COMP26120 - January 2014 - Answers

Todd Davies

December 29, 2014

Please don't assume these answers are right. This is me attempting a past paper for revision purposes; I could have got it all wrong ;)

1 Question 1

1.1 Part a

```
1  public static boolean hasSum(int[] input, int sum) {
2    for(int i = 0; i < input.length; i++) {
3      for(int j = 0; j <= i; j++) {
4        if((input[i] + input[j]) == sum) {
5          return true;
6        }
7      }
8    }
9    return false;
10 }
```

Listing 1: Add all of the pairs of integers together in the list. Do this by looping through the list, and looping from 0→i on each iteration.

Since as we're iterating through the list, we are describing an arithmetic series (we do one additions, then two, then three etc), we can use the formula $\frac{n(1+n)}{2}$ to find how many operations we're doing. This, in the Big-Oh notation, equates to $O(n^2)$ since constant terms are eliminated. In most cases, the result will be found significantly faster though.

1.2 Part b

```
1  import java.util.ArrayList;
2
3  public class part2a {
4    public static ArrayList<Integer> intersect(int[] input, int[] input2) {
5      BitSet bs = new BitSet(input.length);
6      for(int i : input) bs.set(i);
7      ArrayList<Integer> intersection = new ArrayList<Integer>();
8      for(int i : input2) if(bs.get(i)) intersection.add(i);
9      return intersection;
10   }
```

Listing 2: Add the first list to a BitSet (a very compact array of bits), then iterate through the second list and add items that are in the bit set to the output array.

The worst case runtime of this algorithm is $O(n + m)$ since it iterates through both arrays once, and the loops aren't nested.

1.3 Part c

```
1  public static char[] shift(char[] input, int shiftNum) {
2    char[] firstHalf = Arrays.copyOfRange(input, 0, shiftNum);
3    char[] secondHalf = Arrays.copyOfRange(input, shiftNum, input.length);
4    for(int i = 0; i < (input.length - shiftNum); i++) {
5      input[i] = secondHalf[i];
6    }
7    for(int i = (input.length - shiftNum); i < input.length; i++) {
8      input[i] = firstHalf[i - (input.length - shiftNum)];
9    }
10   return input;
11 }
```

Listing 3: Create a new array containing the first half, and a new array containing the second half, then replace them insert them into the original array.

The worst case runtime of this function is $O(n)$, since it only iterates through the array once. The space complexity is $O(2n)$, since I couldn't figure out how to do an in-place circular shift.