NOTTINGHAM TRENT UNIVERSITY

School of Science and Technology

COURSEWORK ASSESSMENT SPECIFICATION

- Details of module and team.
- What learning outcomes are assessed?
- What are my deadlines and how much does this assessment contribute to my module grade?

- What am I required to do in the assessment?
- What are my assessment criteria? (What do I have to achieve for each grade?)
- Can I get formative feedback before submitting? If so, how?
- What extra support could I look for myself?

- How and when do I submit this assessment?
- How and when will I get summative feedback?
- What skills might this work evidence to employers?

MODULE CODE	SOFT30161
MODULE TITLE	Advanced Software Engineering
MODULE LEADER	Dr. Neil Sculthorpe
COURSEWORK TITLE	Software Development Investigation
LEARNING OUTCOMES ASSESSED	Interpret and use formal measures of computational complexity. Comprehend the significance of the performance guarantees of code libraries, and the consequent implications for their selection and use. Implement recursive data structures. Evaluate the suitability of recursion in a given context. Postulate and test a performance hypothesis with respect to software development.
CONTRIBUTION TO MODULE	60%
DATE SET	Thursday 5th December 2019
DATE OF SUBMISSION	11pm Wednesday 26th February 2020
METHOD OF SUBMISSION	NOW Dropbox Folder (Turnitin Enabled)
DATE OF FEEDBACK	Wednesday 18th March 2020
METHOD OF FEEDBACK	Electronic via NOW; Oral feedback available on request

Late Submissions and NECs

Work handed in up to five working days late will be given a maximum grade of Low Third, whilst work that arrives more than five working days will be given a grade of Zero. Work will only be accepted without these caps if a *Notification of Extenuating Circumstances* (NEC) is submitted and upheld. NECs should be submitted through the following online portal:
https://ntu.ac.uk/current_students/resources/student_handbook/appeals/index.html

Plagiarism, Collusion and Turnitin

The University views **plagiarism and collusion** as serious academic irregularities. Penalties range from capped marks to dismissal from the course and termination of studies.

The Student Handbook has a section on Academic Irregularities, which outlines the penalties and states that **plagiarism** includes:

> "The incorporation of material (**including text, graph, diagrams, videos etc.**) derived from the work (published or unpublished) of another, by unacknowledged quotation, paraphrased imitation or other device in any work submitted for progression towards or for the completion of an award, which in any way suggests that it is the student's own original work. Such work may include printed material in textbooks, journals and material accessible electronically for example from web pages."

Whereas **collusion** includes:

> "Unauthorised and unacknowledged copying or use of material prepared by another person for use in submitted work. This may be with or without their consent or agreement to the copying or use of their work."

If copied with the agreement of the other candidate, both parties are considered guilty of Academic Irregularity. To help you avoid plagiarism and collusion, you are permitted to submit your report once to a separate drop box entitled 'Draft report', to view both the matching score and to look at what areas are affected. It is then down to you to make any changes needed. Note that Turnitin cannot say if something has been plagiarised or not. Instead it highlights matches between your text and other Turnitin content. For help, do not contact the module leader, but instead make use of the Library's Plagiarism and Turnitin Support.

I Assessment Requirements

This assessment consists of a series of tasks, some of which involve developing C++ code, and some of which involve analysing and explaining data structures and algorithms, and their time-complexity characteristics. You will also measure the performance of your code, and present and evaluate the results. There are four main tasks, some of which involve sub-tasks. Completing Task 4 is not necessary to *pass* this assessment, but is required for the higher grades.

Submission Requirements

You are required to produce a report containing your explanation/analysis/results for each task. If you complete all tasks, then your report should be approximately 6 to 12 pages in length, with a maximum limit of 15 pages (excluding title page and references). Any performance analysis should make use of appropriate technical notation. When presenting performance results you should use tables **and** graphs — you should *not* use screenshots. If you refer to any documentation in your explanations, then this should be properly referenced.

You should **not** include your source code in the report. Your complete set of source code files should be submitted separately, and should be appropriately commented. If you use or adapt any code fragments that are not your own work, this should be stated by a comment in the source file, and a reference to the original code given (see Section III).

You should submit the following files to the NOW dropbox folder:

- A single **PDF document** containing your report.
- A compressed archive file (e.g. ZIP or TAR) containing the set of C++ source files that you have developed (typically `.cpp` and `.h` files).

Note that your PDF report must be submitted separately from the archive file.[1]

Do not submit any other files relating to developing and compiling your code. In particular, you should not submit any generated files, and if you have used an integrated development environment to develop your code, then you should not submit any files specific to that environment.

The C++ code that you have developed should be portable across operating systems and development environments. If you are unsure about the portability of your code, then you should test the code on multiple operating systems. During the laboratory sessions, the module leader will demonstrate how to compile and run C++ code on a GNU/Linux distribution.

I.1 Implementing Pointer-based Data Structures

A *dictionary* is a data structure that stores a collection of key/item pairs, where each key in the dictionary is unique. Dictionaries are also known as *maps* and *associative arrays*.

Task 1a: Implement a dictionary as a C++ class, **using a singly linked list as the internal data structure**. The dictionary should support keys and items of type `std::string`.

Task 1b: State the time complexity characteristics of the public members of your dictionary implementation (including constructor, destructor, and operator overloads, if implemented). For each member, briefly (a sentence or two) explain why it has that complexity.

The full requirements specification for Task 1a is stated on the next page. Note that Task 1a is similar to, **but not the same as**, some of the formative laboratory activities. Read all of the requirements carefully.

[1] This is necessary because the Turnitin similarity checker does not analyse files within an archive.

For this task you are required to implement a singly linked list yourself using pointers; you should not use a pre-existing implementation of a singly linked list from a library. The class is required to provide (at least) the following public interface:

```
namespace Containers
{
   class Dictionary
   {
      public:
         using Key  = // ... must support at least std::string
         using Item = // ... must support at least std::string

         Dictionary();

         bool insert(Key, Item);
         Item* lookup(Key);
         bool remove(Key);
   };
}
```

The behaviour of this public interface is specified as follows:

1. Each possible value of type '`Key`' is considered to be in either a *present* or *absent* state.
2. If a key is in a *present* state, then it has an associated item of type '`Item`'.
3. The default constructor produces a dictionary in which all keys are considered to be *absent*.
4. After a call of '`insert(k,i)`', the key '`k`' is considered to be *present*, and the associated item becomes '`i`'.
5. After a call of '`remove(k)`', the key '`k`' is considered to be *absent*.
6. A call of '`lookup(k)`' should return a pointer to the associated item if the key '`k`' is *present*; otherwise it should return '**nullptr**'.
7. A call of '`insert(k,i)`' should return '**true**' if the key '`k`' was *absent* before the call; otherwise it should return '**false**'.
8. A call of '`remove(k)`' should return '**true**' if the key '`k`' was *present* before the call; otherwise it should return '**false**'.
9. None of these member functions perform console I/O.

Furthermore, to achieve a high grade for this task, your dictionary implementation should also:

- not introduce any *memory leaks*;
- provide a *copy constructor* and *copy assignment operator* that make a deep copy;
- provide an efficient *move constructor* and *move assignment operator*;
- use template parameters to be parametrically polymorphic in the type of its keys and items;
- provide a *higher-order* member function '`removeIf()`', such that after a call of '`removeIf(p)`', every key '`k`' for which '`p(k)`' returns '**true**' is considered to be *absent*.

To facilitate automated testing by the module leader:

1. ensure that the class is named '`Dictionary`', and is defined in the '`Containers`' namespace;
2. name your source files '`dictionary.h`' and '`dictionary.cpp`' (the latter may be omitted if you use templates).

Two small test files (one monomorphic, one polymorphic) that you can use to check the compatibility of your code with the required interface are available in the NOW learning room (in '`Content/Assessment/`').

I.2 Basket of Names: Search-based Algorithm

The classic textbook *The Art of Computer Programming* [1, Page 9] presents the *Basket of Names* problem:

> Three million men with distinct names were laid end-to-end, reaching from New York to California. Each participant was given a slip of paper on which he wrote down his own name and the name of the person immediately west of him in the line. The man at the extreme western end didn't understand what to do, so he threw his paper away; the remaining 2,999,999 slips of paper were put into a huge basket and taken to the National Archives in Washington, D.C. Here the contents of the basket were shuffled completely and transferred to magnetic tapes.
>
> At this point an information scientist observed that there was enough information on the tapes to reconstruct the list of people in their original order. And a computer scientist discovered a way to do the reconstruction with fewer than 1000 passes through the data tapes, using only sequential accessing of tape files and a small amount of random-access memory. How was that possible?
>
> [In other words, given the pairs (x_i, x_{i+1}), for $1 \leq i < N$, in random order, where the x_i are distinct, how can the sequence $x_1 x_2 \ldots x_N$ be obtained, restricting all operations to serial techniques suitable for use with magnetic tapes?]

If we ignore the 'serial techniques only' constraint, then it is fairly straightforward to construct a search-based algorithm that solves this problem. One such algorithm is as follows:

1. Insert the contents of the pieces of paper into a data structure suitable for searching.
2. Create an empty container suitable for storing the resulting sequence of names.
3. Arbitrarily choose one of the names as a starting point.
4. Insert the name at the back of the result sequence, and then perform a search to identify his westerly neighbour (if any). Repeat this step until the most westerly name has been added to the sequence.
5. Now return to the starting name.
6. Perform a search to identify his easterly neighbour (if any), and then insert that name at the *front* of the result sequence. Repeat this step until the most easterly name has been added to the sequence.

The C++ standard library provides numerous containers that we could use to implement this algorithm, including:

- `std::list`
- `std::map`
- `std::unordered_map`

Task 2a: Describe the typical data structures used to implement these three containers. For each container, identify suitable functions for searching and insertion that could be used to implement the search-based algorithm. State the time-complexity guarantees of these functions, and explain how the data structures used enable those guarantees.

Task 2b: Explain and compare the performance consequences of using different combinations of these containers to implement the search-based algorithm. For a high grade, this should include analysing the time-complexity of the algorithm as whole. If there are any space usage implications, these should also be discussed. Choose which combination you consider best, and justify your choice.

I.3 Basket of Names: Search-based Implementation

Task 3a: Implement the search-based algorithm from Task 2 using your chosen combination of containers. For a high grade, repeat this task using one of the other combinations of containers that you considered.

Task 3b: Use your implementation(s) to generate empirical performance data, and present the data in your report. Evaluate your results, and explain whether they support or contradict the performance predicted by your analysis from Task 2.

A set of data files representing instances of the *Basket of Names* problem are available in the NOW learning room (in '`Content/Assessment/`'). Data files of varying sizes are provided; use those that allow you to present results from which you will be able to draw meaningful conclusions.

To allow the module leader to easily check the correctness of your implementation, your submitted files should include a `main()` function that:

- accepts input data in the same format as the test data provided;
- takes a file path to the input file as a command-line argument;
- outputs the result sequence (but nothing else) to the standard output.

Note that you are not required to use this particular `main()` function for generating your performance data; this is just to facilitate correctness checking.

I.4 Basket of Names: Serial Algorithm

The Art of Computer Programming [1, Pages 590–591] contains a solution to the *Basket of Names* problem. Unlike the search-based algorithm from Task 2, this solution satisfies the constraint that only serial techniques are used for storing and accessing the data.

Make another copy of the input file; sort one copy on the first components and the other on the second. Passing over these files in sequence now allows us to create a new file containing all pairs (x_i, x_{i+2}) for $1 \leq i \leq N-2$, and to identify (x_{N-1}, x_N). The pairs $(N-1, x_{N-1})$ and (N, x_N) should be written on still another file.

The process continues inductively. Assume that file F contains all pairs (x_i, x_{i+t}) for $1 \leq i \leq N-t$, in random order, and that file G contains all pairs (i, x_i) for $N-t < i \leq N$ in order of the second components. Let H be a copy of file F, and sort H by first components, F by second. Now go through F, G, and H, creating two new files F' and G', as follows. If the current records of files F, G, H are, respectively (x, x'), (y, y'), (z, z'), then:

i) If $x' = z$, output (x, z') to F' and advance files F and H.

ii) If $x' = y'$, output $(y-t, x)$ to G' and advance files F and G.

iii) If $x' > y'$, advance file G.

iv) If $x' > z$, advance file H.

When file F is exhausted, sort G' by second components and merge G with it; then replace t by $2t$, F by F', G by G'.

Thus t takes the values $2, 4, 8, \ldots$; and for fixed t we do $O(\log N)$ passes over the data to sort it. Hence the total number of passes is $O((\log N)^2)$. Eventually $t \geq N$, so F is empty; then we simply sort G on its *first* components.

This algorithm is *described*, but not *explained*. Consequently you may have to study the algorithm for some time before you understand it.

In particular, notice that the algorithm consists of three stages. The first stage (the first paragraph) is not described in complete detail — you have to figure out how to create the file contents that are described. The second stage (the second paragraph) is described in more detail, though there are still some ambiguities in the pseudo-code. Notice that the process is described as *inductive* (i.e. a recursive process that is structurally guaranteed to terminate), so the second stage should be repeated recursively. The third paragraph comments on the time complexity of the algorithm, states the termination condition for the recursion, and describes the third stage of the algorithm.

You will probably find it helpful to work through a small example by hand (say, 10 names), so that you can observe how the algorithm proceeds.

Task 4a: Analyse the time complexity of the serial algorithm, and relate your analysis to the claimed complexity.

Task 4b: Implement the serial algorithm. For simplicity you may use sequence containers instead of files, **but you must respect the 'serial techniques only' constraint** by only iterating through those containers sequentially — do not use random access or searching. You may use standard library algorithms such as '`std::sort()`' in the same manner that the serial algorithm does.[2]

Task 4c: Use your implementation to generate empirical performance data, and present the data in your report. Evaluate your results, and explain whether they support or contradict the performance predicted by your analysis from Task 4a.

II Assessment Criteria

Task 1a — Implementing Pointer-based Data Structures (Weighting 20%)	
Class	**Indicative Criteria**
1st	A C++ class has been implemented providing all of the specified functionality, and precisely meeting all requirements. Use of the class does not introduce any memory leaks or cause the stack to overflow.
2.1	A C++ class has been implemented providing the `insert()`, `lookup()` and `remove()` functions, and correct *copy* and *move* operations. The implementation is bug-free, though minor aspects of the specification may have been misinterpreted. Use of the class does not introduce any memory leaks.
2.2	A C++ class has been implemented providing the `insert()`, `lookup()` and `remove()` functions. Use of these operations and the destructor does not introduce any memory leaks. The implementation provides usable functionality, but may contain minor bugs.
3rd	A C++ class has been implemented providing the `insert()` and `lookup()` functions. The implementation provides some usable functionality, but may contain serious or numerous bugs.
Marginal Fail	A C++ class has been implemented, but does not provide usable functionality because of significant bugs or incomplete definitions.
Mid/Low Fail	An attempt has been made to implement a C++ class, but it does not come close to providing usable functionality. OR, the submitted work does not address the assignment brief (e.g. a linked list is not used for the internal implementation).

[2] You may assume that the sorting algorithm is implemented using sequential access. This assumption may not be true, but the time-complexity guarantees given by the standard-library sorting algorithms can also be met by a sequential sorting algorithm, so this assumption does not invalidate the investigation.

Tasks 1b, 2a, 2b & 4a — Performance Analysis (Weighting 40%)	
Class	**Indicative Criteria**
1st	Correctly identifies and explains in detail the typical implementation data structures and time-complexity characteristics of the standard-library containers, and clearly conveys the reasons for those characteristics, demonstrating deep understanding. Correctly analyses the time-complexity characteristics of the dictionary implementation, the search-based algorithm, and the serial algorithm. Formally states the characteristics, making detailed and precise use of appropriate technical notation. Correct and compelling justification for the choice of containers.
2.1	Correctly identifies and explains the typical implementation data structures and time-complexity characteristics of the standard-library containers, and conveys the reasons for those characteristics. Correctly analyses the time-complexity characteristics of the dictionary implementation and the search-based algorithm. Formally states the characteristics, correctly distinguishing between best/average/worst case, and amortised/non-amortised complexity, where appropriate. Mostly valid justification for the choice of containers, though reasoning may neglect some details.
2.2	Mostly correctly identifies and describes the typical implementation data structures and time-complexity characteristics of the standard-library containers, and provides some limited justification for those characteristics. Mostly correctly analyses the time-complexity characteristics of the dictionary implementation. Formally states the characteristics, correctly distinguishing between best/average/-worst case, though may not distinguish between amortised/non-amortised complexity.
3rd	Mostly correctly identifies and describes the typical implementation data structures and time-complexity characteristics of the standard-library containers, perhaps with some lack of clarity. Formally states the characteristics, though may not distinguish between best/average/worst case, or between amortised/non-amortised complexity.
Marginal Fail	Discusses the typical implementation structures and performance characteristics or the standard library containers, but either does not formally state the time complexities, or the claims contain significant inaccuracies or misconceptions.
Mid/Low Fail	Discussion of the containers is limited to orthogonal or irrelevant aspects.

Tasks 3a & 4b — Implementing Algorithms (Weighting 20%)	
Class	**Indicative Criteria**
1st	Both the search-based and serial algorithms have been implemented, and produce correct output for all test data. The implementations make ideal use of library components.
2.1	The search-based algorithm has been implemented, produces correct output for all test data, and mostly makes good use of library components. The serial algorithm has been mostly implemented, but it is not quite complete or correct.
2.2	The search-based algorithm has been correctly implemented and produces correct output for all test data.
3rd	The search-based algorithm has been implemented and produces correct output for some test data, though may contain minor bugs.
Marginal Fail	The search-based algorithm has been mostly implemented, but it is not quite complete or correct.
Mid/Low Fail	An attempt has been made at an implementation, but it is mostly incomplete or incorrect.

Tasks 3b & 4c — Performance Measurement (Weighting 20%)	
Class	**Indicative Criteria**
1st	Well-chosen performance data has been generated, allowing for a thorough and reliable comparison of different solutions to the *Basket of Names* problem. The data is excellently presented, making ideal use of tables AND graphs. An insightful interpretation is given, and correctly related to the performance analysis. The conclusions are strongly supported by the data.
2.1	Sensible performance data has been generated, allowing for a meaningful comparison of different solutions to the *Basket of Names* problem. The data is well presented, using tables AND graphs. A reasoned interpretation is given, and correctly related to the performance analysis. The conclusions are supported by the data, but perhaps may be somewhat overstated.
2.2	Relevant performance data has been generated, and presented using tables and/or graphs. Some interpretation is given, and an attempt is made to relate it to the performance analysis, though perhaps with inaccuracies. The conclusions may not be reliably supported by the data.
3rd	Some limited performance data has been generated and presented. Some interpretation is given, though without relating the results to the performance analysis. The data may be insufficient to support the conclusions.
Marginal Fail	Some performance data has been generated and presented. However, either no interpretation of that data is given OR insufficient data has been generated to allow meaningful conclusions to be drawn.
Mid/Low Fail	An incomplete attempt at generating and presenting performance data has been made.

III Referencing, Plagiarism and Collusion

This coursework assessment is an individual assignment. This means that the solutions to the tasks should be developed individually, and the report presenting and describing your solutions should be individually authored.

In particular, any program code in your submission that was adapted or taken from another source must be clearly identified and an appropriate reference given. The reference must identify precisely where the code is to be found so that it can be examined when assessing your work. For example, a reference to a website must identify the specific webpage on which the code is found, whereas a reference to a textbook must identify the specific subsection and pages on which the code is found.

Failure to identify and correctly reference such material is **plagiarism**. This includes code written by another student — such code, if not publicly available, can be referenced as a 'personal communication', and a copy of the original code submitted as an additional file with your submission. If any code in your submission is adapted from another piece of code of which you are a *joint* author, then you should state this, provide a copy of the original code, and make clear what adaptations you have made individually. You should not jointly develop code to solve the coursework tasks — doing so is **collusion**.

In the report, any text that was not written by yourself must be enclosed in quotation marks and the source acknowledged by providing an accompanying citation referring to an entry in your reference list. Any diagrams that you did not create should also be cited and referenced. Failure to do so is **plagiarism**. If you summarise or paraphrase another person's work without a citation and reference to acknowledge the source, then that is also **plagiarism**.

Note that the key here is *acknowledgement*. You may make appropriate use of code fragments or diagrams created by others (subject to copyright restrictions), provided acknowledgement is given. You are also encouraged to discuss ideas and share reference materials with other students, prior to individually developing your own code and writing your own report. If you are in any doubt about whether your use of another's work is appropriate or correctly acknowledged, then ask the module leader for guidance before submitting your assignment solution.

IV Feedback Opportunities

During the laboratory sessions, you will receive verbal feedback on the formative laboratory activities that relate to the coursework assessment. You will receive written feedback on your submitted work within three weeks of the submission deadline, together with your awarded grade.

V Moderation

The clarity of this specification, and the appropriateness of the assessment criteria, have been checked by two members of the Department of Computing and Technology. The grades awarded by the assessor will be reviewed by another member of the Department to check for consistency and fairness.

VI Resources that may be Useful

NTU library offers support with academic writing, and with mathematics and statistics skills. Individual appointments with librarians or student mentors can be booked online [2]. The library has also published a guide for the correct use of citations and references [3]. NTU also provides online advice for avoiding plagiarism [4].

VII Aspects for Professional Development

This assignment will develop your abilities to make reasoned and justified choices when selecting data structures and algorithms to implement solutions to specific software engineering problems. You will also gain experience of testing and evaluating the performance of your programs.

References

[1] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1998. ISBN 0-201-89685-0.

[2] NTU Library. NTU Library bookings. URL http://librarybookings.ntu.ac.uk/. [Accessed 16/07/18].

[3] NTU Library. *Citing references: A guide to NTU Library Harvard Style*, 10th edition, 2016. URL https://now.ntu.ac.uk/d2l/le/content/52836/viewContent/1019510/View.

[4] NTU. Plagiarism. URL http://www4.ntu.ac.uk/current_students/studying/skills-for-success/copyright-plagiarism/plagiarism.html. [Accessed 16/07/18].