SIT221

Data Structures and Algorithms

Learning Summary Report

Neb Miletic
218489118

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit. Please tick (✓) the relevant boxes.

	Pass (D)	Credit (C)	Distinction (B)	HD (A)	Very High HD (A⁺)
Self-Assessment			✓		

Self-Assessment Statement

	Included
Learning Summary Report	✓
All six (6) Pass tasks are Complete *	✓

Minimum Pass Checklist

	Included
At least three (3) Credit Tasks are Complete *	✓

Minimum Credit Checklist (in addition to Pass Checklist)

	Included
All five (5) credit tasks are Complete *	✓
At least two (2) Distinction tasks are Complete (This can include HD tasks marked as only achieving a D) *	✓

Minimum Distinction Checklist (in addition to Credit requirements)

	Included
All three (3) Distinction tasks are Complete (This can include HD tasks marked as only achieving a D) *	
At least one (1) HD task is Complete (marked as achieving a HD)	

Minimum High Distinction Checklist (in addition to Distinction requirements)

	Included
Both High Distinction tasks are Complete (marked as achieving a HD)	
At least one (1) HD task is Complete (marked as three (3) stars)	

Minimum Very High HD Checklist (in addition to High Distinction requirements)

* You can include higher grade tasks in this count but cannot then use those same tasks when counting higher grade tasks. Eg each task can only be counted once.

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 or automated system.

Signature: **Neb Miletic**

Portfolio Overview

This portfolio includes work that demonstrates that I have achieve all Unit Learning Outcomes for SIT221 Data Structures and Algorithms to a **Distinction** level.

While I missed one Distinction task (Helping your peers), I think there is elements to consider for Distinction grade. I have done all Pass and Credit tasks, and besides that in my portfolio I think I was able to show that I can build and implement more complex solutions requiring a range of data structures and algorithms to solve more challenging real-world problems.

Reflection

The most important things I learnt:

Main data structures and algorithms that we are expected to know (and beyond that) were very well covered, with lecturers and colleagues who were keen to help and support me on this journey.

The things that helped me most were:

Well structured lectures and Maksym and Richard who were approachable and really knowledgeable about the topics we covered

I found the following topics particularly challenging:

Time and space complexity

I found the following topics particularly interesting:

P and NP problems, graph algorithms etc

I feel I learnt these topics, concepts, and/or tools really well:

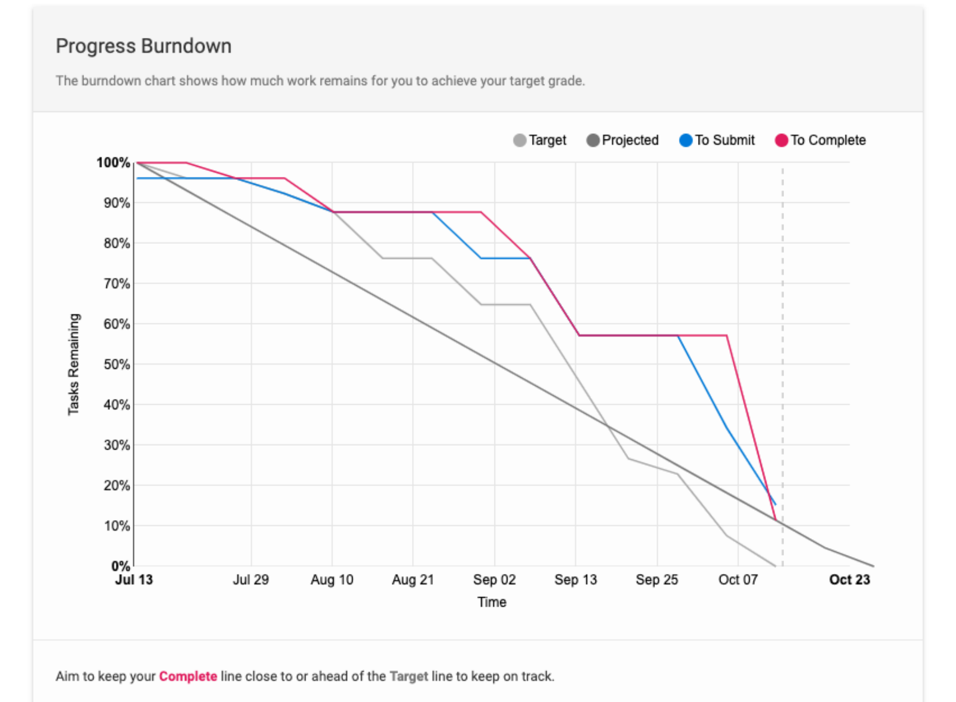
C# with Visual Studio, most of basic data structures and algorithms (vectors, linked lists, search algorithms, etc.)

I still need to work on the following areas:

More complex algorithms such Dijkstra, A*, dynamic computing etc

My progress in this unit was …:

While this graph shows that I did everything last moment, this was not the case. I needed help with some of the concepts , and due to unforeseen circumstances (everyone being on the cloud) it was hard to get help on time



This unit will help me in the future:

I consider this unit as the most important in this course, and it will definitely help me to understand and tackle a lot of new concepts in IT science and industry.

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

I would dedicate even more time to learn the material and I would try to research even deeper the topics that was a bit hard to grasp.

Other…:

Generally, I'm very happy with the outcome of this unit. The only regret is that I was at the level to get High Distinction, I would be very proud of that.

Deakin University

Data Structures and Algorithms

Nebojsa Miletic

Portfolio Submission

Submitted By:
Nebojsa Miletic
mileticn

Tutor:
Richard Dazeley

October 13, 2020

Produced by Doubtfire

Contents

2 Overall Task Status

Task	Status	Times Assessed
Initial Task	Complete	1
Vector: A simple list-like collection class	Complete	2
Helping your peers	Not Started	
Algorithm complexity (pass)	Complete	5
Algorithm complexity (credit)	Complete	1
Basic Sorting	Complete	1
Recursive Sorting	Complete	1
Iteration and Search	Complete	3
Smart Evacuation	Not Started	
Doubly Linked List	Complete	3
Problem solving: Search and Stack	Complete	2
Programming - Problem Solving	Complete	1
AVL-Trees	Complete	2
Programming - Heap	Complete	2
Programming - Coin Combinations	Not Started	
Problem Solving: Graphs	Complete	2
Quiz - Final two weeks	Complete	1

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
Algorithm complexity (credit)	♦◊◊◊◊	Complete	1
Problem solving: Search and Stack	♦◊◊◊◊	Complete	2
Programming - Problem Solving	♦◊◊◊◊	Complete	1
AVL-Trees	♦◊◊◊◊	Complete	2
Programming - Heap	♦◊◊◊◊	Complete	2
Problem Solving: Graphs	♦◊◊◊◊	Complete	2

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
Vector: A simple list-like collection class	♦♦♦◊◊	Complete	2
Basic Sorting	♦◊◊◊◊	Complete	1
Iteration and Search	♦◊◊◊◊	Complete	3
Doubly Linked List	♦◊◊◊◊	Complete	3

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
Initial Task	♦◊◊◊◊	Complete	1
Algorithm complexity (pass)	♦◊◊◊◊	Complete	5
Recursive Sorting	♦◊◊◊◊	Complete	1
Iteration and Search	♦◊◊◊◊	Complete	3
Quiz - Final two weeks	♦◊◊◊◊	Complete	1

4 Initial Task

Initial task to detail how the assessment will be run this trimester

Outcome	Weight
Document solutions	♦◊◊◊◊

task

Date	Author	Comment
2020/07/07 08:52	Nebojsa Miletic	Ready to Mark
2020/07/23 17:25	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Initial Task

Submitted By:
Nebojsa Miletic
mileticn
2020/07/07 08:52

Tutor:
Richard Dazeley

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

task

July 7, 2020

Practical Task 0.1

(Pre-Pass Task)

Submission deadline: 11:59 pm Sunday, July 19
Discussion deadline: 11:59 pm Sunday, July 26

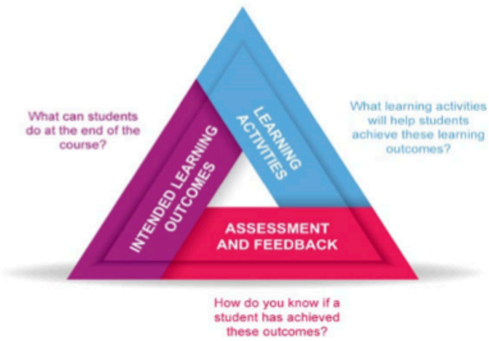
Task Objective

This task is to ensure 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 complete the form included at the end, submit the form to OnTrack and have a short introductory discussion with your tutor via OnTracks Intelligence Discussion Tool.

Background

SIT221 Data Structures and Algorithms has a reputation for being a very challenging and enjoyable unit. To help students tailor this unit to achieve the outcomes they are after the unit will use 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.



http://www.ntu.edu.sg/tlpd/tlr/DesigningYourCourse/OBTL/Pages/ConstructiveAlignment.aspx

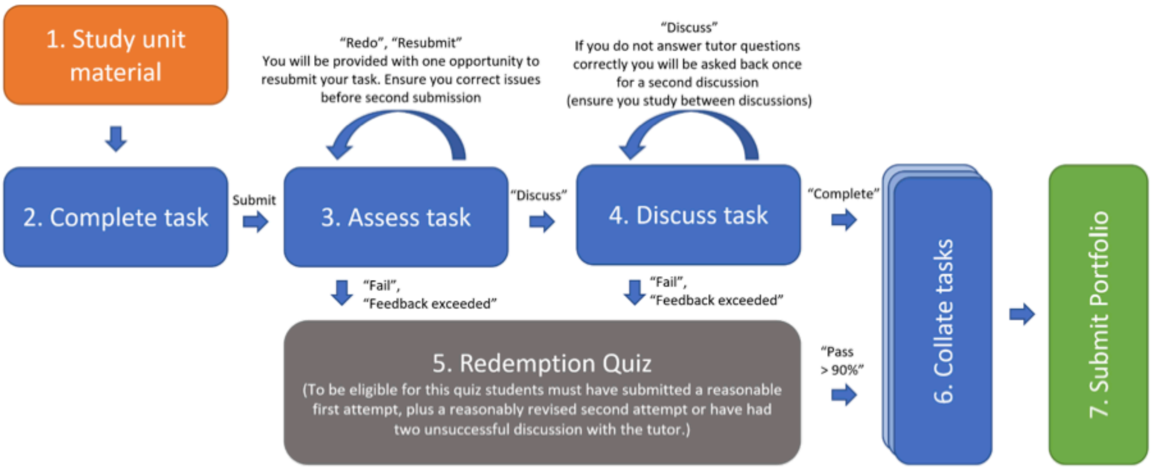
In this unit this Constructive Alignment is achieved via OnTrack. OnTrack is an in house developed online software platform that facilitates this formative learning approach via a task-oriented portfolio assessment. The strategy is to organise learning activities across multiple tasks structured around grade outcomes. You the student select the grade you are aiming to achieve – this decision may 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 selection allows you to scaffold your own learning to achieve greater depth than you previously may have thought was possible.

- Pass tasks: cover the defined learning tasks of the unit – they show a basic level of knowledge and skills required to progress in your degree.
- Credit tasks: develop your skills beyond the minimum required and show a solid level of knowledge and skills in the topics of the unit.
- Distinction tasks: create advanced skills that show that you can research and apply theory to solve complex problems.
- High distinction tasks: research approaches that go well beyond the core content of the unit showing you are highly capable of meeting employer expectations in this domain.

Your final grade is based on the number of tasks that you complete at each task level, see below. Students aiming for higher grades will need to do 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.

- Pass – must do all 6 pass tasks (in addition to this pre-pass task).
- Credit – must do all pass task and more than half (3) of the (5) credit tasks.
- Distinction – must do all pass and credit tasks and more than half (2) of the (3) distinctions tasks.
- High distinction – must do all pass, credit and distinction tasks, plus at least one high distinction task.

Finally, there will also be quizzes in week 11/12 providing you an opportunity to improve your final grade. Note: if you do not complete required tasks at a level then we will accept higher tasks as a replacement.



Assessment Process

Knowledge and skills in this unit continuously build on those learnt the week before. Therefore, if you fall behind it becomes impossible to understand subsequent content. Therefore, tasks are spread out throughout the semester and need to be done in sequence and on time.

To complete a task, students follow the formative learning process illustrated in the above process diagram. This is designed to ensure your educators can provide a reasonable level of support to all students. The following describe the steps to be followed during this unit.

1. **Study material:** Ensure you attend/watch lectures and read any associated reading material.
2. **Complete task:** during your lab time your tutor will provide some general advice and answer questions to clarify a task's requirements. They will not provide solutions – partial or otherwise. You may also seek peer support by posting questions on the SIT221 forum in student sync. Once you believe you have the correct solution you can submit your solution via OnTrack (https://ontrack.deakin.edu.au/). When you submit please indicate to your tutor if you would prefer to do your discussion via Microsoft Teams during your lab time or via OnTrack's Intelligent Discussion facility – see step 4 for details of these two discussion approaches.
3. **Assess task:** Your tutor will assess 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, they will mark the task as either "redo" or "resubmit". You will now get one (1) 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 one resubmission – if on the second submission you are very close then the tutor may decide to give you a day to make minor changes and submit again, but this is up to their discretion and should not be expected. If the tutor does not accept your submission, then it will be marked as "Fail" or "Feedback exceeded". If it is marked as "Feedback exceeded" then go to step 5. Plagiarised tasks or purchased solutions will be marked as "Fail" and you will not be eligible to go to step 5 and cannot pass this task – you will need to replace this task with a more advanced task.
4. **Discuss (Demonstrate) task:** If your solution is correct in step 3 then you must also illustrate to your tutor that you understood the material in an interview with the tutor. Before the discussion deadline (see unit guide) you must either:
    - Attend the tutorial via Microsoft Teams and have a one-on-one meeting with your tutor. To do this you should ensure you have a working camera and microphone (Teams has a mobile app available if you do not have a webcam). When you attend the lab, class announce to the tutor that you are ready for an interview and wait for them to get to you.
    - Use the intelligent discussion facility in OnTrack. If you have indicated this is your preferred approach, then your tutor will record up to three audio question. 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. Please see this link for more details about this OnTrack facility.

If the tutor is satisfied with your answers then they will mark your task as "Complete". If they do not accept your responses show the required level of understanding you will be asked to discuss again. The tutor is only expected to give you two opportunities to discuss your task. If the tutor does not accept your submission then it will be marked as "Feedback exceeded", see step 5.

5. **Redemption quiz:** If your task is marked as "Feedback exceeded" then you can still receive a pass by sitting the redemption quiz for that pass task. This facility is only provided to pass tasks. Your tutor will indicate with a comment in the original tasks chat window if you are eligible for the redemption quiz. To be eligible you need to have made a reasonable attempt to do the task (copied tasks will not be accepted as a reasonable attempt). Each time you were allowed a resubmit you showed a reasonable attempt to fix your submission. Simply resubmitting the original will not be accepted as a reasonable attempt. You may sit each task's redemption quiz up to three times. You must get 90% correct on the quiz for it to count as a replacement to the pass task. Note the questions are drawn randomly from a pool of questions and so will change each time you attempt the quiz. When doing the quiz, you are also required to record your time doing the quiz. This means you should use Zoom (or another application with similar functionality) and set it to record your whole of screen which will include your browser while you do the quiz. This recording should also show a mini window showing your face while doing the quiz. You will provide a link to the zoom recording in your chat session of the original pass task for the tutor to validate. If you gain 90% or more and submit the video and the tutor or unit chair is satisfied with these submissions, then the original pass task will be marked as complete. Should you be unsuccessful in passing the quiz on all three occasions you should do a more advanced task and have that used to replace the missed pass task.
6. **Collate all tasks:** At the end of week 11 you prepare your final portfolio for submission. This will involve filling in a cover sheet and 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 argue for the grade you believe you should receive.
7. **Portfolio:** formally this is your actual assessment task for the unit. Ensure this is submitted with the required form by the due date.

Please note: The redemption quiz provides you with the opportunity to pass a task even if the tutor is not satisfied by your submission or interview. Any attempt to intimidate or bully a tutor during the assessment or interview of a task will not be tolerated.

Task Details

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: 218489118

Name: Neb Miletic

☐ 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:

Date: 6/07/2020

Eg a passport style photo

5 Vector: A simple list-like collection class

Note that we will not check 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 the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work through the unit.

Outcome	Weight
Implement Solutions	♦♦♦◊◊

task

Date	Author	Comment
2020/07/14 09:14	Nebojsa Miletic	Need Help
2020/07/15 19:26	Nebojsa Miletic	Working On It
2020/07/15 19:29	Nebojsa Miletic	Need Help
2020/07/17 10:15	Nebojsa Miletic	Ready to Mark
2020/07/23 17:26	Maksym Slavnenko	Please try to run Remove(2000) and see whether your code fails
2020/07/23 17:26	Maksym Slavnenko	Fix and Resubmit
2020/07/23 17:27	Maksym Slavnenko	on top of that once you do a ‘Clear()‘ method, your Vector becomes unusable, you can’t do anything with it
2020/07/23 17:27	Maksym Slavnenko	don’t set your previous array to null there, just create a new empty array and reassign the private ‘data‘ field to it
2020/07/23 17:43	Maksym Slavnenko	What is that ‘T‘ in ‘Vector<T>‘?
2020/07/23 17:43	Maksym Slavnenko	What is the difference between a Vector and an array?
2020/07/23 17:43	Maksym Slavnenko	what is that: ’private T[] data‘?
2020/07/23 17:47	Maksym Slavnenko	You have passed an interview. You just need to resubmit the correct code to make this task ‘Complete‘.
2020/07/23 17:48	Maksym Slavnenko	‘public int Capacity => data.Length;‘
2020/07/30 18:10	Nebojsa Miletic	Ready to Mark
2020/08/07 13:09	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Vector: A simple list-like collection class

Submitted By:
Nebojsa Miletic
mileticn
2020/07/30 18:10

Tutor:
Maksym Slavnenko

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

task

July 30, 2020

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

namespace Vector
{
    public class Vector<T>
    {
        // This constant determines the default number of elements in a newly created vector.
        // It is also used to extended the capacity of the existing vector
        private const int DEFAULT_CAPACITY = 50;

        // This array represents the internal data structure wrapped by the vector class.
        // In fact, all the elements are to be stored in this private  array.
        // You will just write extra functionality (methods) to make the work with the array more convenient for the user.
        private T[] data;

        // This property represents the number of elements in the vector
        public Vector(int count, int capacity)
        {
            this.Count = count;
            //this.Capacity = capacity;

        }


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

        // This property represents the maximum number of elements (capacity) in the vector
        public int Capacity => data.Length;

        // This is an overloaded constructor
        public Vector(int capacity)
        {
            data = new T[capacity];
        }

        // This is the implementation of the default constructor
        public Vector() : this(DEFAULT_CAPACITY) { }

        // 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.
        // For example, introducing the following indexer you may address an element of the vector as vector[i] or vector[0] or ...
        public T this[int index]
        {
            get
            {
```

```
                if (index >= Count || index < 0) throw new IndexOutOfRangeException();
                return data[index];
            }
            set
            {
                if (index >= Count || index < 0) throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }

        // This private method allows extension of the existing capacity of the vector by another 'extraCapacity' elements.
        // The new capacity is equal to the existing one plus 'extraCapacity'.
        // It copies the elements of 'data' (the existing array) to 'newData' (the new array), and then makes data pointing to 'newData'.
        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;
        }

        // This method adds a new element to the existing array.
        // If the internal array is out of capacity, its capacity is first extended to fit the new element.
        public void Add(T element)
        {
            if (Count == data.Length) ExtendData(DEFAULT_CAPACITY);
            data[Count++] = element;
        }

        // This method searches for the specified object and returns the zerobased index of the first occurrence within the entire data structure.
        // This method performs a linear search; therefore, this method is an O(n) runtime complexity operation.
        // If occurrence is not found, then the method returns 1.
        // Note that Equals is the proper method to compare two objects for equality, you must not use operator '=' for this purpose.
        public int IndexOf(T element)
        {
            for (var i = 0; i < Count; i++)
            {
                if (data[i].Equals(element)) return i;
            }
            return -1;
        }

        // ********************************************************************************************
        //   Your task is to implement all the remaining methods.
        // Read the instruction carefully, study the code examples from above as they should help you to write the rest of the code.
```

```
        public void Insert(int index, T element)
        {
            if (Count == Capacity)
                {
                    ExtendData(Capacity);
                    for (int i = Count - 1; i >= index; i--)
                    {
                        data[i + 1] = data[i];
                    }
                    data[index] = element;
                    Count++;


            }

                else if (index == Count)
                {
                    Add(element);
            }

            else if (index < 0 || index > Count)
                {
                    throw new IndexOutOfRangeException();
                }
                else
                {
                    for (int i = Count-1; i >= index; i--)
                        {
                         data[i+1] = data[i];
                        }
                        data[index] = element;
                        Count++;
                }


        }

        public void Clear()
        {

                data = new T[Capacity];
                Count = 0;
        }

        public bool Contains(T element)
        {

                return IndexOf(element) != -1;

        }

        public bool Remove(T element)
        {
```

```
                if(IndexOf(element) == -1)

                  return false;

                else

                  RemoveAt(IndexOf(element));
                  return true;




        }
        public void RemoveAt(int index)
        {

                if (index < Count)
                {
                    for(int i = index;i < Count-1;i++)
                    {
                      data[i] = data[i+1];
                    }
                    Count--;
                }
                else if(index < 0 || index >= Count)
                {
                    throw new IndexOutOfRangeException();
                }

        }

        public override string ToString()
        {

             var sb = new StringBuilder();

             for (int i = 0; i < Count; i++)
                 sb.Append(String.Format("{0},",data[i]));

            string sd = String.Join(",",sb);
            return "[" + sd.TrimEnd(',') + "]";


        }

    }
}
```

6 Algorithm complexity (pass)

New Description

Outcome	Weight
Document solutions	♦◇◇◇◇

task

Date	Author	Comment
2020/07/23 14:06	Nebojsa Miletic	Ready to Mark
2020/07/23 17:50	Maksym Slavnenko	rand() returns a float number between [0,1]
2020/07/23 17:51	Maksym Slavnenko	q 1.2 incorrect for the best case
2020/07/23 17:53	Maksym Slavnenko	‘undefined3 +undefined2 +106undefined=undefined(undefined4)Thisistrue‘ - you answer, incorrect
2020/07/23 17:54	Maksym Slavnenko	q 4.3 incorrect
2020/07/23 17:58	Maksym Slavnenko	Fix and Resubmit
2020/07/30 17:33	Maksym Slavnenko	some of your q.1 are incorrect. Please check them
2020/07/30 18:08	Nebojsa Miletic	Ready to Mark
2020/08/07 13:13	Maksym Slavnenko	‘3+p(p is 0 or 1 or 0.5)*(log(n)+log(n)+log(n)+1)‘
2020/08/07 13:14	Maksym Slavnenko	p = 0 : 3 + 0 = 3p = 1 : 3 + 1 + 3log(n) = 3log(n) + 4p = .5 : 3 + .5 + 1.5log(n) => 1.5log(n) + 3.5
2020/08/07 13:16	Maksym Slavnenko	image comment
2020/08/07 13:20	Maksym Slavnenko	What is big O?
2020/08/07 13:20	Maksym Slavnenko	What is the lower bound name?
2020/08/07 13:20	Maksym Slavnenko	What is big Theta?
2020/08/07 13:22	Maksym Slavnenko	n^3 * n^2 = n^5
2020/08/07 13:23	Maksym Slavnenko	q.4.3 correct answer, incorrect explanation
2020/08/07 13:23	Maksym Slavnenko	You passed the interview on this one, please fix your code and resubmit and I will mark it as ‘Complete‘
2020/08/07 13:25	Maksym Slavnenko	Fix and Resubmit
2020/09/07 20:29	Nebojsa Miletic	Ready to Mark
2020/09/07 20:29	Maksym Slavnenko	Time Exceeded
2020/09/10 17:44	Maksym Slavnenko	‘Thisisfalse,giventhatbigOinthiscaseisprobablyn^5,so average case is possible n^4‘
2020/09/10 17:44	Maksym Slavnenko	this is incorrect in 4.3
2020/09/10 17:45	Maksym Slavnenko	Fix and Resubmit
2020/09/15 19:04	Nebojsa Miletic	Ready to Mark
2020/09/15 19:04	Maksym Slavnenko	Time Exceeded
2020/09/24 16:21	Maksym Slavnenko	q 4.3 is incorrect
2020/09/24 16:21	Maksym Slavnenko	Fix and Resubmit
2020/10/01 13:14	Nebojsa Miletic	4.3: n^3 + n^2 + 10^6n = Big Theta(n^4) This is false. The left and right side have a different quadratic speed (n^3 and n^4) so big Theta can not be n^4.
2020/10/02 11:42	Nebojsa Miletic	Ready to Mark
2020/10/02 11:42	Maksym Slavnenko	Time Exceeded
2020/10/06 12:58	Maksym Slav-	Neb, please fix your submission

Deakin University

Data Structures and Algorithms

OnTrack Submission

Algorithm complexity (pass)

Submitted By:
Nebojsa Miletic
mileticn
2020/10/02 11:42

Tutor:
Maksym Slavnenko

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

task

October 2, 2020

1. i.

int a = 0; *1= assigning the variable*
a += rand(); *1 = add the value*
if(a < 0.5) a += rand(); *2*
if(a < 1.0) a += rand(); *2*
if(a < 1.5) a += rand(); *2*
if(a < 2.0) a += rand(); *2*
if(a < 2.5) a += rand(); *2*
if(a < 3.0) a += rand(); *2*

1 + 1 + 2 + 2+ 2 + 2+ 2 + 2 = 14 operations.

What is the number of operations of the best, worst and average cases?

The worst case for this function is 14 operations. The best case would be 10. Average 12

Describe the best, worst and average case using Big-Θ notation.

In this case we can say that there are two constant values growing at the same rate: There exists an n0 and constants c1, c2 > 0 such that for all n > n0, $c1g(n) \leq |f(n)| \leq c2g(n)$. For worst case or O(n) = 14 or O(1) and best case Ω(g) = 10 or Ω(1) we can say that Θ(1) or average case is constant, because big O and big Omega are the same.

Describe each algorithm's overall performance using the tightest possible class in Big-O notation.

Because the worst case in this algorithm is 14, we can say that f(n) = O(g(n)) or that function n is growing no faster than g(n). Or because the Big O is constant, we say O (1).

Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.

The best case is 8, so f(n) = Ω(g(n)) therefore f(n) grows at least as fast as g(n). So Big- Ω(1).

Describe each algorithm's overall performance using Big-Θ notation.

f(n) = Θ(g(n)) therefore f(n) grows at same rate as g(n). So Θ(1).

Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.

14: O(1), Ω(1), Θ(1).

ii.

```
int count = 0;                  1(initialisation)
for (int i = 0; i < N; i++) {   1(initialization)+n+1(checks)operation

        int num = rand();            n(initialisation)

        if( num < 0.5 ){             n(checks)

                count += 1; }        n(add)

}                                    n(increment)
```

Worst case: T(n)= 1 + 1 + (n+1) + n + n + n + n = 5n + 3 operations

Average case: 1 + 1 + (n+1) + n + n/2 + n/2 + n = 4n + 3 operations

Best case: T(n) = 1 + 1 + (n+1) + n + n = 3n + 2 operations

What is the number of operations of the best, worst and average cases?

Number of worst cases is 5n + 3. The best case is 3n + 2, so if the N =< 0, that would mean that algorithm would stop after initialising variable count and variable i to 0 and comparing the N and i, so that are 3 operations. The average case is usually similar at 4n + 3.

Describe the best, worst and average case using Big-Θ notation

Worst case: 5n + 3 = O(n)
Best case: 3n + 2 = Ω(n)
Average case: 4n + 3 = Θ (n)

Describe each algorithm's overall performance using the tightest possible class in Big-O notation
f(n) = O(g(n)) or f(n) is growing no faster than g(n). So, 5n + 3 = O(n)

Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.

f(n) = Ω(g(n)) therefore f(n) grows at least as fast as g(n). In this case, 3n oreations are linear the best case, so we can say 3n + 3 = Ω(n).

Describe each algorithm's overall performance using Big-Θ notation.

f(n) = Θ(g(n)) therefore f(n) grows at same rate as g(n). *Θ(n)*

Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.

5n + 3 = O(n)

iii.

```
int count = 0;                  1(initialisation)
for (int i = 0; i < N; i++) {   1 + n+1

    if (unlucky)                n(check)
    {

       for (j = N; j > i; j--) {    n(1+n+1) (check)


          count = count + i + j; }    n(adding)

    }                           n(decrementing)

}                               n(incrementing)
```

What is the number of operations of the best, worst and average cases?

Worst case: 3n + 3 + n + n$(\frac{n+1}{2}$ + 1) + $(\frac{n(n+1)}{2})$ + $(\frac{n(n+1)}{2})$ = $\frac{3n^2}{2}$ + $\frac{3n}{2}$ + 5n + 3 = $\frac{3n^2}{2}$ + $\frac{13n}{2}$ + 3(if always unlucky)

Average case: 3n + 3 + n + $\frac{n}{2}(\frac{n+1}{2}$ + 1) + $(\frac{n(n+1)}{2})$ + $(\frac{n(n+1)}{2})$ = $\frac{23n + 5n^2}{4}$ + 3

Best case: 1 + 1 + (n+1) + n + n = 3n + 3 operations.

Describe the best, worst and average case using Big-Θ notation

Worst case $\frac{3n^2}{2}$ + $\frac{13n}{2}$ + 3 = O(n^2)

Best case: 3n+3 = Ω(n)

Big O and Big Omega are different so Big-Theta is not applicable. Therefore, average is similar as worst case: O(n^2)

Describe each algorithm's overall performance using the tightest possible class in Big-O notation

$\frac{3n^2}{2}$ + $\frac{13n}{2}$ + 3= O(n^2)

Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.

$3n+3 = \Omega(n)$

Describe each algorithm's overall performance using Big-Θ notation

f(n) = Θ(g(n)) therefore f(n) grows at same rate as g(n). We cannot say that in this case, so there is not a value for big Theta in this case.

Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.

$\frac{3n^2}{2} + \frac{13n}{2} + 3 = O(n^2), \Omega(n)$

iv.

```
int count = 0                     1 (initialisation)

int i = N;                        1 (initialisation)

if (unlucky)                      1 (check)
{

      while (i > 0) {              log(n)check)

            count += i;           log(n)(add)

             i /= 2; }             log(n)(divide N)

}
```

What is the number of operations of the best, worst and average cases?

Worst case: 1+1+1+log(n)+ log(n) + log(n) =3+3log(n)

Average case:
In this case, it seems that average case is the same as worst case, due to the nature of the algorithm. The only difference would be if clause (lucky or unlucky), which triggers the while loop.

Best case: 3 operations

Average case: log(n)

Describe the best, worst and average case using Big-Θ notation

Worst: 3+3log(n) = O(log(n))

Best: 3 = Ω(1)

Average: 3+3log(n) = O(log(n))

Describe each algorithm's overall performance using the tightest possible class in Big-O notation

3+3log(n) = O(log(n))

Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.

3 = Ω(1)

Describe each algorithm's overall performance using Big-Θ notation
f(n) = Θ(g(n)) therefore f(n) grows at same rate as g(n). We cannot say that in this case, so there is not a value for big Theta in this case, so we use big O notation.

Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.

3+3log(n) = O(log(n)), Ω(1)

v.

```
int count = 0;                          1(initialisation)

for (int i = 0; i < N; i++) {           1+n+1

    Int num = rand();                   n

     if( num < 0.5 )                    n
     {

        count += 1;}                    n

}                                       n(increment
int num = count;                        1
for (int j = 0; j < num; j++) {         1 + (n+1)

   count = count + j; }                 n

                                        n
```

What is the number of operations of the best, worst and average cases?

Worst case: 1+ 1+(n+1)+n+n+n+n+1+1+ (n+1)+n+n = 6+8n

Best case: 1+ 1+(n+1)+n+n

Average: we should consider that rand in this case would be 50:50 for every n half will be less than 0.5 and half more so, $\frac{n}{2}$ for count += 1. That affects second for loop, whose n would be $\frac{n}{2}$, so if we have 6 + 8n for worst case, average would be 6 + 7n.

Describe *the best, worst and average case using Big-Θ notation*

Worst case: 6+8n= O(n)
Best case: 3n + 3
Average: 6 + 7n *Θ* (n)

Describe each algorithm's overall performance using the tightest possible class in Big-O notation
6+8n = O(n)

Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.
3+3n = Ω(n)

Describe each algorithm's overall performance using Big-Θ notation

$6 + 7n = \Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

It is n because Big Omega function grows at the linear rate and big O at linear rate

Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.

6 + 8n = O(n)

vi.

```
for (int i = 0; i < N - 1; i++) {          1+ n+1

  for (int j = 0; j < N-i-1; j++) {        1 + n(n+1)
```

```
                if (a[j] > a[j+1])          n
                {
                    Swap(a[j], a[j + 1]);   n
                }

        }                                   n

                                            n

}
```

What is the number of operations of the best, worst and average cases?

Worst case: $1+ (n + 1) +1+ n(n+1) +n + n + n+n = 4+5n+ n^2$
Best case: $4 + 3n + n^2$
Average: $4+4n+ n^2$

Describe *the best, worst and average case using Big-Θ notation*

Worst case: $4+5n+ n^2 = O(n^2)$

Best case: $4 + 3n + n^2= \Omega(n^2)$

Average case: $4+4n+ n^2 = \Theta(n^2)$

Describe each algorithm's overall performance using the tightest possible class in Big-O notation
$4+5n+ n^2 = O(n^2)$

Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.
$4 + 3n + n^2= \Omega(n^2)$
Describe each algorithm's overall performance using Big-Θ notation

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
$4+4n+ n^2 = \Theta(n^2)$
Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.

$4+5n+ n^2 = O(n^2)$

2.Arguably, the most commonly used asymptotic notation used is frequently Big-O. Discuss why this is so commonly the case.

The reason why the big O is the most common asymptotic notation is because usually for algorithm the most important factor is how it works in worst case scenarios, so that is one of the ways how to measure the effectiveness of the function.

3. Is it true that $\theta(n^3)$ algorithm always takes longer to run than an $\theta(\log n)$ algorithm? Explain your answer.

It wont' be correct to say that. It depends of the size of the list that algorithms work on. It would be possible that $\theta(n^3)$ is quicker, it all depends of the constant in front of n. (29 log n can be slower than 0.1 n^3

1. Answer whether the following statements are right or wrong and explain your answers. – $2n^2 + 6^{13}n = O\ (n^2)$

   This statement is true. The quadratic function is the slowest in this case., so we could say that is the worst case.

   – $n\ log\ n = O(n)$ This statement is not true. n log n is slower in growth than linear growth.
   – $n^3\ + n^2\ + 10^6 n = \theta(n^4)$ This is false. The left and right side have a different quadratic

   speed (n^3 and n^4) so big Theta can not be $n^{4.}$

   – $n\ log\ n = \Omega(n)$ False . n log n is slower than n

7 Algorithm complexity (credit)

New Description

Outcome	Weight
Complexity	♦◊◊◊◊

task

Date	Author	Comment
2020/08/05 10:42	Nebojsa Miletic	need time to finish the task
2020/08/07 13:34	Maksym Slavnenko	‘n*log(9n) / 9n*log(n)‘
2020/08/07 13:34	Maksym Slavnenko	‘log(a*b) = log(a) + log(b)‘
2020/08/07 13:36	Maksym Slavnenko	‘(log(9) + log(n)) / 9*log(n)‘
2020/08/07 13:36	Maksym Slavnenko	‘(a+b)/c = a/c + b/c‘
2020/08/07 13:37	Maksym Slavnenko	‘(log(9) / 9*log(n)) + (log(n)/9*log(n))‘
2020/08/07 13:39	Maksym Slavnenko	1/2 > 1/4 > 1/8
2020/08/07 13:39	Maksym Slavnenko	0 + 1/9 = 1/9
2020/08/07 13:40	Maksym Slavnenko	‘n*log(9n) / 9n*log(n)‘
2020/08/07 13:44	Maksym Slavnenko	(1/n) / (9/n) = 1/n * n/9 = (1*n)/ (n*9) = 1/9
2020/09/08 18:46	Nebojsa Miletic	Ready to Mark
2020/09/08 18:46	Maksym Slavnenko	Time Exceeded
2020/09/10 17:45	Maksym Slavnenko	‘lim(F1/F2) = c‘ => what does this imply? ‘F1 = ?(F2) : O,Θ,Ω‘
2020/09/10 17:47	Maksym Slavnenko	q2 QuickSort worst case is nˆ2
2020/09/10 17:48	Maksym Slavnenko	You stated that it’s MergeSort during the interview
2020/09/10 17:48	Maksym Slavnenko	Please fix it in the portfolio
2020/09/10 17:48	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Algorithm complexity (credit)

Submitted By:
Nebojsa Miletic
mileticn
2020/09/08 18:46

Tutor:
Maksym Slavnenko

Outcome	Weight
Complexity	◆◇◇◇◇

task

September 8, 2020

1.

Task 1.1P asked you to develop / provided you with a number of the Vector<T> class's methods (and properties), such as *Count*, *Capacity*, *Add*, *IndexOf*, *Insert*, *Clear*, *Contains*, *Remove*, and *RemoveAt*. What is the algorithmic complexity of each of these operations? Does your implementation match the complexity of the corresponding operations offered by Microsoft .Net Framework for its List<T> collection?

Big O for Count is constant or O(1),same as on Windows.Net.Framework site. For the Capacity: Retrieving the value of this property is an O(1) operation; setting the property is an O(n) operation, where n is the new capacity.

For Add method, the one in Task 1.1 has the complexity of O(1) if the index is less than count, same as the Windows one, where the big O is constant, too.

IndexOf function is linear search method, so it is O(n).

Insert function is overridden. In the Task 1.1 the complexity is the same as original, with the big O(n).

Clear method is overridden as well. The function in the task makes new array with the same capacity as old, and sets the counter to 0. The Microsoft version has a time complexity of O(n), while the version in the task has constant complexity – O(1).

Remove function in Microsoft version implements linear search with ObjectEquals as a default comparer. The method is O(n) operation, with the implemented version same as original – O(n).

RemoveAt function is originally O(n) time complexity, where n is (Count – index). Same is with the implemented version, it has big O of n (linear) time complexity.

2.

The function which satisfies this condition would be QuickSort algorithm. This algorithm relies on partition called pivot to divide an array into two parts. So, if pivot is let's say the highest value on the list, that is consider as a worst case and it would run quadratic time, because it would have to go through all array and swap the numbers. On the other hand, the best case is other way around. On sorted list, and with a wisely chosen pivot point, the algorithm would run n times. The average case would be θ(nlogn) time, so in between of those two cases.

3.

a)

$\mathrm{f(n)} = n^{\frac{1}{2}}$ and $\mathrm{g(n)} = \log \mathrm{n}$ $\quad \lim_{n \to \infty} \frac{n^{\frac{1}{2}}}{\log \mathrm{n}} = \infty.$ So, f grows faster than g and that is $\mathrm{f} = \Omega(\mathrm{g})$.

b)

f(n) = 1500 and g(n) = 2 $\lim_{n \to \infty} \frac{1500}{2} = 750$ so $f = \theta(g)$

c)

f(n) = 800 * 2^n and g(n) = 3^n $\lim_{n \to \infty} \frac{800 * 2^n}{3^n} = 0$ so $f = O(g)$

d)

f(n) = 4^{n+13} and g(n) = 2^{2n+2} $\lim_{n \to \infty} \frac{4^{n+13}}{2^{2n+2}} = 16777216$ so $f = \theta(g)$

e)

f(n) = 9n * log n and g(n) = n*log9n $\lim_{n \to \infty} \frac{9n * \log(n)}{n * \log(9n)} = 9$ $f = \theta(g)$

f)

f(n) = n! and g(n) = (n+1)! $\lim_{n \to \infty} \frac{n!}{(n+1)!} = 0$ so $f = O(g)$

8 Basic Sorting

Implement basic sorting routines

Outcome	Weight
Implement Solutions	♦◊◊◊◊

t

Date	Author	Comment
2020/08/06 18:03	Nebojsa Miletic	Ready to Mark
2020/08/07 13:25	Maksym Slavnenko	‘IComparer<T>‘ and the ‘IComparable<T>‘ . What is the difference between those 2? Why is ‘IComparer<T>‘ needed at all if there is already ‘IComparable<T>‘?What are the complexities of the sorting algorithms that you used? Is any of those 3 algorithms better than other in certain cases? If yes, then in what case is it better?
2020/08/07 13:25	Maksym Slavnenko	What is the ‘worst‘-time complexity for those?
2020/08/07 13:25	Maksym Slavnenko	what are the best case complexities ?
2020/08/07 13:30	Maksym Slavnenko	‘bool isSwapped‘
2020/08/07 13:31	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Basic Sorting

Submitted By:
Nebojsa Miletic
mileticn
2020/08/06 18:03

Tutor:
Maksym Slavnenko

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

t

August 6, 2020

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

namespace Vector
{
    public class Vector<T> where T : IComparable<T>
    {

        // This constant determines the default number of elements in a newly created vector.
        // It is also used to extended the capacity of the existing vector
        private const int DEFAULT_CAPACITY = 10;

        // This array represents the internal data structure wrapped by the vector class.
        // In fact, all the elements are to be stored in this private  array.
        // You will just write extra functionality (methods) to make the work with the array more convenient for the user.
        private T[] data;

        // This property represents the number of elements in the vector
        public int Count { get; private set; } = 0;

        // This property represents the maximum number of elements (capacity) in the vector
        public int Capacity
        {
            get { return data.Length; }
        }

        // This is an overloaded constructor
        public Vector(int capacity)
        {
            data = new T[capacity];
        }

        // This is the implementation of the default constructor
        public Vector() : this(DEFAULT_CAPACITY) { }

        // 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.
        // For example, introducing the following indexer you may address an element of the vector as vector[i] or vector[0] or ...
        public T this[int index]
        {
            get
            {
                if (index >= Count || index < 0) throw new IndexOutOfRangeException();
                return data[index];
            }
```

```
            set
            {
                if (index >= Count || index < 0) throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }

        // This private method allows extension of the existing capacity of the vector by another 'extraCapacity' elements.
        // The new capacity is equal to the existing one plus 'extraCapacity'.
        // It copies the elements of 'data' (the existing array) to 'newData' (the new array), and then makes data pointing to 'newData'.
        private void ExtendData(int extraCapacity)
        {
            T[] newData = new T[Capacity + extraCapacity];
            for (int i = 0; i < Count; i++) newData[i] = data[i];
            data = newData;
        }

        // This method adds a new element to the existing array.
        // If the internal array is out of capacity, its capacity is first extended to fit the new element.
        public void Add(T element)
        {
            if (Count == Capacity) ExtendData(DEFAULT_CAPACITY);
            data[Count++] = element;
        }

        // This method searches for the specified object and returns the zerobased index of the first occurrence within the entire data structure.
        // This method performs a linear search; therefore, this method is an O(n) runtime complexity operation.
        // If occurrence is not found, then the method returns 1.
        // Note that Equals is the proper method to compare two objects for equality, you must not use operator '=' for this purpose.
        public int IndexOf(T element)
        {
            for (var i = 0; i < Count; i++)
            {
                if (data[i].Equals(element)) return i;
            }
            return -1;
        }
        public override string ToString()
        {

            var sb = new StringBuilder();

            for (int i = 0; i < Count; i++)
                sb.Append(String.Format("{0},", data[i]));

            string sd = String.Join(",", sb);
```

```
            return "[" + sd.TrimEnd(',') + "]";


        }

        public ISorter Sorter { set; get; } = new DefaultSorter();

        internal class DefaultSorter : ISorter
        {
            public void Sort<K>(K[] sequence, IComparer<K> comparer) where K :
             ↪ IComparable<K>
            {
                if (comparer == null) comparer = Comparer<K>.Default;
                Array.Sort(sequence, comparer);
            }
        }

        public void Sort()
        {
            if (Sorter == null) Sorter = new DefaultSorter();
            Array.Resize(ref data, Count);
            Sorter.Sort(data, null);
        }

        public void Sort(IComparer<T> comparer)
        {
            if (Sorter == null) Sorter = new DefaultSorter();
            Array.Resize(ref data, Count);
            if (comparer == null) Sorter.Sort(data, null);
            else Sorter.Sort(data, comparer);
        }

    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace Vector
{


    public class BubbleSort : ISorter
    {


        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
        {


            for (int i = 1; i <= sequence.Length ; i++)
            {
                int swaps = 0;
                for (int j = 0; j < sequence.Length - i; j++)
                {
                    K temp;

                    if (comparer.Compare(sequence[j], sequence[j + 1]) > 0)
                    {

                        temp = sequence[j];
                        sequence[j] = sequence[j + 1];
                        sequence[j + 1] = temp;
                        swaps += 1;

                    }
                    else
                    {

                        continue;
                    }
                }



                if (swaps == 0)

                    break;
            }
        }


    }


```

```
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace Vector
{
    public class InsertionSort : ISorter
    {
        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
        {
            for (int i = 0; i < sequence.Length; i++)
            {
                int j = i;
                while (j > 0 && comparer.Compare(sequence[j-1],sequence[j]) > 0)
                {
                    K temp = sequence[j];
                    sequence[j] = sequence[j - 1];
                    sequence[j - 1] = temp;
                    j -= 1;
                }
            }

        }
    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace Vector
{


    public class SelectionSort : ISorter
    {
        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K :
        ↪   IComparable<K>
        {
            for (int i = 0; i < sequence.Length; i++)
            {

                int  min = i;
                for (int j = i +1; j < sequence.Length; j++)
                {
                    if (comparer.Compare(sequence[j], sequence[min]) < 0)
                        min = j;
                    else
                    {
                        continue;
                    }
                }

                if (min != i)
                {
                    K temp = sequence[i];
                    sequence[i] = sequence[min];
                    sequence[min] = temp;
                }

            }


        }

    }
}
```

9 Recursive Sorting

Implementation of Advanced Sorting

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

task

Date	**Author**	**Comment**
2020/08/18 14:16	Nebojsa Miletic	more time needed
2020/08/25 09:50	Nebojsa Miletic	Ready to Mark
2020/09/03 16:04	Maksym Slavnenko	MergeSort 'Ω(?)' and 'O(?)', QuickSort 'Ω(?)'and 'O(?)'
2020/09/03 16:04	Maksym Slavnenko	'[9,20,8,5,40,5,3,1,16,29]' - what would be the worst pivot point to pick here for a Randomized Quick Sort?
2020/09/03 16:05	Maksym Slavnenko	Demonstrate
2020/09/03 16:05	Maksym Slavnenko	1. Auxiliary Space : Mergesort uses extra space, quicksort requires little space and exhibits good cache locality. Quick sort is an in-place sorting algorithm. In-place sorting means no additional storage space is needed to perform sorting. Merge sort requires a temporary array to merge the sorted arrays and hence it is not in-place giving Quick sort the advantage of space.2. Worst Cases : The worst case of quicksort O(n2) can be avoided by using randomized quicksort. It can be easily avoided with high probability by choosing the right pivot. Obtaining an average case behavior by choosing right pivot element makes it improvise the performance and becoming as efficient as Merge sort.3. Locality of reference : Quicksort in particular exhibits good cache locality and this makes it faster than merge sort in many cases like in virtual memory environment.4. Merge sort is better for large data structures: Mergesort is a stable sort, unlike quicksort and heapsort, and can be easily adapted to operate on linked lists and very large lists stored on slow-to-access media such as disk storage or network attached storage. Refer this for details https://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/
2020/09/03 16:12	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Recursive Sorting

Submitted By:
Nebojsa Miletic
mileticn
2020/08/25 09:50

Tutor:
Maksym Slavnenko

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

task

August 25, 2020

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

namespace Vector
{
    public class RandomizedQuickSort : ISorter
    {
        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
        {
                if (comparer == null) comparer = Comparer<K>.Default;
                Array.Sort(sequence, comparer);
        }

        public void Sort<K>(K[] sequence, IComparer<K> comparer,int a,int b) where K : IComparable<K>
        {




            if (a >= b) return;
            int left = a;
            int right = b - 1;


            Random random = new Random();
            K pivot = sequence.ElementAt(random.Next(0,sequence.Length));
            K temp;



            while(left <= right)
            {
                while (left <= right && comparer.Compare(sequence[left], pivot) < 0) left++;
                while (left <= right && comparer.Compare(sequence[right], pivot) > 0) right --;
                if (left <= right)
                {
                    temp = sequence[left];
                    sequence[left] = sequence[right];
                    sequence[right] = temp;
                }
                left++;
                right--;
            }

            temp = sequence[left];
            sequence[left] = sequence[right];
            sequence[right] = temp;

```

```
            }

        }

}
```

```
using System;
using System.Collections.Generic;

namespace Vector
{
    public class MergeSortTopDown : ISorter
    {

        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
        {

            int n = sequence.Length;
            if (n < 2) return;
            int mid = n / 2;
            int sHalfMid;




            if (n % 2 == 0)

                sHalfMid = mid;
            else
                sHalfMid = mid + 1;

            K[] fHalf = new K[sHalfMid];
            K[] sHalf = new K[mid];

            Array.Copy(sequence, 0, fHalf, 0, sHalfMid);

            Array.Copy(sequence,sHalfMid,sHalf,0, mid);



            Sort(fHalf, comparer);
            Sort(sHalf, comparer);




            Merge(fHalf, sHalf, sequence, comparer);

        }



        public void Merge<K>(K[] firstHalf, K[] secondHalf, K[] sequence, IComparer<K> comparer)
        {
            int i = 0;
            int j = 0;

```

```
            while (i + j < sequence.Length )
                if (j == secondHalf.Length || (i < firstHalf.Length &&
                ↪  comparer.Compare(firstHalf[i], secondHalf[j]) <= 0))
                    sequence[i + j] = firstHalf[i++];
                else
                    sequence[i + j] = secondHalf[j++];


        }

    }
}
```

```
using System;
using System.Collections.Generic;

namespace Vector
{
    public class MergeSortBottomUp : ISorter
    {

        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K :
         ↪ IComparable<K>
        {
            int n = sequence.Length;

            K[] src = sequence;
            K[] dest = new K[n];
            K[] temp;


            for (int i = 1; i < n; i *= 2)
            {
                for (int j = 0; j < n; j += 2*i)

                    Merge(sequence, dest, comparer, i, j);
                temp = src;
                src = dest;
                dest = temp;
            }

            if (sequence != src)
                Array.Copy(sequence, 0, src, 0, n);
        }





        public void Merge<K>(K[] input, K[] output,IComparer<K> comparer,int
         ↪ start,int inc)
        {
            int end1 = Math.Min(start + inc, input.Length);
            int end2 = Math.Min(start + 2 * inc, input.Length);

            int x = start;
            int y = start + inc;
            int z = start;

            while (x < end1 && y < end2)
                if (comparer.Compare(input[x], input[y]) < 0)
                    output[z++] = input[x++];
                else
                    output[z++] = input[y++];


```

```
            if (x < end1)
                Array.Copy(input, x, output, z, end1 - x);
            else
                Array.Copy(input, y, output, z, end2 - y);


        }
    }
}
```

10 Iteration and Search

Iteration and Search

Outcome	Weight
Implement Solutions	♦◊◊◊◊

task

Outcome	Weight
Document solutions	♦◊◊◊◊

task

Date	Author	Comment
2020/08/25 09:46	Nebojsa Miletic	Ready to Mark
2020/09/03 16:05	Maksym Slavnenko	Demonstrate
2020/09/03 16:12	Maksym Slavnenko	What interfaces did you use to implement and 'Iterator' pattern?What is the best and worst case complexities of Binary Search? What is recursion? What benefits does iterator provide in comparison to a regular 'for' loop?https://jonskeet.uk/csharp/csharp2/iterators.html
2020/09/03 16:13	Maksym Slavnenko	image comment
2020/09/03 16:13	Maksym Slavnenko	Fix and Resubmit
2020/09/03 16:14	Maksym Slavnenko	'foreach'
2020/09/03 16:14	Maksym Slavnenko	You passed the interview on this one, please fix your code and resubmit and I will mark it as 'Complete'
2020/09/15 19:02	Nebojsa Miletic	Ready to Mark
2020/09/15 19:02	Maksym Slavnenko	Time Exceeded
2020/09/24 16:22	Maksym Slavnenko	Binary Search need to be recursive
2020/09/24 16:23	Maksym Slavnenko	Fix and Resubmit
2020/10/01 09:56	Nebojsa Miletic	Ready to Mark
2020/10/01 09:56	Maksym Slavnenko	Time Exceeded
2020/10/08 17:21	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Iteration and Search

Submitted By:
Nebojsa Miletic
mileticn
2020/10/01 09:56

Tutor:
Maksym Slavnenko

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

task

October 1, 2020

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

namespace Vector
{

    public class Vector<T> : IEnumerable<T> where T : IComparable<T>
    {
        // This constant determines the default number of elements in a newly created vector.
        // It is also used to extended the capacity of the existing vector
        private const int DEFAULT_CAPACITY = 10;

        // This array represents the internal data structure wrapped by the vector class.
        // In fact, all the elements are to be stored in this private  array.
        // You will just write extra functionality (methods) to make the work with the array more convenient for the user.
        private T[] data;
        private T[] _data;

        // This property represents the number of elements in the vector
        public int Count { get; private set; } = 0;

        // This property represents the maximum number of elements (capacity) in the vector
        public int Capacity
        {
            get { return data.Length; }
        }

        // This is an overloaded constructor
        public Vector(int capacity)
        {
            data = new T[capacity];
        }

        // This is the implementation of the default constructor
        public Vector() : this(DEFAULT_CAPACITY) { }

        // 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.
        // For example, introducing the following indexer you may address an element of the vector as vector[i] or vector[0] or ...
        public T this[int index]
        {
            get
            {
                if (index >= Count || index < 0) throw new IndexOutOfRangeException();
```

```
                return data[index];
            }
            set
            {
                if (index >= Count || index < 0) throw new IndexOutOfRangeException();
                data[index] = value;
            }
        }


        // This private method allows extension of the existing capacity of the vector by another 'extraCapacity' elements.
        // The new capacity is equal to the existing one plus 'extraCapacity'.
        // It copies the elements of 'data' (the existing array) to 'newData' (the new array), and then makes data pointing to 'newData'.
        private void ExtendData(int extraCapacity)
        {
            T[] newData = new T[Capacity + extraCapacity];
            for (int i = 0; i < Count; i++) newData[i] = data[i];
            data = newData;
        }

        // This method adds a new element to the existing array.
        // If the internal array is out of capacity, its capacity is first extended to fit the new element.
        public void Add(T element)
        {
            if (Count == Capacity) ExtendData(DEFAULT_CAPACITY);
            data[Count++] = element;
        }

        // This method searches for the specified object and returns the zerobased index of the first occurrence within the entire data structure.
        // This method performs a linear search; therefore, this method is an O(n) runtime complexity operation.
        // If occurrence is not found, then the method returns 1.
        // Note that Equals is the proper method to compare two objects for equality, you must not use operator '=' for this purpose.
        public int IndexOf(T element)
        {
            for (var i = 0; i < Count; i++)
            {
                if (data[i].Equals(element)) return i;
            }
            return -1;
        }

        public override string ToString()
        {

            var sb = new StringBuilder();

```

```
            for (int i = 0; i < Count; i++)
                sb.Append(String.Format("{0},", data[i]));

            string sd = String.Join(",", sb);
            return "[" + sd.TrimEnd(',') + "]";


        }


        public ISorter Sorter { set; get; } = new DefaultSorter();

        internal class DefaultSorter : ISorter
        {
            public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
            {
                if (comparer == null) comparer = Comparer<K>.Default;
                Array.Sort(sequence, comparer);
            }
        }

        public void Sort()
        {
            if (Sorter == null) Sorter = new DefaultSorter();
            Array.Resize(ref data, Count);
            Sorter.Sort(data, null);
        }

        public void Sort(IComparer<T> comparer)
        {
            if (Sorter == null) Sorter = new DefaultSorter();
            Array.Resize(ref data, Count);
            if (comparer == null) Sorter.Sort(data, null);
            else Sorter.Sort(data, comparer);
        }

        //TODO: Your task is to implement all the remaining methods.
        //Read the instruction carefully, study the code examples from above as they should help you to write the rest of the code.

        public int BinarySearch(T element, IComparer<T> comparer)
        {

            return BinarySearch( element, 0, data.Length - 1);
        }



        public int BinarySearch(T element,int imin,int imax)
        {
            Comparer<T> comparer = Comparer<T>.Default;

```

```
            if (imin >= imax)
            {
                return 0;
            }

            int mid = (imin + imax) / 2;
            while (imax > imin)
            {
                if (comparer.Compare(element, data[mid]) == 0)
                    return mid;

                else if (comparer.Compare(element, data[mid]) < 0)
                    BinarySearch(element, imin, mid - 1);

                else
                    BinarySearch(element, mid + 1, imax);

                return -1;

            }

            return -1;


        }



        public IEnumerator<T> GetEnumerator()
        {
            return new Iterator(this);
        }



        private IEnumerator GetEnumerator1()
        {
            return GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator1();
        }




        internal class Iterator : IEnumerator<T>
        {


            private int currIndex = -1;
```

```
            Vector<T> _data = new Vector<T>();



            public Iterator(Vector<T> vector)
            {
                _data = vector;
            }

            public T Current
            {

                get
                {
                    try
                    {
                        return _data[currIndex];

                    }
                    catch (IndexOutOfRangeException)
                    {
                        throw new InvalidOperationException();
                    }
                }


            }

            private T Current1
            {
                get { return Current; }
            }


            object IEnumerator.Current
            {
                get { return Current1; }
            }

            void IDisposable.Dispose() { }


            public bool MoveNext()
            {
                return ++currIndex < _data.Count;


            }



            public void Reset()
            {
```

```
                    currIndex = -1;
                }


            }
        }

}
```

11 Doubly Linked List

Doubly Linked List

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

task

Date	**Author**	**Comment**
2020/08/25 09:51	Nebojsa Miletic	Ready to Mark
2020/09/03 16:15	Maksym Slavnenko	What is the name of the first element of the a linked list? What is the name of the last element? What is the complexity of **getting** an 'n'th element of a linked list? What about an array? What is the complexity of **removing** an 'n'th element of a linked list? What about an array?
2020/09/03 16:15	Maksym Slavnenko	image comment
2020/09/03 16:16	Maksym Slavnenko	I think you have introduced a memory leak in your 'Remove' method. You need to invalidate the node that you delete 'node_.Next = null;' 'node_.Previous = null' in order for the garbage collector to clear the memory.
2020/09/03 16:16	Maksym Slavnenko	the same with the 'Clear' method. You can't just set the Count to 0
2020/09/03 16:16	Maksym Slavnenko	Fix and Resubmit
2020/09/10 18:09	Nebojsa Miletic	Ready to Mark
2020/09/24 16:23	Maksym Slavnenko	Does your code pass the tests?
2020/09/24 16:23	Maksym Slavnenko	Please drag and drop the screenshot of the running program here
2020/09/24 16:23	Maksym Slavnenko	Fix and Resubmit
2020/09/24 18:14	Nebojsa Miletic	image comment
2020/10/08 14:56	Nebojsa Miletic	Ready to Mark
2020/10/08 14:56	Maksym Slavnenko	Time Exceeded
2020/10/08 17:22	Maksym Slavnenko	image comment
2020/10/08 17:22	Maksym Slavnenko	IF you can, please fix it in your portfolio
2020/10/08 17:22	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Doubly Linked List

Submitted By:
Nebojsa Miletic
mileticn
2020/10/09 12:06

Tutor:
Maksym Slavnenko

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

task

October 9, 2020

```
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.
        // The both are introduced in order to significantly simplify the implementation of the class and make insertion functionality reduced just to a AddBetween(...)
        // 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> is formally empty.
```

```
        // Remember about this crucial fact when you design and code other functions of the DoublyLinkedList<T> in this task.
        private Node<T> Head { get; set; }
        private Node<T> Tail { get; set; }
        public int Count { get; private set; } = 0;

        public DoublyLinkedList()
        {
            Head = new Node<T>(default(T), null, null);
            Tail = new Node<T>(default(T), Head, null);
            Head.Next = Tail;
        }

        public INode<T> First
        {
            get
            {
                if (Count == 0) return null;
                else return Head.Next;
            }
        }

        public INode<T> Last
        {
            get
            {
                if (Count == 0) return null;
                else return Tail.Previous;
            }
        }

        public INode<T> After(INode<T> node)
        {
            if (node == null) throw new NullReferenceException();
            Node<T> node_current = node as Node<T>;
            if (node_current.Previous == null || node_current.Next == null) throw new InvalidOperationException("The node referred as 'before' is no longer in the list");
            if (node_current.Next.Equals(Tail)) return null;
            else return node_current.Next;
        }

        public INode<T> AddLast(T value)
        {
            return AddBetween(value, Tail.Previous, Tail);
        }

        // 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.
        // Use it when you wish to insert a new value (node) into the DoublyLinkedList<T>
        private Node<T> AddBetween(T value, Node<T> previous, Node<T> next)
        {
```

```
            Node<T> node = new Node<T>(value, previous, next);
            previous.Next = node;
            next.Previous = node;
            Count++;
            return node;
        }

        public INode<T> Find(T value)
        {
            Node<T> node = Head.Next;
            while (!node.Equals(Tail))
            {
                if (node.Value.Equals(value)) return node;
                node = node.Next;
            }
            return null;
        }

        public override string ToString()
        {
            if (Count == 0) return "[]";
            StringBuilder s = new StringBuilder();
            s.Append("[");
            int k = 0;
            Node<T> node = Head.Next;
            while (!node.Equals(Tail))
            {
                s.Append(node.ToString());
                node = node.Next;
                if (k < Count - 1) s.Append(",");
                k++;
            }
            s.Append("]");
            return s.ToString();
        }

        // TODO: Your task is to implement all the remaining methods.
        // Read the instruction carefully, study the code examples from above as
        ↪ they should help you to write the rest of the code.

        public INode<T> Before(INode<T> node)
        {
            if(node == null) throw new NullReferenceException();
            Node<T> node_current = node as Node<T>;
            if (node_current.Previous == null || node_current.Next == null) throw
            ↪ new InvalidOperationException("The node referred as 'before' is no
            ↪ longer in the list");
            if (node_current.Next.Equals(Tail)) return null;
            else return node_current.Previous;


        }

```

```
        public INode<T> AddFirst(T value)
        {
           return AddBetween(value,Head,Head.Next);

        }

        public INode<T> AddBefore(INode<T> before, T value)
        {
           Node<T> node = before as Node<T>;
           return  AddBetween(value, node.Previous, node);

        }

        public INode<T> AddAfter(INode<T> after, T value)
        {
            Node<T> node = after as Node<T>;
            return AddBetween(value, node, node.Next);
        }

        public void Clear()
        {

            Node<T> current = Head;
            while (current != null)
            {

                Node<T> temp = current;
                current = current.Next;
                temp = null;
            }
            Head = null;
            Count = 0;

        }

        public void Remove(INode<T> node)
        {
            Node<T> node_ = node as Node<T>;

            if (Find(node.Value) == null) throw new InvalidOperationException();



                if (Head == node_)
                    RemoveFirst();

                if (node_.Next != null)

                    node_.Next.Previous = node_.Previous;



                if (node_.Previous != null)
```

```

                    node_.Previous.Next = node_.Next;




            Count--;



        }

        public void RemoveFirst()
        {
            if (Count == 0)
                throw new InvalidOperationException();
            Head = Head.Next;
            Head.Previous = null;
            Count--;
        }





        public void RemoveLast()
        {
            if (Count == 0)
                throw new InvalidOperationException();
            Tail = Tail.Previous;
            Tail.Next = null;
            Count--;

        }

    }
}
```

12 Problem solving: Search and Stack

Problem solving: Search and Stack

Outcome	**Weight**
Complexity	♦◊◊◊◊

task

Date	Author	Comment
2020/09/15 18:57	Nebojsa Miletic	Ready to Mark
2020/09/24 16:24	Maksym Slav-nenko	q1. Here is a sample array for you‘[1,2,3,5,8,13,21,34,55,89]‘ x = 42Please provide all the steps here in the chat box on this specific array.q2. Here is a sample Stack for you. ‘[10,5,7,5]‘Please provide all the steps and states in each step when you push elements (starting from 10) into this stack one by one and then pop elements one by one from it. I expect to see 8 steps with 4 pushes and 4 pops and 8 states after each operation. Thanks.
2020/09/24 16:24	Maksym Slav-nenko	q1 how would you sort an array? What sorting algorithm should be used?
2020/09/24 16:25	Maksym Slav-nenko	to satisfy O(n*log(n)) complexity?
2020/09/24 16:25	Maksym Slav-nenko	Fix and Resubmit
2020/09/24 16:27	Maksym Slav-nenko	q1 looks ‘O(nˆ2)‘
2020/10/01 10:15	Nebojsa Miletic	q2. make a variable minElement for tracing minimum in the list. We push 10 in the list and minElement is equal (x) =10.Next, push 5 into the stack. Compare X and minElement. x is less than minElement (10). So, minElement is now 5. We insert (2 * 10 - 5) = 15 in the stack.
2020/10/01 12:06	Nebojsa Miletic	Next is 7.Compare it with the minelement, It is greater than minElement, and we just push it at the top of the stack. Similar with next element, compare 5 with the minElement,it is equal than miElement, and push it into the stack. Next move is Pop functioRemove element from the top. Removed element is y. compare y with the minElement. It is => than minElement, so Pop it from the stack. Next element is 7. It is greater than minElement so pop it form the stack. next element. Next element is 5. That was the minimum element in the stac so it pops out, and next element is the last in the stack, so it gets to become minElement. Pop 10 from the stack, and the stack is empty.
2020/10/01 12:08	Nebojsa Miletic	Important fact about this method is that the stack actually does’t hold actual value of the element if it is minimum. Actual value is stored in minValue varable.
2020/10/01 12:12	Nebojsa Miletic	q1. First for loop iterates through the list starting from first element,so i = 1: Inner for loop starts from the last element which is 89.
2020/10/01 12:18	Nebojsa Miletic	So i + j =90; 90 > x (42). We do herer j–, so the next value is55 i + j = 56 and 56 < x. So going further down the list with the j value. 34 + 1 = 35. This time the value i + j < x, so we iterate up the list with i++. The value of i is 2 now , so 34 + 2 = 36 still less than x. Next i is 3 so i + j sill less than x. i++ gives us 5 this time 34 + 5 < 42. Value 8 is next, so this time 34 + 8 = 42 and the loop stops.
2020/10/01 12:20	Nebojsa Miletic	Ready to Mark
2020/10/01 12:20	Maksym Slav-nenko	Time Exceeded
2020/10/06 13:37	Maksym Slav-nenko	that’s much better. Great solution, Neb!

Deakin University

Data Structures and Algorithms

OnTrack Submission

Problem solving: Search and Stack

Submitted By:
Nebojsa Miletic
mileticn
2020/10/01 12:20

Tutor:
Maksym Slavnenko

Outcome	**Weight**
Complexity	♦◊◊◊◊

task

October 1, 2020

Practical Task 6.1

1. *Design a $\theta(n \log n)$ time algorithm that, given a set S of n integer numbers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.*

First step in solving this problem would be to sort the set with merge sort in ascending order. Then, there should be a variable which represent the number of elements:

int no.Of.elements = set S number of elements;

The next step is iterating through the set:

for each element in set S [i];

and another for loop starting from the end of set with the condition:
if i+j > x;
reduce j – j--;
if i + j < x;
go to next i ; i++;

if (sum is exactly the x)
return both numbers;

else:

return “No match”

This algorithm has Theta (n log n), because we have sorted list to iterate through, so there is one for loop (n), with inner for loop to iterate through values.

2. *A Stack data structure provides Push and Pop, the two operations to write and read data, respectively. Using the Stack as a starting point design a data structure that, in addition to these two operations, also provides the Min operation to return the smallest element of the stack. Remember that the new data structure must operate in a constant $\theta(1)$ time for all three operations.*

Function PUSH()
Make a variable minElement which updates the minimal element in the list.
If stack is empty, insert x in the stack and make minElement equals to x.
If it is not empty, compare x with minElement:
If x is greater than minElement,insert x

If x is less than minElement, insert (2*x – minElement) into the stack and make minElement equal to x. For example, let previous minElement was 3. Now we want to insert 2. We update minElement as 2 and insert 2*2 -3 = 1 into the stack.

Pop()

Remove element from top. Removed element is y.

If y is greater or equl to minElement, the min element in the stack is still minElement.

If y is less than minElement, the min element now becomes (2*minelemet – y), so update (minElement = 2*minElement – y). This is where we retrieve previous minimum from current minimum and its value in stack. For example, let the element to be removed be 1 and minElement 2. We remove 1 and update minElement as 2*2 – 1 = 3.

Important factor is that stack doesn’t hold actual value of an element if it is minimum so far. Actual minimum is stored in minElement variable.

getMin()

The variable minElement stores the minimum element in the stack, so we just need to trace that variable to retrieve minimum element.

13 Programming - Problem Solving

Dynamic programming

Outcome	Weight
Complexity	♦◊◊◊◊

task

Date	Author	Comment
2020/09/09 09:16	Nebojsa Miletic	Ready to Mark
2020/09/10 17:49	Maksym Slavnenko	Please drag and drop a screenshot of the running program here.
2020/09/10 17:49	Maksym Slavnenko	Demonstrate
2020/09/10 17:56	Nebojsa Miletic	image comment
2020/09/24 16:31	Maksym Slavnenko	image comment
2020/09/24 16:31	Maksym Slavnenko	Let's assume that you have 8 boxes and you created a table 8x8, where in this table would be the maximum possible output for Alex? What will be stored in the cell A2? What about A1?
2020/09/24 16:31	Maksym Slavnenko	Please provide the whole output of the program
2020/09/24 16:32	Maksym Slavnenko	Discuss
2020/10/01 12:24	Nebojsa Miletic	image comment
2020/10/01 12:24	Nebojsa Miletic	image comment
2020/10/01 12:43	Nebojsa Miletic	This solution is bottom-up so in this case A would be T[0,0] = Max(823,823
2020/10/01 13:00	Nebojsa Miletic	) so first value on the list A(0,0) = 823. A1 would be 912. Whoever starts first, the best possible solution will be find at the A(0, n-1), so that is Cindy. Last A(0,n) holds biggest value for the player who starts next.
2020/10/08 17:28	Maksym Slavnenko	H1 for Alex
2020/10/08 17:28	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Programming - Problem Solving

Submitted By:
Nebojsa Miletic
mileticn
2020/09/09 09:16

Tutor:
Maksym Slavnenko

Outcome	**Weight**
Complexity	♦◊◊◊◊

task

September 9, 2020

```
using System;

namespace BoxOfCoins
{
    public class BoxOfCoins
    {
        private static int calculate(int[,] T, int x, int y)
        {


            if (x <= y)

                return T[x,y];

            return 0;
        }


        public static int Solve(int[] boxes)
        {
            int n = boxes.Length;

            if (n == 1)
                return   boxes[0];
            if (n == 2)
                return Math.Max(boxes[0], boxes[1]) - Math.Min(boxes[0], boxes[1]);

            int[,] T = new int [n,n];
            int max = 0;

            for (int iteration = 0; iteration < n; iteration++)
            {
                max += boxes[iteration];
                for (int i = 0, j = iteration; j < n; i++,j++)
                {

                    int alex = boxes[i] + Math.Min(calculate(T, i + 2, j),
                     ↪ calculate(T, i + 1, j - 1));

                    int cindy = boxes[j] + Math.Min(calculate(T, i + 1, j - 1),
                     ↪ calculate(T, i, j - 2));

                    T[i, j] = Math.Max(alex, cindy);

                }
            }

            return T[0, n - 1] - (max - T[0, n - 1]);
        }


    }
}
```

14 AVL-Trees

AVL-Trees

Outcome	Weight
Complexity	♦◊◊◊◊

Task 7.1

Date	Author	Comment
2020/09/17 10:53	Nebojsa Miletic	Ready to Mark
2020/09/24 16:27	Maksym Slavnenko	What is the complexity of the main operations of the AVL-tree (e.g. search, max, min, insert, delete)? What is the height of the AVL-tree? How do you determine whether the tree needs to be balanced?
2020/09/24 16:28	Maksym Slavnenko	image comment
2020/09/24 16:28	Maksym Slavnenko	5 balancing factor is incorrect
2020/09/24 16:28	Maksym Slavnenko	it is -1
2020/09/24 16:29	Maksym Slavnenko	image comment
2020/09/24 16:29	Maksym Slavnenko	the same problem here
2020/09/24 16:30	Maksym Slavnenko	q3 is missing
2020/09/24 16:30	Maksym Slavnenko	Fix and Resubmit
2020/10/01 13:10	Nebojsa Miletic	Insert, delete and search functions in AVL tree have a O(log n) nad Big Theta(log n), while Min and Max have constant time complexity
2020/10/01 13:11	Nebojsa Miletic	For each node, the difference in height of the left and right subtrees is at most 1, which means it is balanced
2020/10/01 13:16	Nebojsa Miletic	the height of a non-empty tree is one greater than the maximum height of its two subtrees
2020/10/01 13:17	Nebojsa Miletic	f it has N nodes, its height is log2(N + 1)
2020/10/01 13:18	Nebojsa Miletic	So, if the height difference is more than 1, we have unbalanced tree and we need to do the balancing operation.
2020/10/02 11:39	Nebojsa Miletic	Ready to Mark
2020/10/02 11:39	Maksym Slavnenko	Time Exceeded
2020/10/08 17:23	Maksym Slavnenko	Complete

DEAKIN UNIVERSITY

DATA STRUCTURES AND ALGORITHMS

ONTRACK SUBMISSION

AVL-Trees

Submitted By:
Nebojsa MILETIC
mileticn
2020/10/02 11:39

Tutor:
Maksym SLAVNENKO

Outcome	**Weight**
Complexity	♦◊◊◊◊

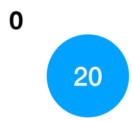
Task 7.1

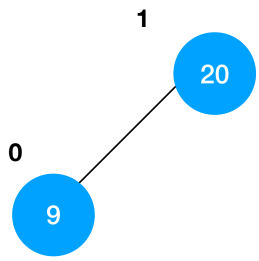
October 2, 2020

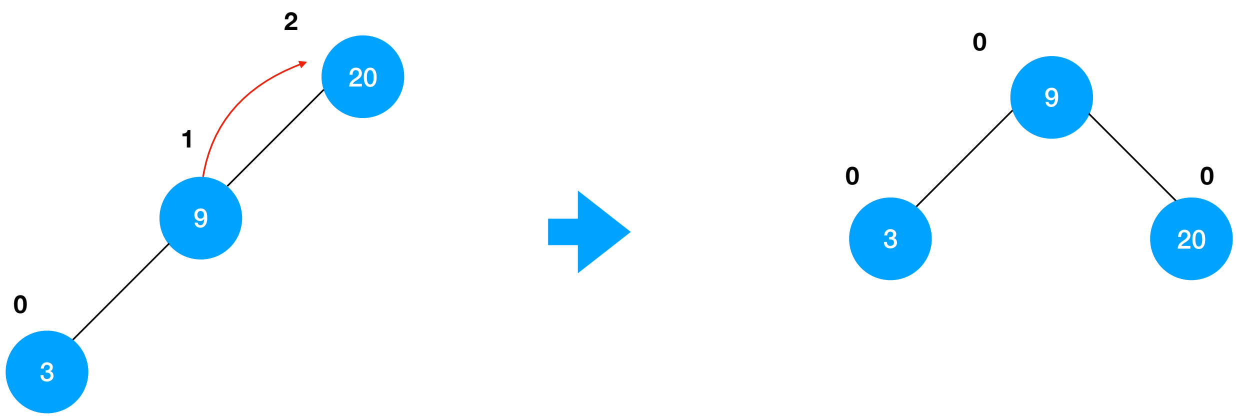
Task 7.1

AVL Tree Insertion and Deletion

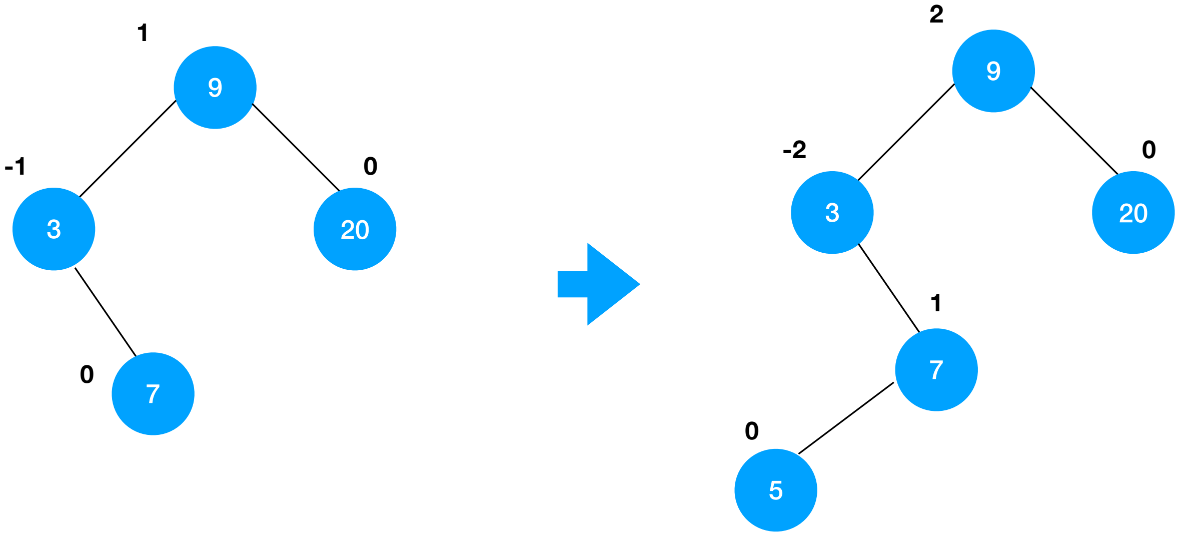
Insertion of the values:



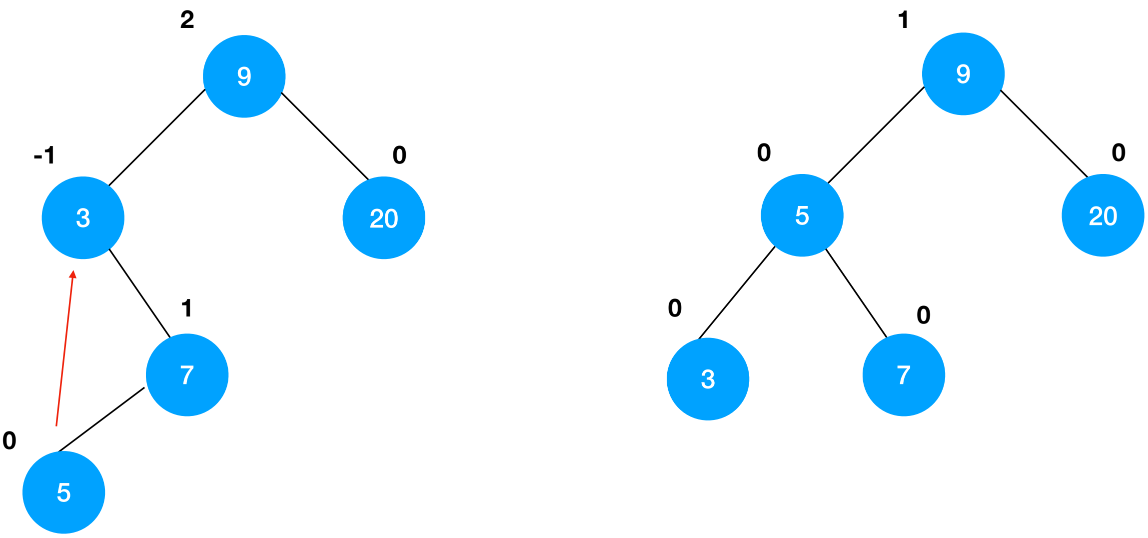

1
20
0
9



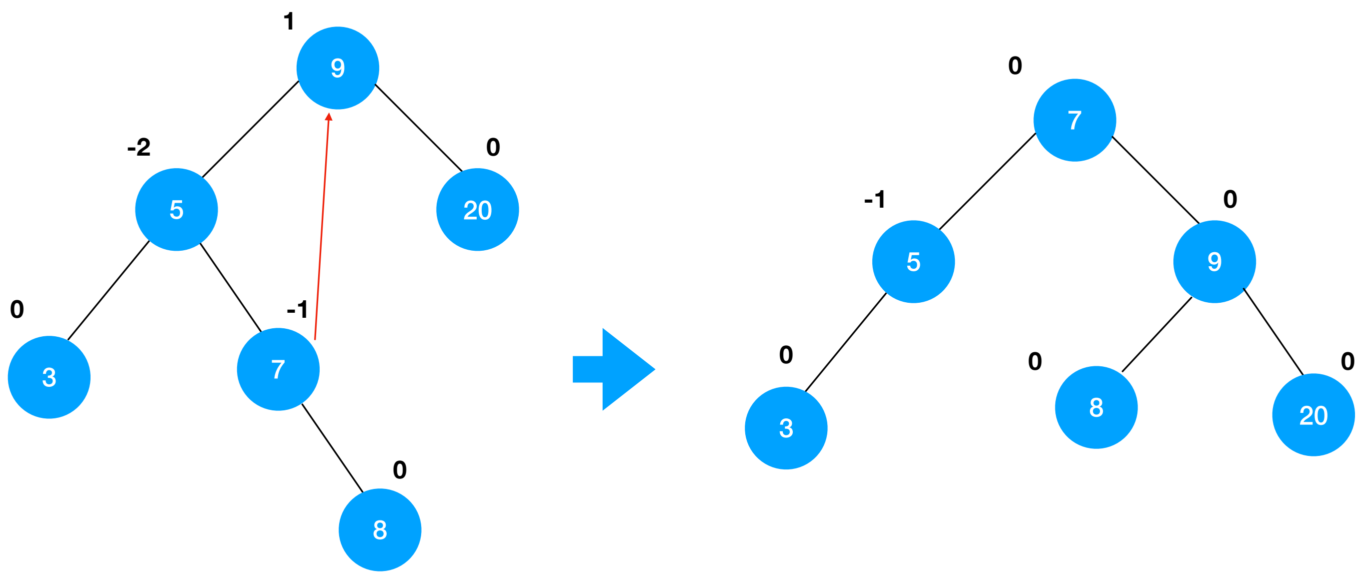
Unbalanced => Right rotation



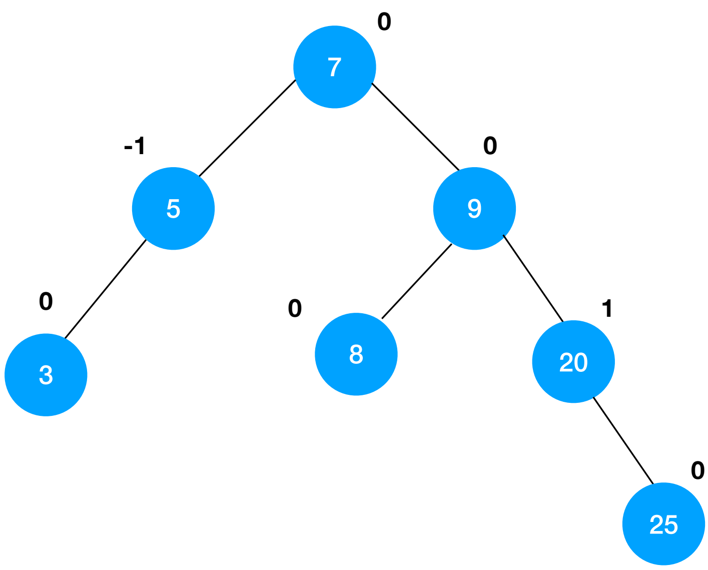
1
9
-1
0
3
20
0
7
2
9
-2
0
3
20
1
7
0
5

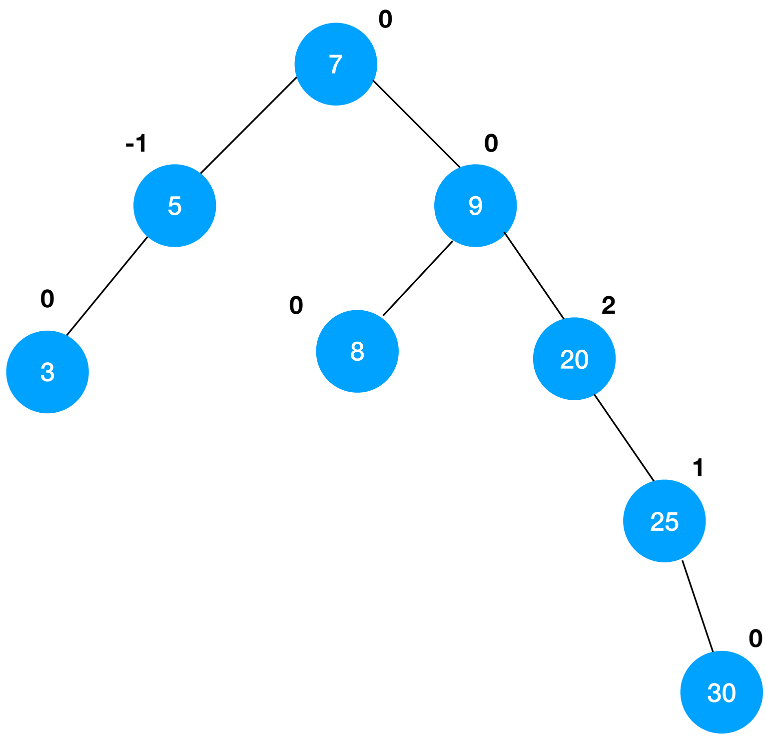


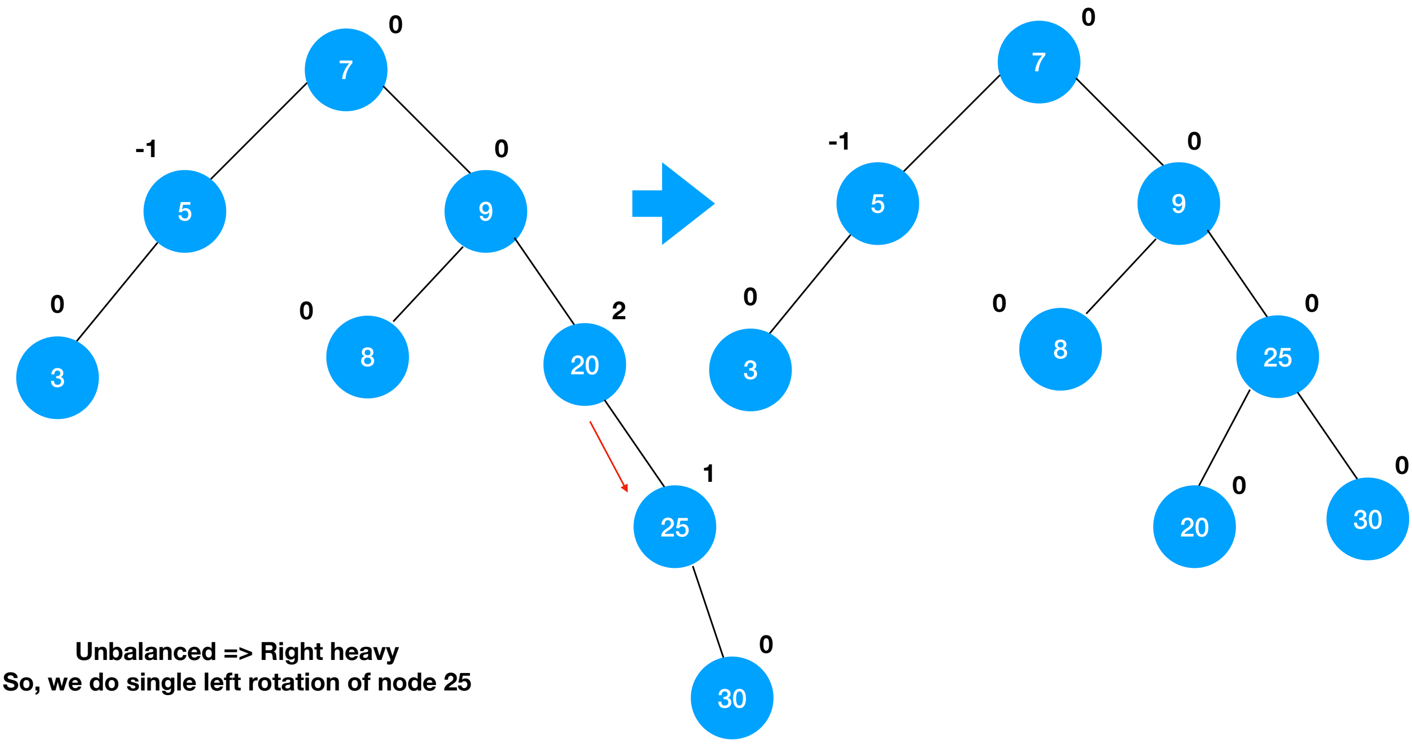
Unbalanced => 3 is right heavy, and node 7 is left heavy.
So, we do a right- left rotation on node 5.



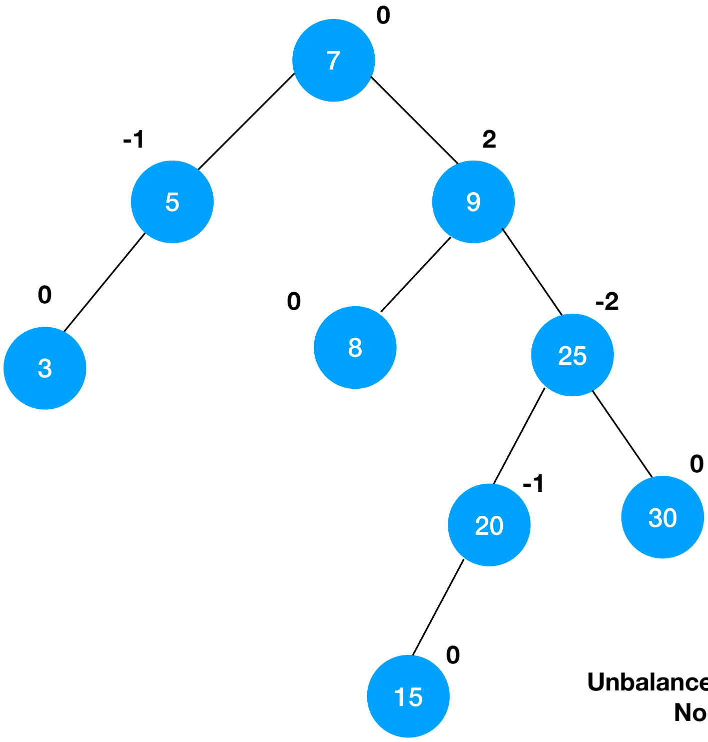
Left heavy. Unbalanced => Left-Right rotation of node 7

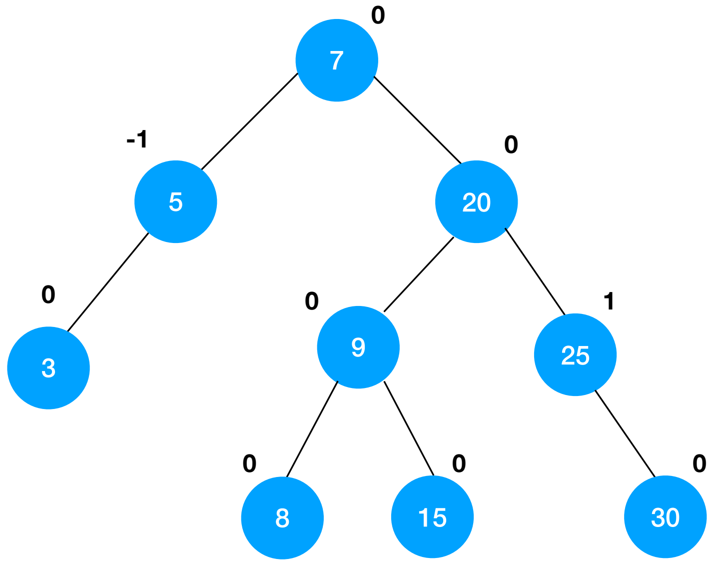

0
7
-1
5
0
9
0
3
0
8
1
20
0
25


0
7
-1
5
0
9
0
3
0
8
2
20
1
25
0
30

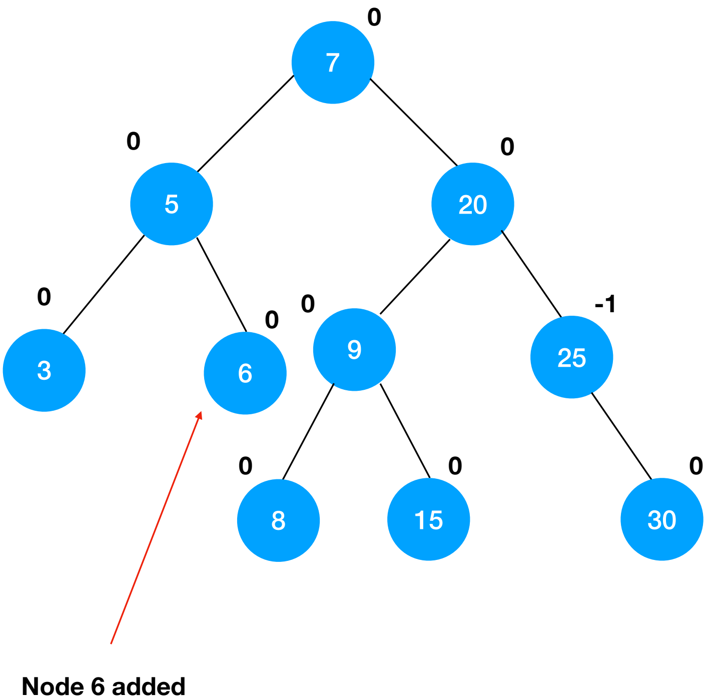


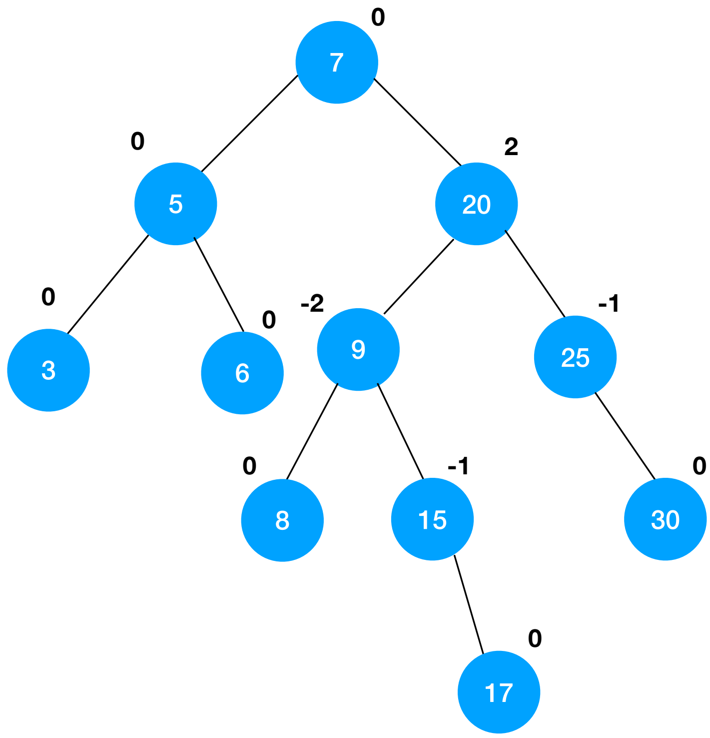
Unbalanced => Right heavy
So, we do single left rotation of node 25

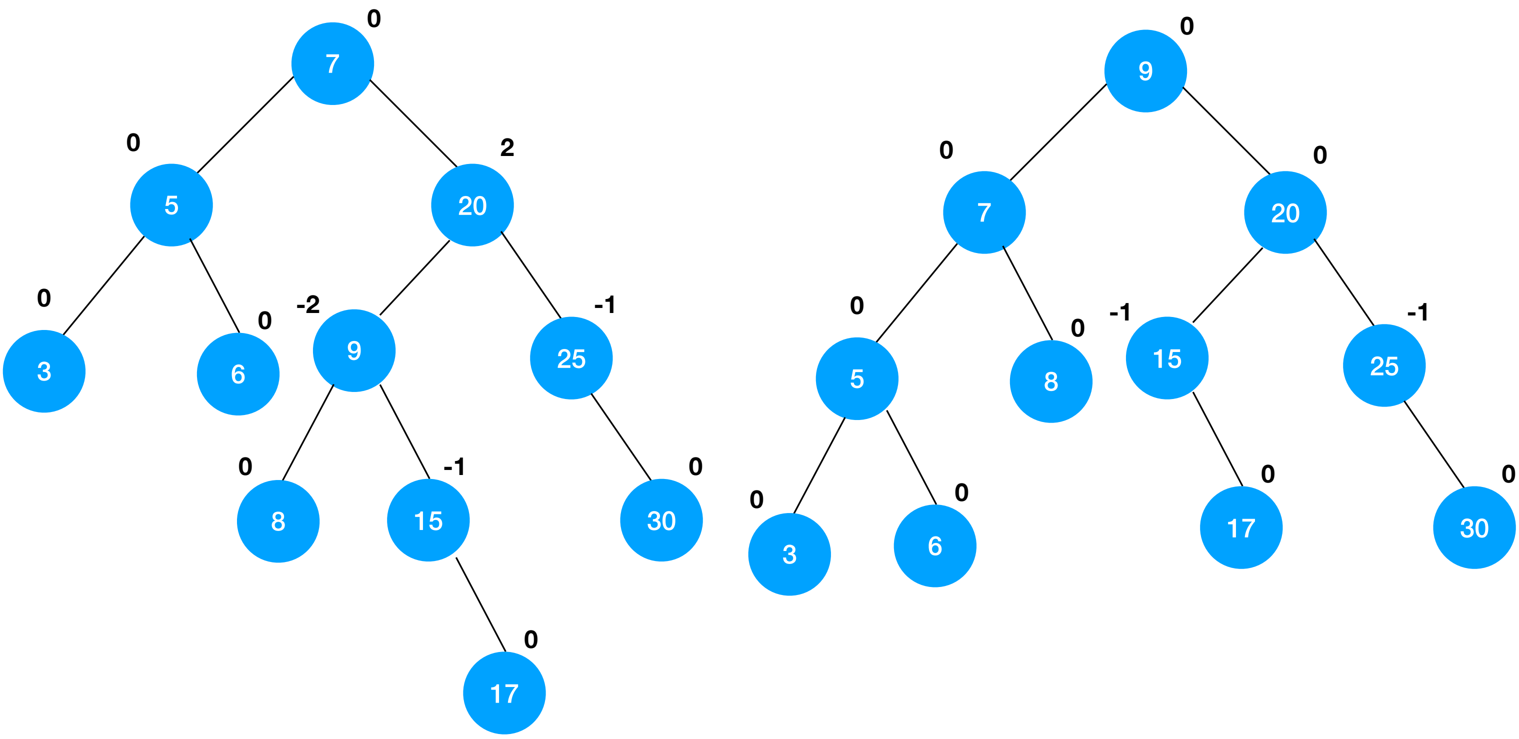




Unbalanced => Right-Left rotation
Node 25 left heavy.

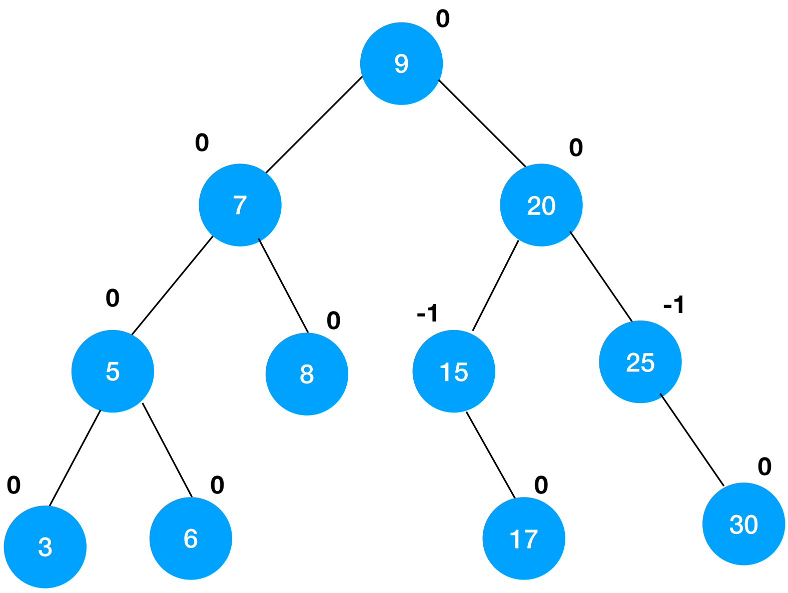

0
7
0
5
0
20
0
3
0
6
0
9
-1
25
0
8
0
15
0
30
Node 6 added


0
7
0
5
2
20
0
3
0
6
-2
9
-1
25
0
8
-1
15
0
30
0
17

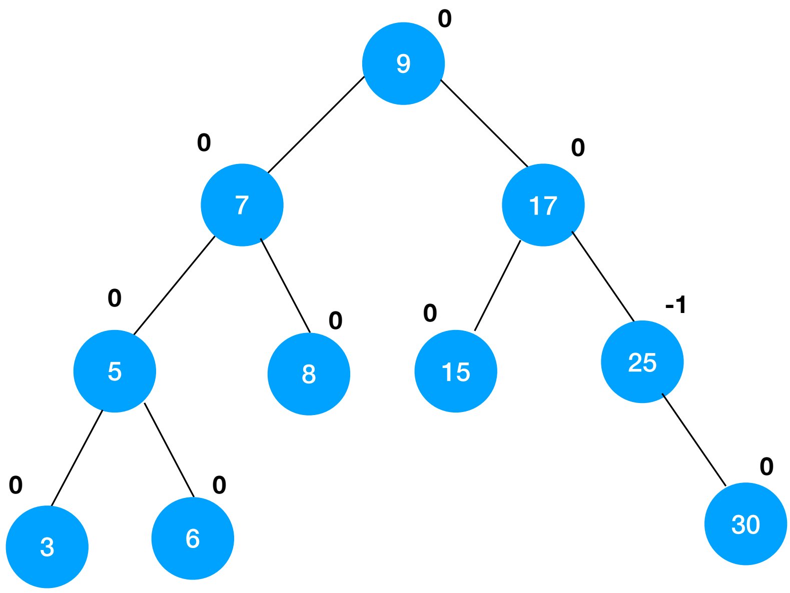


Unbalanced => Right-Left rotation of node 9

Deletion

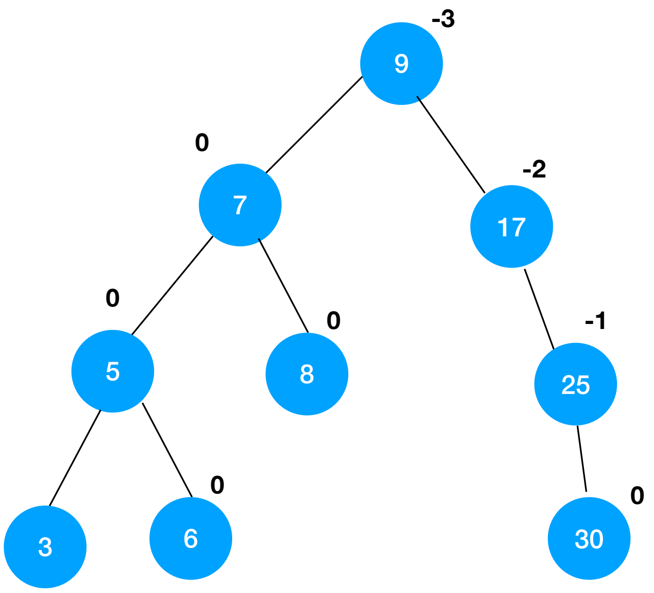


Delete node 20

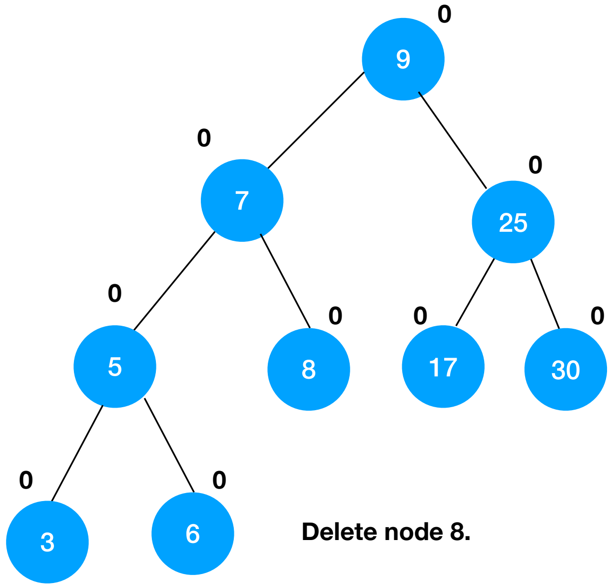


Unbalanced => Node 20 had two children.
The greater was picked(17) and pushed up
using single right rotation

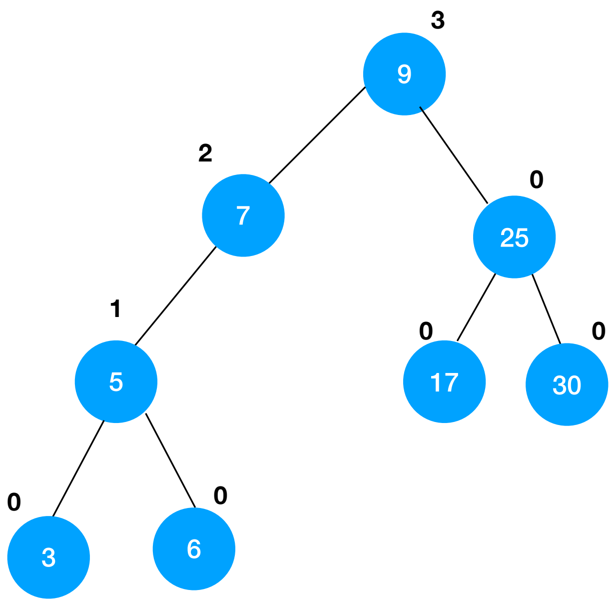
Next node to delete is 15



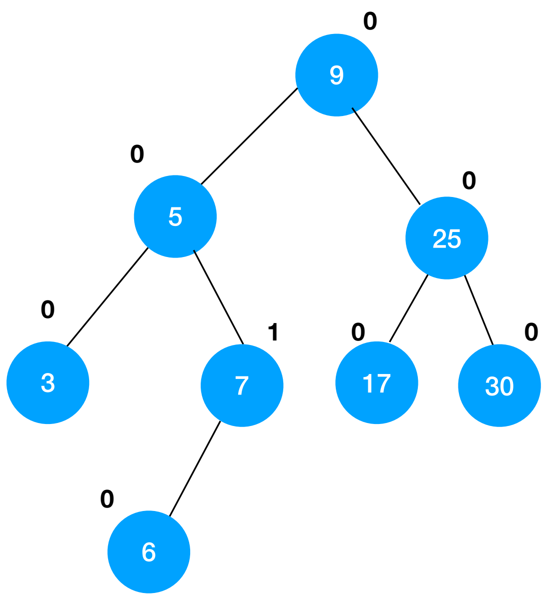
Unbalanced => Right heavy.
So, single left rotation needs
to be done on 25 node



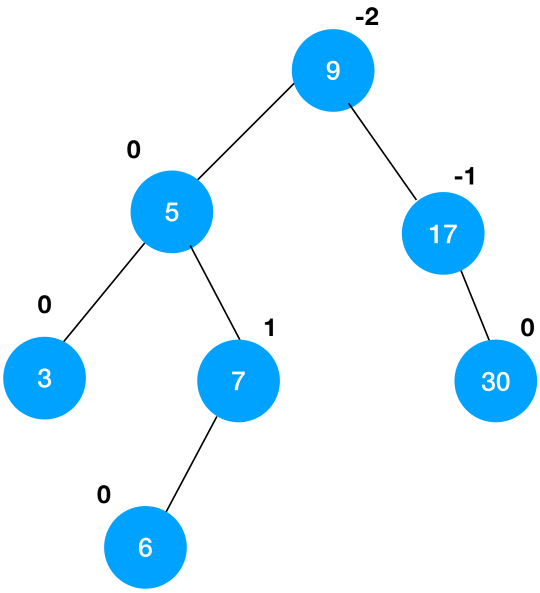
Delete node 8.



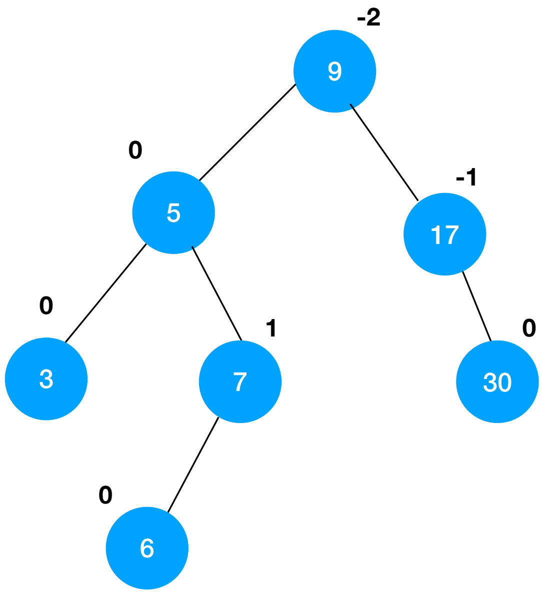
Unbalanced => The tree is left heavy now
Single right rotation on node 5



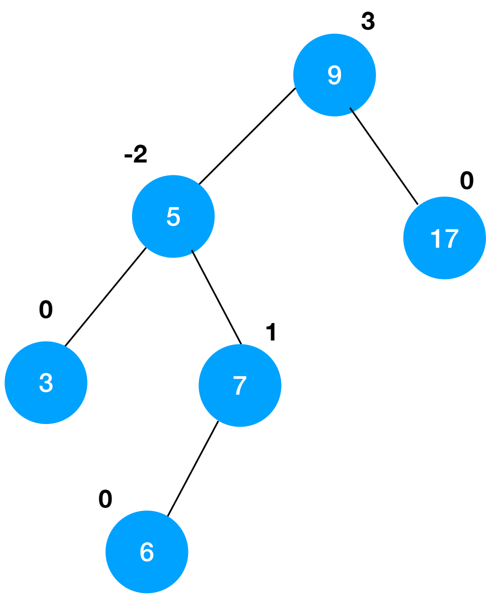
Next node to delete is 25

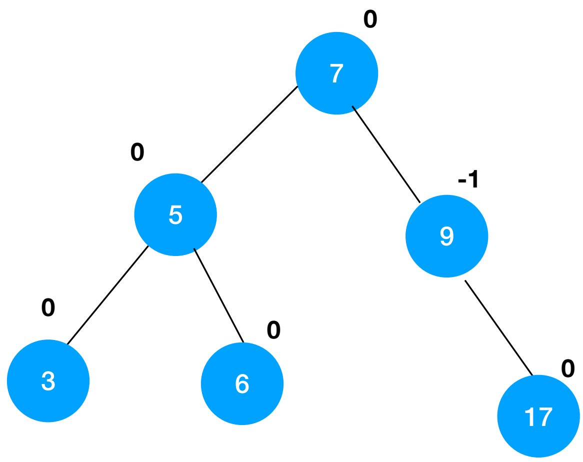


Node 25 has two children
The highest left was node 17



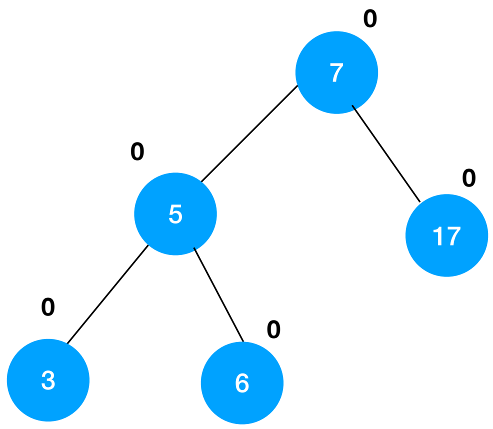
Next node to delete is 30



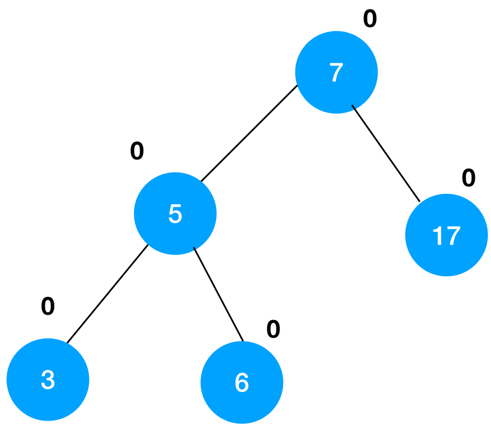


Next node to delete is 9

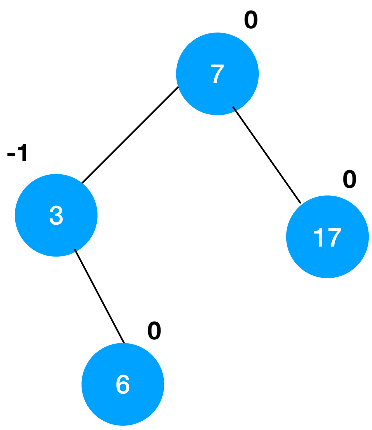
Unbalanced=>Left heavy on node 7.
Right heavy on node 5. Left-Right rotation on nodes mentioned.



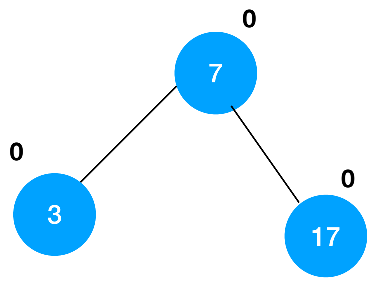
Tree is still balanced



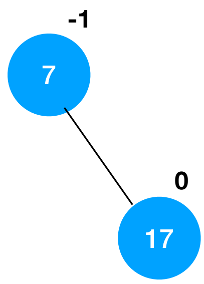
Next node to delete is 5



Deleted node had two children.
Left one becomes deleted node - 3.



Node 6 was deleted here



7 was deleted here

Last node - 17

15 Programming - Heap

New Description

Outcome	Weight
Complexity	♦◊◊◊◊

task

Date	Author	Comment
2020/10/02 11:43	Nebojsa Miletic	Ready to Mark
2020/10/02 11:43	Maksym Slavnenko	Time Exceeded
2020/10/08 14:38	Nebojsa Miletic	Ready to Mark
2020/10/08 14:38	Maksym Slavnenko	Time Exceeded
2020/10/08 17:23	Maksym Slavnenko	What is the complexity of building a heap? What is the complexity of getting a minimum element from a min-heap? If you delete the top element of the heap, what should be done to the heap in order to maintain its properties?
2020/10/08 17:23	Maksym Slavnenko	O(1)
2020/10/08 17:24	Maksym Slavnenko	Re-Hea-Pi-fi-Ca-tion!
2020/10/08 17:25	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Programming - Heap

Submitted By:
Nebojsa Miletic
mileticn
2020/10/08 14:38

Tutor:
Maksym Slavnenko

Outcome	**Weight**
Complexity	♦◊◊◊◊

task

October 8, 2020

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


namespace Heap
{
    public class Heap<K, D> where K : IComparable<K>
    {

        // This is a nested Node class whose purpose is to represent a node of a heap.
        private class Node : IHeapifyable<K, D>
        {
            // The Data field represents a payload.
            public D Data { get; set; }
            // 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 the key of its children.
            public K Key { get; set; }
            // The Position field reflects the location (index) of the node in the array-based internal data structure.
            public int Position { get; set; }

            public Node(K key, D value, int position)
            {
                Data = value;
                Key = key;
                Position = position;
            }

            // This is a ToString() method of the Node class.
            // It prints out a node as a tuple ('key value','payload','index')}.
            public override string ToString()
            {
                return "(" + Key.ToString() + "," + Data.ToString() + "," + Position + ")";
            }
        }

        // ---------------------------------------------------------------------------------
        // Here the description of the methods and attributes of the Heap<K, D> class starts

        public int Count { get; private set; }

        // The data nodes of the Heap<K, D> are stored internally in the List collection.
        // Note that the element with index 0 is a dummy node.
        // The top-most element of the heap returned to the user via Min() is indexed as 1.
        private List<Node> data = new List<Node>();
```

```
        
        // We refer to a given comparer to order elements in the heap.
        // Depending on the comparer, we may get either a binary Min-Heap or a binary  Max-Heap.
        // 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.
        private IComparer<K> comparer;

        // We expect the user to specify the comparer via the given argument.
        public Heap(IComparer<K> comparer)
        {
            this.comparer = comparer;

            // We use a default comparer when the user is unable to provide one.
            // This implies the restriction on type K such as 'where K : IComparable<K>' in the class declaration.
            if (this.comparer == null) this.comparer = Comparer<K>.Default;

            // We simplify the implementation of the Heap<K, D> by creating a dummy node at position 0.
            // This allows to achieve the following property:
            // The children of a node with index i have indices 2*i and 2*i+1 (if they exist).
            data.Add(new Node(default(K), default(D), 0));
        }

        // This method returns the top-most (either a minimum or a maximum) of the heap.
        // It does not delete the element, just returns the node casted to the IHeapifyable<K, D> interface.
        public IHeapifyable<K, D> Min()
        {
            if (Count == 0) throw new InvalidOperationException("The heap is empty.");

            return data[1];
        }

        // Insertion to the Heap<K, D> is based on the private UpHeap() method
        public IHeapifyable<K, D> Insert(K key, D value)
        {
            Count++;
            Node node = new Node(key, value, Count);
            data.Add(node);
            UpHeap(Count);
            return node;
        }

        private void UpHeap(int start)
        {
            int position = start;
            while (position != 1)
```

```
            {
                if (comparer.Compare(data[position].Key, data[position / 2].Key) < 0) Swap(position, position / 2);
                position = position / 2;
            }
        }

        // This method swaps two elements in the list representing the heap.
        // Use it when you need to swap nodes in your solution, e.g. in DownHeap() that you will need to develop.
        private void Swap(int from, int to)
        {
            Node temp = data[from];
            data[from] = data[to];
            data[to] = temp;
            data[to].Position = to;
            data[from].Position = from;
        }

        public void Clear()
        {
            for (int i = 0; i<=Count; i++) data[i].Position = -1;
            data.Clear();
            data.Add(new Node(default(K), default(D), 0));
            Count = 0;
        }

        public override string ToString()
        {
            if (Count == 0) return "[]";
            StringBuilder s = new StringBuilder();
            s.Append("[");
            for (int i = 0; i < Count; i++)
            {
                s.Append(data[i + 1]);
                if (i + 1 < Count) s.Append(",");
            }
            s.Append("]");
            return s.ToString();
        }

        // TODO: Your task is to implement all the remaining methods.
        // Read the instruction carefully, study the code examples from above as they should help you to write the rest of the code.
        public IHeapifyable<K, D> Delete()
        {
            if (Count == 0) throw new InvalidOperationException("The heap is empty.");

            Node answer = data[1];

            Swap(1, Count);
            data.RemoveAt(Count);
```

```

            Count--;
            ReheapDown(1);

            return answer;


        }
        private void ReheapDown(int start)
        {
            int leftChild = start * 2;
            int rightChild = start * 2 + 1;

            if (leftChild <= Count)
            {
                int largest = leftChild;
                if (rightChild < Count && comparer.Compare(data[largest].Key, data[rightChild].Key) > 0)


                    largest = rightChild;

                if (comparer.Compare(data[start].Key, data[largest].Key) > 0)

                    Swap(start, largest);

                    ReheapDown(largest);

            }
        }

        // Builds a minimum binary heap using the specified data according to the bottom-up approach.
        public IHeapifyable<K, D>[] BuildHeap(K[] keys, D[] data)
        {
            if (Count != 0) throw new InvalidOperationException("The heap is empty.");

            Node[] array = new Node[keys.Length];

            for (int i = 0; i < Math.Min(keys.Length,data.Length); i++)
            {
                Count++;
                Node temp = new Node(keys[i], data[i], Count);
                array[i] = temp;
                this.data.Add(temp);
            }

            Heapify();


            return array;
        }
```

```
        private void Heapify()
        {
            int startIndex = Count;

            for (int i = startIndex - 1; i > 0; i--)
            {
                ReheapDown(i);
            }
        }
        public void DecreaseKey(IHeapifyable<K, D> element, K new_key)
        {
            Node selected_node = element as Node;

            if (!data[selected_node.Position].Key.Equals(element.Key)) throw new
            ↪ InvalidOperationException();

            selected_node.Key = new_key;

            UpHeap(element.Position);
        }
        public IHeapifyable<K, D> DeleteElement(IHeapifyable<K, D> element)
        {
            Node newNode = element as Node;

            int nodeIndex = newNode.Position;

            if (!data[nodeIndex].Key.Equals(element.Key) ||
            ↪ !data[nodeIndex].Data.Equals(element.Data)) throw new
            ↪ InvalidOperationException();

            Swap(nodeIndex, Count);

            Count--;

            return element;
        }
        public IHeapifyable<K, D> KthMinElement(int k)
        {
            Node node = new Node(data[1].Key, data[1].Data, 1);

            Heap<K, D> arr = new Heap<K, D>(comparer);

            arr.data = data;

            arr.Count = Count;


            for (int i = 1; i < k; i++)
            {
                arr.Delete();

            }

```

```
            node = (Node)arr.Min();


            arr = null;

            return node;



        }

    }
}
```

16 Problem Solving: Graphs

Problem Solving: Graphs

Outcome	**Weight**
Complexity	♦◊◊◊◊

Task

Date	**Author**	**Comment**
2020/09/30 19:12	Nebojsa Miletic	need more time
2020/10/02 17:10	Nebojsa Miletic	Ready to Mark
2020/10/08 17:25	Maksym Slavnenko	How many times do we need to run Dijkstra for that particular task?Is it possible to have negative edges in that particular task? Does the number of edges ever reach Vˆ2 for that particular task? Can you think of a case when the number of edges is more than 2 between neighbouring nodes? (that's an advanced question, it's ok if you can't)
2020/10/08 17:25	Maksym Slavnenko	Dijskstra's complexity is incorrect in the task sheet, please use the following: 'O(E+V*log(V))' - Dijkstra complexity.'E=Vˆ2' -> 'O(Vˆ2+V*Log(V))' and do it V times. -> 'O(Vˆ3 +Vˆ2*log(V))'
2020/10/08 17:25	Maksym Slavnenko	q1 and q3 are correct
2020/10/08 17:26	Maksym Slavnenko	Fix and Resubmit
2020/10/12 12:21	Nebojsa Miletic	Ready to Mark
2020/10/12 12:21	Maksym Slavnenko	Time Exceeded
2020/10/12 18:14	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Problem Solving: Graphs

Submitted By:
Nebojsa Miletic
mileticn
2020/10/12 12:21

Tutor:
Maksym Slavnenko

Outcome	**Weight**
Complexity	♦◇◇◇◇

Task

October 12, 2020

Task 9.1

Q1.

According to lecture slides and the book “Data Structures and Algorithms in Java”, the running time of depth-first algorithm using adjacency list is O(n+m), where n is number of vertices and m is number of edges.

Adjacency list is actually doubly linked list that stores its outgoing neighbours. Advantage of using this data structure is that insertion of edges goes in constant time and it is well suited for sparse graphs.

Also, using adjacency matrix is another solution. Idea is to store values in matrix or two dimensional list, where vertices are integers in the set { 0, 1, 2 … n - 1} and edges as a pair of such integers. This allow us to store references to edges in two-dimensional n x n array. Insertion, removal and edge queries work in constant time. Big – O(n) is to obtain an edge or leaving a node. However storage requirement are far more worse, with O(n^2). Time complexity for depth - first - search is O(n^2), because in matrix we have n x n nodes and every node is visited once.

Q2.

For solving this problem, it seems that Floyd algorithm is much more suitable, in spite having worse running time O (V^3). Floyd is more efficient in this case, because it searches for shortest distance between all pairs of nodes, while Dijkstra is used just for single pair of nodes. That means that if tourist wants to find out the shortest path with Dijkstra, he would need to run the algorithm every time he wants to go different places. That makes Floyd better choice in this case.

It would not be possible to have negative edges in this particular task, unless there are cases that can be updated in the app that tourist uses, such as that some paths to tourist attractions are inaccessible, let’s say they are flooded or there are some roadworks, etc, but that was not indicated in the task description. That would make using Dijkstra implausible.

It looks like that the tourist would be much better off running algorithm which would calculate shortest path in one go. On the other hand, if tourist wants specifically to calculate the best way for just two places, such as his hotel and nearest museum, Dijkstra would be better solution.

Does the number of edges ever reach V^2 for this particular task?

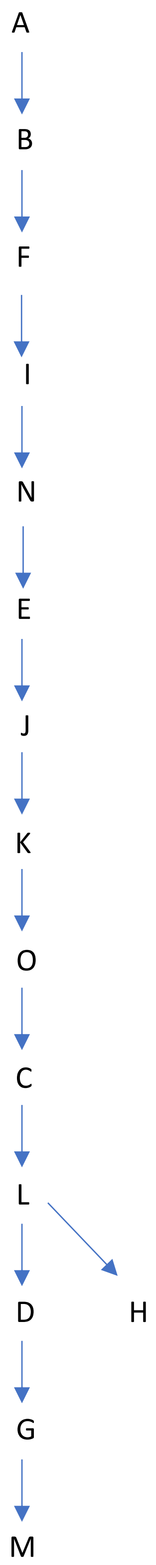
Dijkstra complexity: O(E + V * log (V)) implies that for calculating running time for Dijkstra depends on both of number of edges and number of vertices.
It also depends of the data structure used to store the data, as we saw in question 1 above.

If we use matrices to store the vertices, we would get V^2. Also, if the problem is represented as a sparse graph, it would never reach V^2,whereas if it was represented as a dense graph there would be a possibility to get number of vertices V^2.

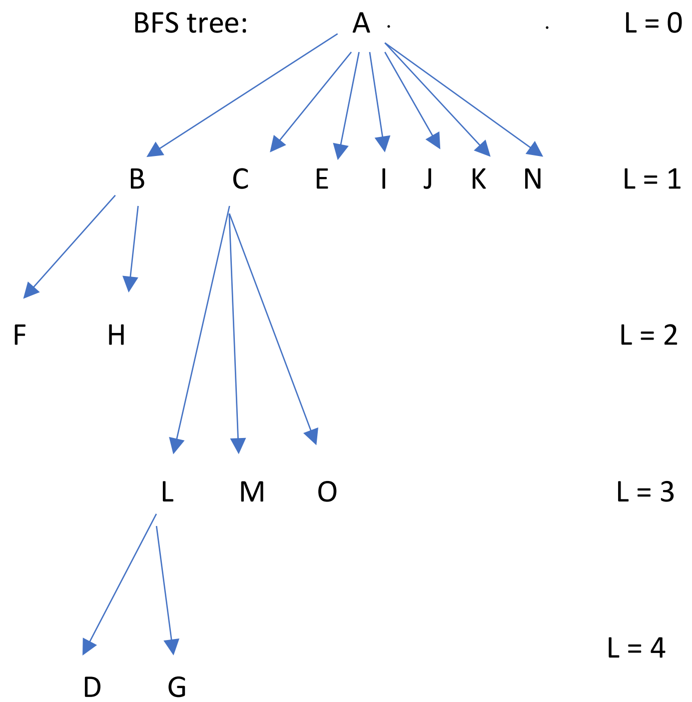
On the other hand, Floyd's algorithm puts the vertices in matrix data structure and use three for loops to find shortest path between the vertices, and that is why it gets V^3 worst case

If there is a case where there are more than one edge between two nodes, the Dijkstra and Floyd implementation should cover that case such as the edge with the lower weight should be chosen, and other thrown away. Same counts for edges with zero value, as they would cause problems with finding shortest path

DFS tree:



BFS tree:



17 Quiz - Final two weeks

New Description

Outcome	Weight
Document solutions	♦◊◊◊◊

task

Date	Author	Comment
2020/10/03 10:20	Nebojsa Miletic	Need more time
2020/10/09 11:00	Nebojsa Miletic	Ready to Mark
2020/10/09 14:57	Maksym Slavnenko	Complete

Deakin University

Data Structures and Algorithms

OnTrack Submission

Quiz - Final two weeks

Submitted By:
Nebojsa Miletic
mileticn
2020/10/09 11:00

Tutor:
Maksym Slavnenko

Outcome	Weight
Document solutions	♦◊◊◊◊

task

October 9, 2020

Task 11.1

Zoom recording: https://youtu.be/NRrPdGkQnpo

Screen recording: https://youtu.be/t_RbCKFrEDw