In this assignment we shall see how to use runtime analysis to make choices when doing actual programming. Without knowing the basics of comparing functions in an asymptotic sense, it is simply not possible to know what the correct choice is when faced with the decision of using either a `java.util.HashMap` or `java.util.TreeMap` or whatever the equivalent is for your programming language of choice. In this assignment you will be forced to think of the trade-offs inherent in making such choices.

1 Collections: Table of Reference (3 pts)

Fill in the following table, no justifications are needed.

Expected time			
	Instruction		
Data structure	INSERT	GET-ITH-ELEMENT	CONTAINS
Linked List			
Array			
Hash Table[1]			
Binary Search Tree		$\mathcal{O}(n)$	

Make sure you fill in expected time, and *not* worst case. You can assume that the set of elements contains no duplicates, and that any array is allocated with sufficient slack capacity as to hold the additional elements you want to insert. Also assume that the collections are not sorted unless for BST where sortedness is intrinsic to the data structure. State any additional assumptions you make, if anything is unclear.

2 Helping Hermione (2+2+3 pts)

Even though Hermione is an awesome witch, it turns out she has problems choosing correct data structures. As her friend, that task is now yours. For all of the questions, answers should have the following form:

Answer: Foo
Since $f_1 + f_2 = \mathcal{O}(g)$ and structure A [has some property], we get better performance with structure B. Structures C and D give even worse performance, namely $\mathcal{O}(h_1)$ and $\mathcal{O}(h_2)$ when performing [some operation], so they are clearly out of the question.

In other words, all answers should clearly state the answer and then justify it in two or three sentences. Justifying includes explaining why other potential answers are not the right ones.

1. Hermione has written an application to keep track of her many books, where she stores her books in a linked list. However, she complains that it when the running time seems to be quadratic, not linear as she expected, when she prints all book titles. Below the code of PRINT-ALL-BOOK-TITLES is given. Explain what the problem is and suggest a solution, either by changing book printing algorithm or changing the data structure used for keeping track of books. Also state the expected running time for book printing when the program is changed according to your suggestion.

   ```
   PRINT-ALL-BOOK-TITLES(Books)
   1  for i ← 1 to size(Books)
   2        do book ← GET-ITH-ELEMENT(Books, i)
   3           Print book
   ```

2. The complaint that the printing of all book names is slow is, although correct, not very important. Way more problematic is the fact that checking whether she has a book or not is linear. As this is an operation performed more often than printing all books, she needs to structure her data in such a way that this sort of lookup can be done efficiently. Suggest what data structure she should use if she wants to optimize performance in this regard, and what the expected running time of lookup would be then.

3. After using her application (modified according to your suggestion above) for some time, Hermione has the following two remarks:

   - It turns out that Hermione does not do very many lookups either. Her empirical work tells her that she does approximately $\lg n$ lookups every week.
   - She is really annoyed by the fact that the list of books that is printed is still not sorted. She of course knows of all sorts of sorting algorithms, but she has a feeling that she should be able to maintain her data in such a way that this is done automagically. She prints her books once every week.

   Suggest the data structure that Hermione should use so that the total work per week (lookups and printing, ignore insertions) is minimized, and state the resulting expected asymptotic bound on the weekly computation time.

[1] Assume uniform hashing and collision resolution by chaining.