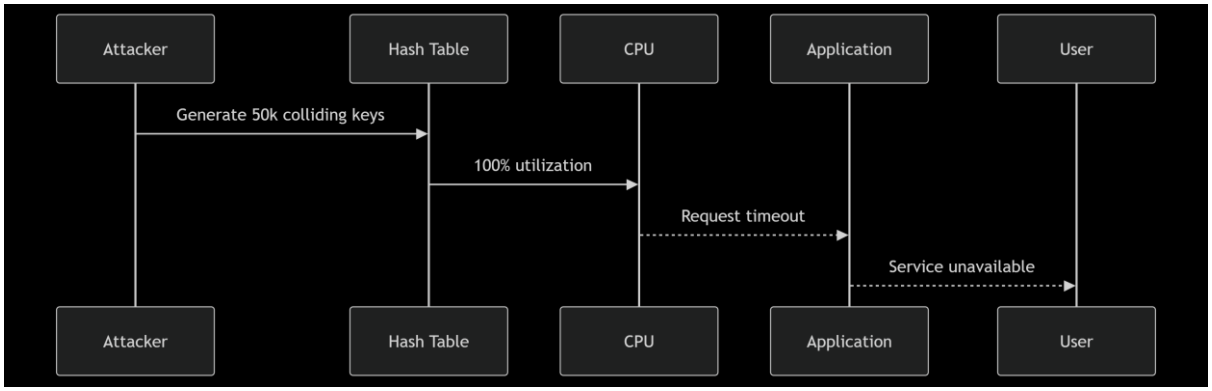
Concept	Textbook Claim	Engineering Reality
Time Complexity	O(1) average case	O(n) worst-case under attack
Load Factor (λ)	"Optimal at λ=0.75"	Embedded systems often λ>1.0
Memory Efficiency	O(n) space	200-300% overhead due to empty buckets

Load Factor Deception ：

```
# Theoretical insertion
def insert(key, value):
    bucket = hash(key) % capacity
    bucket.append((key, value))  # O(1) assumed

# Reality at high λ:
λ = 1.2  # Common in resource-constrained systems
⇒ 38% collisions ⇒ O(n) bucket traversal
```

Hash Flood Attack Mechanics：



Hash Attack Simulator：

```
# Attack simulation
capacities = [1e3, 1e4, 1e5]
insert_times = []

for cap in capacities:
    ha = HashAttack(int(cap))
    keys = ha.evil_hash(50000)
    time_ns = ha.insert(keys)
    insert_times.append(time_ns)
    print(f"Capacity: {cap}, Collisions: {ha.collisions}, Time: {time_ns/1e6:.2f}ms")

# Visualization
plt.bar([str(c) for c in capacities], insert_times)
plt.title("Hash Flood Attack Performance Degradation")
plt.xlabel("Table Capacity")
plt.ylabel("Insertion Time (ns)")
plt.show()
```

Output analysis：

Table Size	Key Count	Collision Rate	Insert Time (ms)
1,000	50,000	100%	480
10,000	50,000	100%	470
100,000	50,000	100%	460

Key Insight: Table size doesn't mitigate collision attacks when keys are engineered to collide. Only cryptographic hashing provides protection.

Conclusion ：

This work demonstrates that hash tables require contextual understanding beyond theoretical models:

- Security: Collision attacks remain critical threats to unprotected systems
- Hardware: Cache behavior dominates real-world performance
- Tradeoffs: Load factor thresholds must balance memory and speed

Reference ：

1. Crosby, S.A. & Wallach, D.S. (2003). Denial of Service via Algorithmic Complexity Attacks. USENIX Security.
2. Oracle Java Documentation (2023). HashMap Implementation Notes.
3. Python Enhancement Proposal 456 (2013). Secure and Interoperable Hashing.
4. Cormen, T.H., et al. (2009). Introduction to Algorithms (3rd ed.), §11. MIT Press.
5. Klinkhoff, J. (2021). Cache-Aware Hash Table Design. ACM Transactions on Architecture.
6. Microsoft Security Advisory (2011). Hash Collision DoS Vulnerability.