Northern Michigan University (Marquette Co, MI)

CS422-01-25W: Algorithms (Andrew A. Poe) Name: _____
Practice Endterm Examination (Exam 2) Monday 21 April 2025 9:00 A.M. EST

Time: 50 minutes

1. Write the code for the following method

```
public String BinaryFileSearch (RandomAccessFile f, int reclen, String s)
 { ... }
```

f is a random access file that is already open. You don't have to open or close it. reclen is the length of each individual record. The records are in sorted order. Use binary search to search the file for a record *beginning with* s. For example, if the record length is 50, and the search string is DOG, you are not looking for the string DOG, you are looking for a record beginning with DOG. If such a record exists, return the entire record. If more than one such record exists, just return the first one you find. If no such record exists, return null.

```
public String BinaryFileSearch (RandomAccessFile f, int reclen,
String s) {

  int rct = (int) (rf.length()/reclen);
  int first = 0, last = rct-1, t = 0;
  boolean found = false;
  while (first <= last && !found) {
   t = (f+l)/2;
   f.seek (t*reclen);
   byte[] ba = new byte[reclen];
   rf.read (ba);
   String A new String (ba);
   int c = s.compareTo (A.substring (0,s.length()));
   if (c==0) found=true;
   else if (c < 0) last = t-1;
   else first = t+1;
  }
  if (!found) A = null;
  return A;
}
```

2 Consider the following sequence: f(0)=0, f(1)=1, f(2)=1, f(3)=3, f(4)=5, f(5)= 11, f(6)=21, etc. Each number in the sequence can be built from the two previous using a simple formula.

Write the code for `BigInteger f (int n) { ... }` which uses recursion and a HashMap to efficiently compute the value of f for a given nonnegative integer. It doesn't take long for the answers to get huge, so you'll have to use BigInteger.

```
private HashMap <String,String> HT = new HashMap ();

public BigInteger f (int n) {
```

```
 BigInteger bi = null;
 String ans = HT.get (""+n);
 if (ans != null) bi = new BigInteger (ans);
 else {
  if (n<=1) bi = new BigInteger (""+n);
  else bi = new BigInteger ("2").multiply(f(n-2)).add (f(n-1));
  HF.put (""+n,bi);
 }
 return bi;
}
```

3. Using the Radix Sort as described in class, sort the following array of five strings. Show all steps.

AAA , A , AAAAA , AAAA , AA

0. AAA
1. A
2. AAAAA
3. AAAA
4. AA

position 5 has 4 out of bounds and 1 A: 4 1 Summed array: 4 5

0.AAA (0 4)
1.A (1 4)
2.AAAA (2 5)
3.AA (3 5)
4.AAAAA (2 4)

Position 4 has 3 out of bounds and 2 A's. A: 3 2 Summed array: 3 5

0.AAA (0 3)
1.A (1 3)
2.AA (2 4)
3.AAAA (2 3)
4.AAAAA (3 4)

Position 3 has 2 out of bounds and 3 A's: 2 3 Summed array: 2 5

0.A (0 3)
1.AA (1 3)

2.AAA (0 2)
3AAAA (2 3).
4.AAAAA (2 4)

Position 2 has 1 out of bounds and 4 A's. 1 4 Summed array: 1 5

0.A. (0 1)
1.AA (1 1)
2.AAA (1 2)
3.AAAA (1 3)
4.AAAAA (1 4)

Position has 0 out of bounds and 5 A's: 0 5 Summed array: 0 5

0.A (0 0)
1.AA (0 1)
2.AAA (0 2)
3.AAAA (0 3)
4.AAAAA (0 4)

4. Given a trie of strings of capital letters, describe a recursive algorithm that given an input string, prints all strings in the trie that END with the input string. For example, if the input string is DOG, the strings DOG , HOUNDDOG , BULLDOG , MADDOG would all be printed if they are in the trie. Your algorithm should move down the tree, never up. As you move through recursion, you should remove the first letter of the input string as needed, but you should only examine the first letter at any given point. You don't have to provide code. A description in English is perfectly acceptable. Be sure to list all base cases and all recursive cases.

We have an algorithm to find an exact match:
If search string is empty, print and return if we're at an end-of-string marker, otherwise just return.
If we there is no node corresponding to the first letter of our search string, just return.
If there is a node corresponding to the first letter of our search string, recurse starting from that node with the search string minus the first letter.

Now, for the desired algorithm.

Recursively run this algorithm on all of your children with the SAME search string.
AND run the exact match algorithm on this same node with this same string.