Comp-2540: Lab Assignment 4: Heap and Hashing (2025)

❀Your Marks (For TA Use Only)

	3.1	3.2	4.1	4.2	Bonus	Total	Marked by
	1.5	0.5	1	1			

1 Due Date and Submission

The due time is your lab section in the due week. To be submitted 10 minutes before the end of your lab section. You can also submit in the labs one week earlier or during office hours. Submit the PDF file to BrightSpace, and test your code at our test site.

Make sure that you add your signature below to reaffirm that you followed Senate Bylaws 31 (click here for the document). Some PDF readers may not support digital signature well. We recommend you to use Acrobat Reader that is free for downloading.

❀Fill and Sign The Form

I, your name _____, **verify that the submitted work is my own work**.

Date   Student Number   EMail ID

2 Objective

The aim of this assignment is to understand Heap and Hashing. We will implement the Heap ADT, and use our Heap class to implement Heap sort. For the Hashing method, we will implement our own HashMap, and solve the collision using separate chain. We will learn how to generate hash code for a string, and how to generate it efficiently using Horner's method.

3 Heap and Heap Sort (2 marks)

`PriorityQueue` is a Java class that is implemented using Heap data structure. We can implement `HeapSort` using `PriorityQueue` as follows, which is not efficient:

```java
public static void heapSortJava(String[] a) {
    PriorityQueue<String> pq=new PriorityQueue<String>();
    int n = a.length;
    for (int i=0; i < n; i++)      pq.add(a[i]);
    for (int i=0; i < n; i++)      a[i] = pq.remove();
}
```

3.1 Implement Heap2540 class (1.5 marks)

Instead of using the PriorityQueue class provided in Java SDK, we will implement it, called `Heap2540` by ourself. The heap sort code is also a little different as listed below. Instead of adding elements one by one, we construct the heap from an array all at once.

```
public static String[] heapSort2540(String[] a) {
        Heap2540 pq = new Heap2540(a);
        int n = a.length;
        String[] result = new String[n];
        for (int i = 0; i < n; i++) {
                result[i] = pq.removeMax();
        }
        return result;
}
```

We provided the starter code in our submission site. It is not runnable yet because there are a few lines are missing in three places:

- `swim()`: Heapify the array bottom up.

- `sink()`: Heapify the array top down.

- The Constructor `Heat2540(String[] a)` : construct a heap from an array;

You will get the marks only if your code can run and pass our test on our submission site. There are no partial marks.

3.2   Complexity of Heap Sort (0.5 mark)

Write below the time complexity of the heap sort, supposing that the data size is n. Give your justification briefly.

4   Hashing

Hashing is the fast way to count word frequency. This time we will not use a sorting method. Instead, each time when we see a word, we will find the word and its frequency, and update its frequency. Instead of the slow sequential searching as we did in A1, we will use hashing to access the word directly. The overall logic is described below:

```
//HashMap2540 to be implemented by you
HashMap2540<String, Integer> map = new HashMap2540<String, Integer>();
int len = tokens.length;
//Update the frequency
for (int i = 0; i < len; i++) {
        String token = tokens[i];
        Integer freq = map.get(token);
        if (freq == null)
                map.put(token, 1);
        else
                map.put(token, freq + 1);
}

//find the most popular word
int max = 0;
```

```
        String maxWord="";
        for (String k : map.keys())
                if (map.get(k) > max){
                        max = map.get(k);
                        maxWord=k;
                }
        return new AbstractMap.SimpleEntry<String, Integer>(maxWord, max);
        }
```

Your task is to implement HashMap2540, so that word counting can work. Note that you are not allowed to use existing implementations, such as the Hashtable class in Java.

In addition to HashMap2540, you will need a list for collision handling. It is provided as LinkedListForHash2540, and you do not need to change anything in this class. We use separate chaining to handle the collision. The collisions are stored in this class. It is a linked list, each element stores a (word, freq) pair.

4.1   Implement the `put(key, value)` method (1 marks)

The starter code for HashMap2540 is at the submission site. Your first task is to fix the `put(key, value)` method that is listed as below. There are three lines missing. Please fill in the three lines so that the program can run and get the correct counting. As before, your code should be able to run on our submission site.

```
public void put(Key key, Value val) {
        // double table size if load factor m/n >0.1
        if (n >= 10 * m)
                resize(2 * m);
        // Put your code here.
        // 1) get the hash value of the key i.
        // 2) then put (key, value) in the i-th linkedList implemented in LinkedListForHash254
        // 3) you need to handle the case whether the key is already there.
        int i = myhash(key);
        if (st[i].get(key)==null)
                n++;
        st[i].put(key, val);
}
```

Write the code below in the form for the convenience of marking. Note that you must get the code run correctly on our submission site.

4.2   Improve the hashing method (1 marks)

In this implementation, you shall observe that the Hash method and Heap method have similar performances. It is against our knowledge –Hashing should be much faster than Heap sorting. A careful reading of the program reveals that the problem is in the $myhash()$ method. It uses the polynomial method to calculate the hashcode of a string using the following formula:

$$hash = 31^{n-1}key[0] + 31^{n-2}key[1] + ... + key[n-1]$$

The approach and the formula have no problem, but the corresponding code below is too slow:

```
//calculate hashcode (h1) using the polynomial method:
for (int i = 0; i < k.length(); i++)
        hash=(int)Math.round((Math.pow(31,k.length()-i-1)))*k.charAt(i)+hash;
```

This code is slow because of the invocation of the power function Math.pow(). Please rewrite the code using Horner's method, so that in each loop you only need to a multiplication once. Please refer to P.413 of the book for the detailed explanation. You will get the marks for this part if the mark program says that the speed improvement is good enough.

5   Bonus:Hash Hacking (1 mark)

If you run the hashing method on the hackedString200kwords.txt, you will find that it is extremely slow: my machine takes 4289ms using Hashing method, but only 115ms using HeapSort. The reason is that most of the words in the file have the same hashcode, hence the chain will be very long in the hashing method. Please rewrite the hashcode part so that it is more efficient. You need to submit your code on our submission site, and the top 6 students will get the bonus mark.