Northern Michigan University (Marquette Co, MI)

CS422-01-25W: Algorithms (Andrew A. Poe) Name: _____
Endterm Examination (Exam 2) Page 1/2 Friday 25 April 2025 9:00 A.M. EDT

Time: 50 minutes

1. Write the code for the following method

```
public void FileReverse (RandomAccessFile f, int reclen)
 { ... }
```

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. You are to reverse the order of records in the file. For example, if the file were "CATDOGBAT" and the record length were 3, the method should transform the file into "BATDOGCAT". You are not allowed to have more than two records in memory at any given time. You are not allowed to use an additional file. Reverse the file using the smallest number of record reads and record writes as possible.

```
public void FileReverse (RandomAccessFile f, int reclen) {
  int rct = (int) (f.length()/reclen);
  int first = 0, last = rct-1;
  while (first < last) {
   f.seek (first*reclen);
   byte[] ba = new byte[reclen];
   rf.read (ba);
   String A new String (ba);
   f.seek (last*reclen);
   byte[] ba = new byte[reclen];
   rf.read (ba);
   String B new String (ba);
   f.seek (first*reclen);
   f.writeBytes (B);
   f.seek (last*reclen);
   f.writeBytes (A);
   first++; last--;
  }
}
```

2 Consider the following sequence: f(0)=1, f(1)=1, f(2)=3, f(3)=5, f(4)=9, f(5)= 15, f(6)=25, etc. Each number in the sequence can be built from the two previous using a simple formula. (There is also a simple formula building each number from the three previous numbers. You can use either formula.)

Write the code for `public 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 ("1");
//else if (n == 2) bi = new BigInteger ("2");
  else bi = f(n-2).add (f(n-1)).add (new BigInteger ("1"));
// OR bi = new BigInteger ("2").multiply(f(n-1)).subtract(f(n-
3));
  HF.put (""+n,bi.toString());
 }
 return bi;
}
```

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

BC , BA , BD, AB , AD , AC

0 BC.
1. BA
2. BD
3. AB
4. AD
5. AC

last char: A: 1 B: 1 C: 2 D: 2 Summed: 1 2 4 6

0. BA [0 1 3 4]
1. AB [1 1 3 5]
2. BC [0 1 2 4]
3. AC [1 2 3 6]
4. BD [1 1 3 4]
5 AD [1 2 3 5].

first char: A: 3 B: 3 Summed: 3 6

0. AB [0 4]
1. AC [1 5]
2. AD [2 6]

3. BA [0 3]
4. BC [1 4]
5. BD [2 5]

4. Given a trie of strings of capital letters, describe a recursive algorithm that, given a search string, prints all strings in the trie that contain the search string as a subsequence. For example, CHART contains CAT as a subsequence. The letters C A T are all found in CHART, in the correct order, but not necessarily all in a row. If the search string is CAT, your search should find CAT, CART, CHART, HELLCAT, CARTHAGE, TACAT, and many others, provided they are in the trie. Remember: use recursion; you can move down the trie but not up. Only look at the first letter of the search string at any given time. You can change the search string only by removing the first letter. You don't have to write code; just describe your algorithm in English. Indicate all base and recursive cases.

If the search string is empty, and I'm at an empty string marker, PRINT the string but don't return, continue.
If the search string is empty, and I'm not at an empty string marker, don't print, but don't return either, continue.

For each of my children, if the child matches the first character of my search string, recurse on that child with the first character removed from the search string.
If the child doesn't match the first character of my search string, recurse on that child with the search string unchanged.

(If the search string is empty, we say that the first character doesn't match anything, since there is no first character.)

There is no true base case here. When a node has no children, recursion can't continue, and that's the closest thing we have.