RMIT UNIVERSITY

Algorithms and Analysis
COSC 2123/1285
Assignment 1: Binary Space Partitioning (BSP)

	Assessment Type	Pair (Group of 2) Assignment.
	Due Date	Week 7, 11:59pm, 11^{th} of Sep, 2019
	Marks	15

Please read all the following information before attempting your assignment. This is a *group* assignment. You may not collude with any other people outside of your group, or plagiarise their work. Each group is expected to present the results of their own thinking and writing. Never copy other student's work (even if they "explain it to you first") and never give your written work to others. Keep any conversation high-level and never show your solution to others. Never copy from the Web or any other resource. Remember you are meant to generate the solution to the questions by yourself (yourselves). Suspected collusion or plagiarism will be dealt with according to RMIT policy.

In the submission (your report) you will be required to certify that the submitted solution *represents your own work only* by agreeing to the following statement:

> *We (I) certify that this is all our (my) own original work. If we (I) took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. We (I) will show we (I) agree to this honor code by typing "Yes":*

1 Overview

You may hear of some famous games (e.g. Doom and Quake), but did you know they had used a technique called Binary Space Partitioning (BSP) to speed up visibility calculations in 3D Rendering? Visibility calculations means "only visible walls and objects must be drawn, and they must be drawn in the right order (close walls should be drawn in front of far away walls)".

In computer science, BSP is a technique that recursively subdivides a space into two convex sets via hyperplanes as partitions. It generates a representation of objects within the space, which is in the form of a *tree* data structure[1].

[1] https://en.wikipedia.org/wiki/Binary_space_partitioning

To render maps in a game, the visibility calculation needs to be performed so that it can be used many times later. Note, for assignment purposes, we only consider static maps here. These calculations will be stored in a tree structure, and will be used in the game.

Briefly, BSP divides the map into many convex polygons, which is a polygon where all its internal angles ≤ 180 degrees. Fig. 1 shows an example of convex and non-convex polygon.

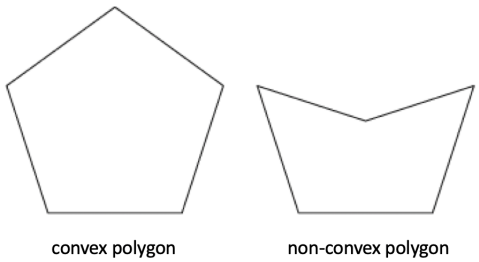


Figure 1: Examples of convex and non-convex polygon

For example, Fig. 2-1 is a given map, which is considered a non-convex polygon. Then, we can draw a line to divide it into two sub-polygons, as shown in Fig. 2-2. In this step, two sub-polygons are created. The corresponding tree structure for this operation is shown at the bottom of Fig. 2-2. BSP keeps dividing these two sub-polygons recursively until the entire map is divided into convex polygons. During this process, each division generates a new branch in the tree structure. Finally, the leaves of the tree are the convex polygons.

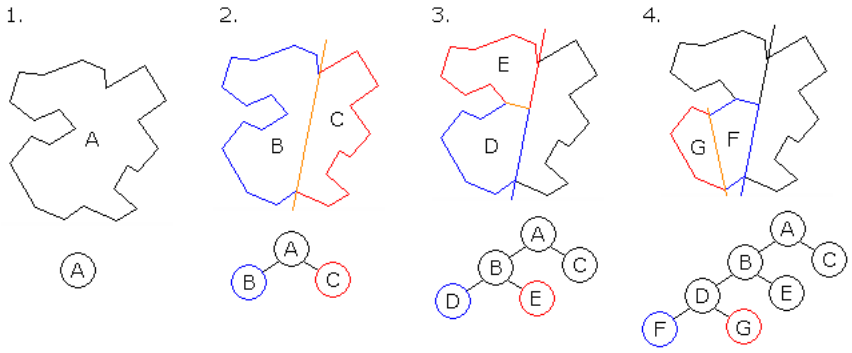


Figure 2: Example of BSP [2]

The core of the visibility ordering system lies in the order in which the rendering function recurses. That is, whether the left or right subtree of the given node is rendered first. For any particular node, there is a dividing line where it splits into two subnodes. If this line is extended to infinity, the viewpoint from which we are rendering can be considered to be on either the "left" or "right" side. The side viewpoint is on determines which of the subnodes is rendered first.

[2]The picture in `https://en.wikipedia.org/wiki/Binary_space_partitioning`

Rendering using BSP trees is also done using a recursive algorithm. The most common approach is to start at the root node (the top of the tree) and work down recursively. This is why it is desirable to make sure the efficiency of doing operations on this tree.

In class, we studied two methods to represent the tree,

- the Sequential Representation, and
- the Linked Representation.

The performance of each representation varies depending on the characteristics of the tree.

In this assignment, we will implement both representations, and evaluate how well they perform when representing some given Binary Space Partitioning Trees and computing the average speed of visiting nodes in the tree.

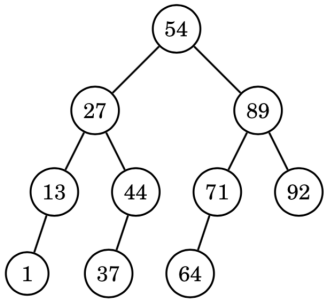
2 Tasks

The assignment is broken up into a number of tasks, to help you progressively complete the project.

Task A: Implement the Tree Representations and their Operations (5 marks)

In this task, you will implement the tree using *Sequential Representation* and *Linked Representation*. Each representation will be implemented by a data structure. Your implementation should support the following operations:

- Create an empty tree (implemented as a constructor that takes zero arguments).
- Set a root node for the tree.
- Split a node into two children nodes.
- Find a node in the tree.
- Find a parent node of a given node.
- Find children nodes of a given node.
- Print out all the nodes in a certain order, including preorder, inorder, and postorder. This is actually Tree-traversal, which refers to the process of recursively visiting (examining and/or updating) each node in a tree data structure, exactly once, in a *systematic* way. Such traversals are classified by the **order** in which the nodes are visited. Starting at the **root** of a binary tree, there are three main steps that can be performed and the order in which they are performed defines the traversal type. These steps are: performing an action on the current node (referred to as "visiting" the node), traversing to the left child node, and traversing to the right child node. Although we didn't go through this in lectures, it is interesting to know about traversals. Consider the following traversal approaches and example as shown in Fig. 3.



Pre: 54,27,13,1,44,37,89,71,64,92
In: 1,13,27,37,44,54,64,71,89,92
Post: 1,13,37,44,27,64,71,92,89,54

Traversal methods

In **preorder** traversal, the root is visited **before** the left and right subtrees are visited:

1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

In **inorder** traversal, the root is visited **after** the left subtree and **before** the right subtree is visited:

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

In **postorder** traversal, the root is visited **after** the left and right subtrees are visited:

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.

Figure 3: Examples of tree-traversal methods

Data Structure Details

Trees can be implemented using a number of data structures. You are to implement the tree abstract data type using the following data structures:

- Sequential Representation, using a 1D array.
- Linked Representation, using a linked list.

For linked list, you must program your own implementation, and not use existing libraries in any kind, e.g. the LinkedList, Set, MultiSet or Tree type of data structures in java.utils or any other libraries. You must implement your own nodes and methods to handle the operations. If you use java.utils or other implementation from libraries, this will be considered as an invalid implementation and attract 0 marks for that data structure.

Operations Details

Operations to perform on the implemented tree abstract data type are specified on the command line. They are in the following format:

```
<operation> [arguments]
```

where `operation` is one of `{RN, SP, FN, FP, FC, TP, TI, TS, Q}` and `arguments` is for optional arguments of some of the operations. The operations take the following form:

- `RN <nodeLabel>` – sets a root node with label '`nodeLabel`' into an empty tree. There should have no output for this operation.
- `SP <nodeLabel> <leftChild> <rightChild>` – splits the given node into two children nodes like the operations in Fig. 2-2. There should have no output for this operation.
- `FN <nodeLabel>` – finds a node with label '`nodeLabel`' in the tree. The output should be `true` if find it, otherwise `false`.
- `FP <nodeLabel>` – finds the parent node of the given node with label '`nodeLabel`' in the tree. The format of the output for node '`B`' should take the form:

  `B <the label of B's parent node>`,
  e.g., '`B A`' or '`B `' if node `B` has no parent.
- `FC <nodeLabel>` – finds the children nodes of the given node with label '`nodeLabel`' in the tree. The format of the output for node '`A`' should take the form:

  `A <the label of A's left child node> <the label of A's right child node>`, e.g., "`A B C`"
- `TP` – prints all the nodes in the *preorder* traversal. The print operation outputs the nodes in the tree in a single line. The line should specify all the valid nodes (note labels) in the tree.

  `<node1> <node2> <node3> ...`
- `TI` – prints all the nodes in the *inorder* traversal. The same output format like `TP`.
- `TS` – prints all the nodes in the *postorder* traversal. The same output format like `TP`.
- `Q` – quits the program.

Refer to the javaDoc of `BSPTree.java` for more implementation requirements. As an example of the operations, consider the output from the following list of operations:

```
RN A
SP A B C
SP B D E
SP C F G
FN C
FN H
FP F
FP A
FC B
FC C
FC G
TP
Q
```

The output should be:

```
true
false
F C
A
B D E
C F G
G
A B D E C F G
```

Testing Framework

We provide Java skeleton code (see Table 1) to help you get started and automate the correctness testing. You may add your own Java files to your final submission, but please ensure that they work with the supplied Python testing script (see below).

file	**description**
`TreeTester.java`	Code that reads in operation commands from `stdin` then executes those on the selected tree implementation. *Do not modify this file*.
`BSPTree.java`	Interface for BSP trees. All implementations should implement the BSPTree class defined in this file. *Read the javaDoc of each method carefully and do not modify this file*.
`SequentialRepresentation.java`	Code that implements the sequential representation of a tree. Complete the implementation (implement parts labelled "**Implement me!**").
`LinkedRepresentation.java`	Code that implements the linked representation of a tree. Complete the implementation (implement parts labelled "**Implement me!**").

Table 1: Table of Java files.

In addition, we provide a Python script that automates testing, based on input files of operations (such as example above). A detailed instruction about how to use this Python script is included at the top of the python file itself. These are fed into the Java framework which calls your implementations. The outputs resulting from any print operations are stored, then compared with the expected output. We have provided two sample input and expected files for your testing and examination.

For our evaluation of the correctness of your implementation, we will use the same Python script and input/expected files that are in the same format as the provided examples. To avoid unexpected failures, please *do not* change the Python script nor `TreeTester.java`. If you wish to use the script for your timing evaluation, make a copy

and use the unaltered script to test the correctness of your implementations, and modify the copy for your timing evaluation. Same suggestion applies for `TreeTester.java`.

As the instructions for the assignment are getting too lengthy, instructions on how the python script runs are available within the header of the script.

Notes

- Use the output of the provided *sample* implementation (sample is the actual name of the implementation) to help you determine the right output format for the operations. If you correctly implement the "Implement me!" parts, you in fact do not need to do anything else to get the correct output formatting. `TreeTester.java` will handle this.
- We will run the supplied test script on your implementation on one of the following university's core teaching servers:
  - `titan.csit.rmit.edu.au`
  - `jupiter.csit.rmit.edu.au`
  - `saturn.csit.rmit.edu.au`

  If you develop on your own machines, please ensure your code compiles and runs on these servers. You don't want to find out last minute that your code doesn't compile on these servers. **If your code doesn't run on these servers, we unfortunately do not have the resources to debug each one and cannot award marks for testing.**
- All submissions should compile with no warnings on **Oracle Java 8**.

Test Data

We provide a Binary Space Partition example of about 4000 nodes in a file called "BSP_combined.txt". This is real, sampled part of a game map.

Task B: Evaluate your Data Structures (5 marks)

In this second task, you will evaluate your two implemented structures in terms of their time complexities for the different operations and different use case scenarios. Scenarios arise from the possible BSP usage.

Write a report on your analysis and evaluation of the different implementations. Consider and recommend in which scenarios each type of implementation would be most appropriate. The report should be **6 pages or less**, in font size 12. See the assessment rubric (Appendix A) for the criteria we are seeking in the report.

Use Case Scenarios

Typically, you use real usage data to evaluate your data structures. However, for this assignment, you will write data generators to enable testing over different scenarios of interest. There are many possibilities, but for this assignment, consider the following

scenarios:

Scenario 1 Growing BSP Tree (Additions): In this scenario, the BSP tree is growing while a map is being divided into smaller new polygons (nodes). In this scenario, you are to evaluate the performance of your implementations in terms of:

- node addition;
- node splitting;

Assume the tree that you start with is the BSP_combined.txt one. You are to evaluate the performance the node addition and splitting operations as the complexity of the initial tree is varied.

Scenario 2 Finding a Node and Its Parent Node and Children Nodes: In this scenario, the tree is not changing, but important operations such as Finding the Node and/or its parent node and/or children nodes are required.

Assume the tree that you start with is the BSP_combined.txt one. You are to evaluate the performance of the finding node and its parent/children node(s) implementations as the complexity of the initial tree is varied.

Scenario 3 Printing the Nodes (Traversal):

This is to evaluate the performance of your implementation in terms of:

- `preorder` traversal
- `inorder` traversal
- `postorder` traversal

Assume the tree that you start with is the one that we provided you with. You are to evaluate the performance the traversal operations as the complexity of the initial tree is varied.

Analysis

In your analysis, you should evaluate each of your representations and data structures in terms of the different scenarios outlined above. For generating tree with different initial complexities, you may want to either generate a series of split node operations ('SP') to grow the tree to the desired complexity, then evaluate for the appropriate scenario. Alternatively, you can consider writing a data generator within Java to insert directly into the data structures. Whichever method you decide to use, remember to generate tree of different complexities to evaluate on (Specifically, this means that you need to generate tress with different sizes, e.g. trees with various the depth or the number of nodes. For example, the number of nodes can be 10^2, 10^3 or 10^6). Due to the randomness of the data, you may wish to generate a few datasets with the same parameters settings (same tree complexity and a scenario) and take the average across a number of runs.

Note, you may be generating and evaluating a significant number of datasets, hence we advise you to get started on this part relatively early.

Task C: Theoretical Complexity Analysis (5 Marks)

Question C1. Complexities of tree operations [3 mark] Fill in the following table with YES/NO answers to indicate whether the corresponding tree operations belong to certain complexity classes in the *worst case*, and explain why. Here n is the number of nodes in the tree implementation for the BSP. Please note answers without explanations will attract zero marks.

	$O(1)$	$O(\log(n))$	$O(n)$	$O(n!)$
Find parent				
Find a node				
Print all nodes				

Question C2. Guessing the age of an alien [2 mark] An astronaut, after an emergency landing on a remote planet, was faced by an alien. Using his portable universal language translator, the desperate astronaut asked the alien for water and food. Luckily, the alien agreed to help, with one condition: the astronaut must be able to guess correctly its age by asking just YES/NO questions before he died of hunger. As there was no way to tell how old the planet or the alien is, the astronaut had to assume that the alien was of any age between 1 to N years old, where N is a very large number. For instance, to be on the safe side, N is around 14 billion, which is the estimated age of the universe. Now, let's consider two strategies for asking the questions.

1. (Strategy 1) The astronaut asked the questions "Are you n years old?" for every $n = 1, 2, \ldots, N$ in any order. If the alien answered YES, the astronaut won the supplies. If not, he continued his questions. What are the number of questions he needed to ask before getting his food in the worst case? Same question for the average case, assuming that every age between 1 and N are equally likely with probability $1/N$? If $N = 14$ billion and that each question and answer took just one second to complete, roughly how long would it take the astronaut before he can have a drink in the worst case and in the average case?

2. (Strategy 2) The astronaut asked a series of questions "Are you at most X years old?" and represented them as nodes in a binary tree. He could start off at Node 1 by asking "Are you at most X_1 years old?". If YES, he create Node 2, which is a left-child of Node 1 and continues asking "Are you at most X_2 years old?", while if NO, he create Node 3, which is a right-child of Node 1 and continues asking "Are you at most X_3 years old?", and so forth. Using the best design of such a tree to minimize the number of questions asked, how many questions do you think the astronaut needed to ask in the worst case and in the average case? If $N = 14$ billion, how many questions did he need in both cases?

3 Report Structure

As a guide, the report could contain the following sections:

- Explain your data generation and experimental setup. Things to include are (brief) explanations of the generated data you decide to evaluate on, the complexity parameters you tested on, describe how the scenarios were generated (a paragraph and perhaps a figure or high level pseudo code suffice), which approach(es) you decide to use for measuring the timing results, and briefly describe the fixed set(s) you used to generate the elements for node addition.
- Evaluation of the data structures using the generated data. Analyse, compare and discuss your results across different densities, representations and scenarios. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each data structure to help in your explanation.
- Summarise your analysis as recommendations, e.g., for this certain data scenario of this complexity, I recommend using this data structure because... We suggest you refer to your previous analysis to help.
- Detail your solutions to Task C.

4 Submission

The final submission will consist of three parts:

- Your **Java source code** of your implementations. Your source code should be placed into in a flat structure, i.e., all the files should be in the same directory/-folder, and that directory/folder should be named as `Assign1-<partner 1 student number>-<partner 2 student number>`. Specifically, if your student numbers are s12345 and s67890, then all the source code files should be in folder Assign1-s12345-s67890.
- All folder (and files within) should be zipped up and named as `Assign1-<partner 1 student number>-<partner 2 student number>.zip`. E.g., if your student numbers are s12345 and s67890, then your submission file should be called `Assign1-s12345-s67890.zip`, and when we unzip that zip file, then all the submission files should be in the folder Assign1-s12345-s67890.
- Your **written report for part B and C** in PDF format, called "assign1.pdf". Place this pdf within the Java source file directory/folder, e.g., Assign1-s12345-s67890.
- Your **data generation code**. Create a sub-directory/sub-folder called "generation" within the Java source file directory/folder. Place your generation code within that folder. We will not run the code, but will examine their contents.
- Your group's **contribution sheet**. See the following 'Team Structure' section for more details. This sheet should also be placed within your source file folder.

Note: **submission of the report and code will be done via Canvas**. More detailed instructions will be provided closer to submission date.

5 Assessment

The project will be marked out of 15. Late submissions will incur a deduction of 1.5 marks per day, and no submissions will be accepted 7 days beyond the due date.

The assessment in this project will be broken down into two parts. The following criteria will be considered when allocating marks.

5.1 Implementation (5/15)

- You implementations will be assessed on whether they are sequential and linked representation, respectively, and on the number of tests it passes in our automated testing.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence these factors will contribute towards your marks.

5.1.1 Checkpoint (1 mark)

As part of the implementation work and to help you progress, you are asked to demo your implementation of the *Sequential* representation and its operations as listed in task A during the lab session of week 5 (Lab 04). One member of the group should at least attend your allocated lab, where your Lab Assistant will assess your progress on the implementations. We will run the provided test script on some inputs to test if your *Sequential* representation compiles, runs and passes a number of basic tests. This does not guarantee your code is bug free and will pass all of our final tests. Please conduct your own further testing.

5.2 Report (10/15)

The marking sheet in Appendix A outlines the criteria that will be used to guide the marking of your empirical evaluation report. Use the criteria and the suggested report structure (Section 4) to inform you of how to write the report.

Task C's solution should also be included in the report (5 marks).

6 Team Structure

This project should be done in **pairs** (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. If there are issues with work division and workload in your group, please contact your lecture as soon as possible.

In addition, please submit what percentage each partner made to the assignment by filling the sheet named `Assign1ContributionSheet.docx`, and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution X%, and student B has contribution Y%, and $X > Y$. The

group is given a group mark of M. Student A will get M for assignment 1, but student B will get $\frac{M}{\frac{X}{Y}}$.

7 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted project work in this subject is to be the work of you and your partner. It should not be shared with other groups. **Multiple automated similarity checking software will be used to compare submissions**. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the students concerned. Plagiarism of any form may result in zero marks being given for this assessment and result in disciplinary action.

For more details, please see the policy at
`http://www1.rmit.edu.au/students/academic-integrity`.

8 Getting Help

There are multiple venues to get help. There are weekly consultation hours with the Lecturer and the Head Tutor(see Canvas for time and location details).

In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Assistant. We will also be posting common questions on the assignment's Q&A section in Canvas. You are encouraged to check and participate in the discussion forum on Canvas. However, please **refrain from posting solutions**.

A Marking Guide for the Report and Empirical Analysis

Design of Evaluation (Maximum = 1 marks)	**Empirical Analysis of Results** (Maximum = 3 marks)	**Report** (Maximum = 1 marks)
1 marks Data generation is well designed, systematic and well explained. All suggested scenarios, data structures and a reasonable range of densities were evaluated. Each type of test was run over a number of runs and results were averaged.	3 marks Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures, scenarios and densities. All analysis, comparisons and conclusions are supported by empirical evidence and possibly theoretical complexities. Well reasoned recommendations are given.	1 marks Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.
0.5 marks Data generation is reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures or reasonable densities. Each type of test was run over a number of runs and results were averaged.	2 marks Analysis is reasonable and demonstrates good understanding and critical analysis. Adequate comparisons and explanations are made and illustrated with most of the suggested scenarios and densities. Most analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Reasonable recommendations are given.	0.5 marks Clear and structured for the most part, with a few unclear minor sections.
0.25 mark Data generation is somewhat adequately designed, systematic and explained. There are several obvious missing suggested scenarios, data structures or reasonable densities. Each type of test may only have been run once.	1 marks Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios and densities. A portion of analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Adequate recommendations are given.	0.25 mark Generally clear and well structured, but there are notable gaps and/or unclear sections.
0 marks Data generation is poorly designed, systematic and explained. There are many obvious missing suggested scenarios, data structures or reasonable densities. Each type of test has only have been run once.	0.5 mark Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario and density setting. Little analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Poor or no recommendations are given.	0 marks The report is unclear on the whole and the reader has to work hard to understand.