COSC 2123/1285 Algorithms and Analysis
Assignment 1: Nearest Neighbour

	Assessment Type	Pair (Group of 2) Assignment. Submit online via Canvas → Assignments → Assignment 1. Clarifications/updates/FAQ can be found in Canvas Announcements and Discussion → Assignment 1 General Discussion.
	Due Date	Week 7, 11:59pm, Friday, September 10, 2021
	Marks	30

1 Objectives

There are a number of key objectives for this assignment:

- Understand how a real-world problem can be implemented by different data structures and/or algorithms.
- Evaluate and contrast the performance of the data structures and/or algorithms with respect to different usage scenarios and input data.

In this assignment, we focus on the nearest neighbour problem.

2 Background

Given a set of points, the nearest neighbour problem is about finding the nearest points to a query point. It appears in many different applications. One of these is facility search, e.g., querying what are the nearest restaurants or parks on your mobile phone. Another is identifying automatically recognising hand written digits on envelopes. In this assignment, we will focus on nearest neighbour problem for a spatial context, implement several well known data structures and algorithms for finding the nearest neighbours and comparing their performance. One of these data structures is the Kd-tree, which is a specialised binary tree and described in the following.

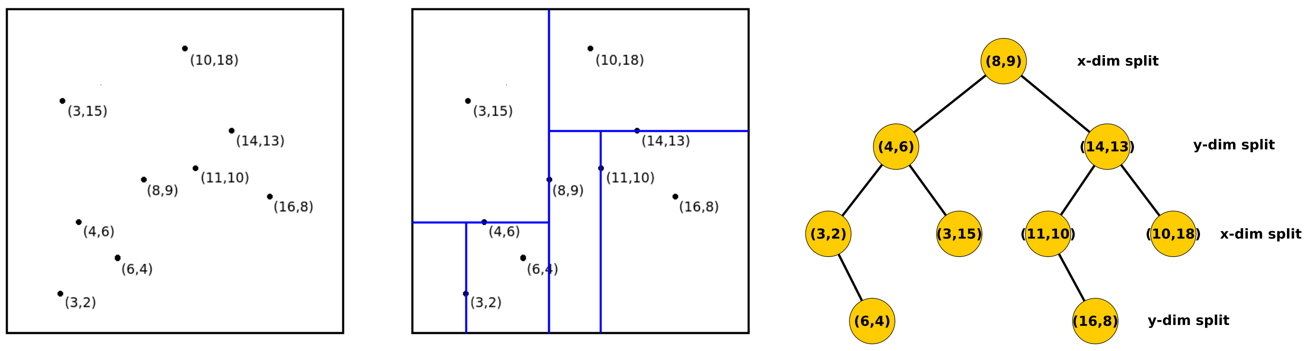
Kd-Trees

Kd-trees is a binary tree data structure that enables more efficient nearest neighbour searches for spatial queries. It builds an index to quickly search for nearest neighbours. Based on the set of points to index, Kd-trees recursively partition a multi-dimensional space, and generally ensures a balanced binary tree (making search relatively fast). In this assignment we focus on 2D spaces (something we are familar with and can be visualise). In the 2D space, we have x (horizonal) and y (vertical) dimensions.

Construction:

Given a set of points in a 2D space, first we find the point that is the median in the x-axis dimension. We draw a vertical line through this median point. Half the points are to the left of this line, and the other half of points are on the right. For each of the halves, we find the median of the points in those halves in the y-axis dimension, and draw a horizontal line through it. Points above this line form one partition, points in the other form another. This process continues for each partition until all the partitions either contains a single point or are empty. The single point partitions form the leaves of our tree, while empty partitions form null nodes.

As an example, consider Figure 1. Initially we find the median point in the x-dimension, which is (8,9), and partition the remaining points to those that are to the left of it, and thoset that are to the right of it. Those left of it form the left subtree (points (3,2), (3,15), (4,6), (6,4)) and the other points the right subtree. For the points in the left subtree, we find the median in y-dimension - could be (6,4) or (4,6), but we select (4,6). We split the points in the y-dimension around (4,6), with those points whose y-coordinates is less than 6 forming its left subtree ((3,2) and (6,4)) while (3,15) forms the right subtree.



(a) Points. (b) Kd-tree partition of 2D space. (c) Kd-Tree.

Figure 1: Points of 2D space and constructed Kd-tree for the points.

Searching:

Given a query, we wish to find which point(s) are closest to it. The searching algorithm basically searches through the tree.

1. Recall for each node, it represents a split in either the x or y dimension. E.g., in Figure 1c, the root node (8.9) is a split on the x dimension and node (4,6) is a split on the y dimension.

2. The search process starts from the root node. The search algorithm moves down the tree recursively, and moves left or right depending on the current visited node, which dimension it is spliting, and the query point. E.g., say the query is (16,14). At the root node, which is split on the x dimension, we compare the x coodinates - is 16 > 8? Yes, so we go to right branch. At node (14,13), it is splitting on y dimension, so we check if 14 > 13, yes, so we go down the right sub-tree.

3. When the search algorithm reaches a leaf node, we stop the recursion and save that node as the current closest point to the query.

4. The search algorithm unwinds/tracks back the recursion of the tree. It is basically searching if there are other predecessor nodes/points that are closer than the current closest one. For each predecessor node that the search algorithm tracks back up:
   - If the current node/point is closer than the saved closest one, the current node is saved as the new closest one.
   - The search algorithm needs to check if there are points on the other side of the splitting line (of the current node) that could be closer to the query point. This is the other subtree that we haven't visited in the kd-tree. This occurs when the distance from query point to current closest point is greater than smallest distance from query point to the splitting line. Hence, if it is the case, then the search algorithm must perform search down this unvisited sub-tree branch, then rewind back up this subtree, doing the closest point checks. It it isn't the case, then the other sub-tree branch cannot hold the closest point and can be eliminated from consideration.
   - When the search algorithm reaches the root node, then we have finished and the current closest point is the closest/nearest neighbour to the query point.

To search for k-nearest neighbours, we keep k closest points (instead of one), and only eliminate a sub-tree branch if the points in that branch cannot be closer than any of the current k closest points.

Addition

When adding a point to the Kd-tree, it is the similar to the construction. First, we start at the root node, and determine which dimension the root node is splitting on, and then whether to go left or right subtrees, based on the point to be inserted. This is recursively repeated, until we reach a leaf node, then we determine whether to add to the left or right child.

Deletion

Deletion is similar to binary trees, but can be a little confusing, so will be explained separately in another note/video (later).

3 Tasks

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

Task A: Implement the Nearest Neighbour Data Structures and their Operations (12 marks)

In this task, you will implement algorithms and data structures for k-nearest neighbour searches, using a *naive* and *Kd-tree* based approaches. Your implementation of each approach should support the following operations:

- Create an empty algorithm/data structure (implemented as a constructor that takes zero arguments).
- Search for the k-nearest neighbours of a query of coordinates and category (k is at least 1).
- Add a point to the set of points.
- Remove a point from the set of points.

Implementation Details

Naive Implementation In this subtask, you will implement a naive implementation for nearest neighbour searches. In the naive implementation, no data structure is used to assist searches. Each time there is a neighbour neighbour query, the naive algorithm will compute all pairwise distances from query, then choose the (k) nearest ones.

KD-Trees Implementation

In this subtask, you will implement the Kd-tree and use it to answer nearest neighbour searches.

Operations Details

Operations to perform on the implementations of the nearest neighbour search are specified on the command file. They are in the following format:

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

where operation is one of {S, A, D, C} and arguments is for optional arguments of some of the operations. The operations take the following form:

- `S <category> <x-coordindates> <y-coordinates> <k>` – searches for the k nearest neighbours to query (x,y), for the specified category. If no nearest neighbours, return empty list.
- `A <id> <category> <x-coordinates> <y-coordinates>` – adds a new point (x,y) to the point set of specified category. If point exists already (id, category, both coordinates all match), return false.
- `D <id> <category> <x-coordinates> <y-coordinates>` – delete a point (x,y) with id from the point set of specified category. Id, category and coordinates must match for delete to proceed. If fail to delete, return false.
- `C <id> <category> <x-coordinates> <y-coordinates>` – check if a point (x,y) is in the point set of specified category. Id, category and coordinates must match for check to be success.

The format of the output of a search operation for (category, x, y) should return one neighbour per line. For each neighbour, it will be in the form:

```
Point{id=IdX, cat=category, lat=x, lon=y}
```

where id is the identifier, cat is the category, lat is latitude (x-coodinate) and lon is the longitude (y-coordinate). In the supplied code, we use latitude and longitude because the context and application is geographical based. While not technically 100% correct, for simpler understanding you can consider latitute to correspond to x-coordinate and longitude as y-coordinate. If the category has no locations, then the nearest neighbour list should be empty. If less than k points in category, e.g., k = 10, but only 6 points, return all 6 points.

As an example of the operations, consider the input and output from the provided testing files, e.g., test1.in and the expected output, test1.out.

Note, you do not have to do the input and output reading yourself. Use the provided Java files, they will handle the necessary input and output formats. Your task is to implement the missing functionality in the NN classes.

Testing Framework

We provide Java (see Table 1) and Python skeleton codes to help you get started and automate the correctness testing. You may add your own Java/Python files to your final submission, but please ensure that they work with the supplied Python testing script (see below). Below we describe code structure in Java. The Python version is similar.

file	**description**
`NearestNeighFileBased.java`	Code that reads in operation commands from file then executes those on the specified nearest neighbour data structure. *No need to modify this file.*
`Category.java`	Class defining the different categories of queries. *No need to modify this file.*
`NearestNeighbour.java`	Interface for the two query agents. *No need to modify this file.*
`Point.java`	Class representing a point in 2D space. *No need to modify this file.*
`NaiveNN.java`	Skeleton code that implements a naive implementation of nearest neighbour searches. Complete the implementation (implement parts labelled "To be implemented").
`KDTreeNN.java`	Skeleton code that implements Kd-tree implementation of nearest neighbour searches. Complete the implementation (implement parts labelled "To be implemented").

Table 1: Table of Java files.

To compile, from the directory where NearestNeighFileBased.java is, execute:

```
> javac *.java
```

To run, from the directory where NearestNeighFileBased.java is, execute:

```
> java NearestNeighFileBased [approach] [data filename] [command filename] [output filename]
```

where

- approach is one of "naive" or "kdtree";
- data filename is the name of the file containing the intial set of points;
- command filename is the name of the file with the commands/operations;
- output filename is where to store the output of program.

In addition, we provide a Python script that automates testing, based on input files of operations (such as example above). These are fed into the Java framework which calls your implementations. The outputs resulting from search and check operations are stored, as well as error messages from the operations. The outputs are 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 NearestNeighFileBased.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 NearestNeighFileBased.java.

Notes

- If you correctly implement the "To be implemented" parts, you in fact do not need to do anything else to get the correct output formatting. NearestNeighFileBased.java will handle this.
- We will run the supplied test script on your implementation on the university's core teaching servers, e.g., `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 machines. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, 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 1.8**.

Test Data

We provide a sample set of restaurants, hospitals and education (schools) from Melbourne and surrounding area. It contains 1240 points. It is stored in file sampleData.txt. Each line represents a point. For each point/line, the format is as follows:

```
id category latitude longitude
```

where id is the unique identifier for that point, category is the category, and latitude (x-coordinate) and longitude (y-coordinate) are the coordinates of the point. This is also the expected input file format when passing data information to NearestNeighFileBased.java. When creating your own files for the initial set of points, use this format.

Task B: Evaluate your Data Structures for Nearest Neighbour Queries (18 marks)

In this second task, you will evaluate your two implementions in terms of their time complexities for the different operations and different use case scenarios. Scenarios arise from the possible use cases of nearest neighbour searches.

Write a report on your analysis and evaluation of the different implementations. Consider and recommend in which scenarios each approach would be most appropriate. The report should be no more than **5 pages**, in font size 12 and A4 pages. 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 k-nearest neighbour searches: In this scenario, we evaluate how the algorithms perform as the k nearest neighbour parameter is varied for searching. As this might be affected by the number of points in the dataset, construct three datasets with different number of points, then vary the number of searches and k on these.

Scenario 2 Dynamic points set: In this scenario, the points are constantly added and deleted (e.g., nearest pubic transport). We wish to evaluate the performance of the Kd-tree, and how it compares to the naive algorithm as we add/delete more and more points. For this scenario, construct a few datasets with different initial number of points. Randomly add, and delete the same number of points (i.e, the total number of points added and deleted should be the same), and then perform searches on the modified Kd-tree structure.

The points generated to add, delete, or to find the nearest neighours to, will have an effect on the timing performance. If the distribution of these operations and search queries are available, we can use those to generate them. However, without the usage and search query data, it is difficult to specify what this distributions might be. Instead, in this assignment, for search queries and adding a point, uniformly sample the x and y coordinates from within the region boundaries. For points to remove, uniformly sample an existing point.

Analysis

In your analysis, you should evaluate each of your implementations in terms of the different scenarios outlined above. Due to the randomness of the data, you should generate a few datasets with the same parameters settings (e.g. initial number of points) 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.

4 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 parameter settings you tested on, describe how the scenarios were generated (a paragraph and a figure or high level pseudo code), which approach(es) you decide to use for measuring the timing results..
- Evaluation of the approaches using the generated data. Analyse, compare and discuss your results across different parameter settings, data structures/algorithms 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 approach to help in your explanation.
- Summarise your analysis as recommendations, e.g., for this certain data scenario of this parameter setting, I recommend to use this approach because... We suggest you refer to your previous analysis to help.

5 Submission

The final submission (**in one single .zip file**) will consist of the codes, the report, and the contribution sheet:

- Your **Java/Python source code** of your implementations. Your source code should be placed into in the same code hierarchy as provided. The root directory/folder should be named as `Assign1-<partner 1 student number>-<partner 2 student number>-JAVA` or `Assign1-<partner 1 student number>-<partner 2 student number>-PYTHON`. Below, we will discuss the Java submission. A submission in Python can be done similarly, replacing 'JAVA' by 'PYTHON' in the name. Specifically, if your student numbers are s12345 and s67890, then the Java source code files should be under the folder Assign1-s12345-s67890-JAVA as follows:

- Assign1-s12345-s67890-JAVA/NearestNeighFileBased.java
- Assign1-s12345-s67890-JAVA/nearestNeigh/*.java (all the java files in nearestNeigh sub-directory, particularly NaiveNN.java and KDTreeNN.java, should be in this sub-directory).
- Any files you added, make sure they are in the appropriate directories/folders such that we can run script with the following command:
  ```
  python assign1TestScript.py <path to your folder>/Assign1-s12345-s67890-JAVA
  ...
  ```
- Assign1-s12345-s67890-JAVA/generation (generation files, see below).

When we unzip your submission, then everything should be in the folder Assign1-s12345-s67890-JAVA.

- Similarly, that folder also contains your **written report for part B** in PDF format, called "assign1-s12345-s67890.pdf". We have Assign1-s12345-s67890-JAVA/assign1-s12345-s67890.pdf.
- Your **data generation code** should be in Assign1-s12345-s67890-JAVA/generation. We will not run the code, but will examine their contents.
- Your group's **contribution sheet** in docx or PDF. See the following 'Team Structure' section for more details. This sheet should also be placed in Assign1-s12345-s67890-JAVA/.

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

6 Assessment

The assignment will be marked out of 30. Late submissions will incur a deduction of 3 marks per day, and no submissions will be accepted 7 days beyond the due date (i.e., last acceptable time is on 23:59, 17th of September, 2021).

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

Implementation (12/30):

- You implementation will be assessed on whether they implement the naive nearest neighbour search and KD-tree, respectively, and on the number of tests it passes in the automated testing.
- While the emphasis of this assignment is not programming, we would like you to maintain decent coding design, readability and commenting, hence these factors will contribute towards your marks.

Report (18/30):

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

7 Team Structure

This assignment could be done in pairs (group of two) or individually. If you have difficulty in finding a partner, post on the discussion forum. If you opt for a team of two, **it is still your sole responsibility to deliver the implementation and the report** by the deadline, even when the team breaks down for some reason, e.g., your partner doesn't show up for meetings, leaves the team, or, in some rare case, withdraws from the course. Effective/frequent communication and early planning are key.

In addition, please submit what percentage each partner made to the assignment (a contribution sheet will be made available for you to fill in), 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 $\frac{M}{\frac{X}{Y}}$.

8 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted assignment 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`.

9 Getting Help

There are multiple venues to get help. There are weekly lectorial Q&A sessions (see Canvas for time and location details). We may have one or two help sessions for questions on Assignment 1 if there is a demand. We will also be posting common questions on the Assignment 1 Q&A section on Canvas and we encourage you to check and participate in the discussion forum on Canvas. However, please **refrain from posting solutions**.

A Marking Guide for the Report

Design of Evaluation (Maximum = 5 marks)	**Analysis of Results** (Maximum = 8 marks)	**Report Clarity and Structure** (Maximum = 5 marks)
5 marks Data generation is well designed, systematic and well explained. All suggested scenarios, data structures/approaches and a reasonable range of parameters were evaluated. Each type of test was run over a number of runs and results were averaged.	7-8 marks Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures/approaches, scenarios and parameters. All analysis, comparisons and conclusions are supported by empirical evidence and possibly theoretical complexities. Well reasoned recommendations are given.	5 marks Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.
4 marks Data generation is reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures/approach or reasonable parameter setting. Each type of test was run over a number of runs and results were averaged.	5-6 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, data structures/approaches and parameters. Most analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Reasonable recommendations are given.	4 marks Clear and structured for the most part, with a few unclear minor sections.
3 marks Data generation is somewhat adequately designed, systematic and explained. There are several obvious missing suggested scenarios, data structures/approaches or reasonable densities. Each type of test may only have been run once.	3-4 marks Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios, data structures/approaches and parameter settings. A portion of analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Adequate recommendations are given.	3 marks Generally clear and well structured, but there are notable gaps and/or unclear sections.
0-2 marks Data generation is poorly designed, systematic and explained. There are many obvious missing suggested scenarios, data structures/approaches or reasonable parameters. Each type of test has only have been run once.	0-2 marks Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario, data structur/approach and parameter setting. Little analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Poor or no recommendations are given.	0-2 marks The report is unclear on the whole and the reader has to work hard to understand.