

# Questions for Week 1

## Monday

### Warm up Questions

1. We have two monkeys, a and b, and the boolean variables aSmile and bSmile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Which of the following statements returns true if and only if we are in trouble.



* 1. return (aSmile || bSmile);
	2. return (aSmile && bSmile);
	3. return (aSmile == bSmile);

### Algorithms and Data Structures

1. In your own words, complete the following sentences.

An algorithm is …

A data structure is ….

[you are asked to answer these *in your own words* - so we do not supply a sample solution]

### Big O Notation

1. Group the following functions into equivalent Big-Oh functions:

x2, x, x2 + x, x2 − x, and (x3 / (x − 1)).

* x
* x2, x2 + x, x2 − x, (x3 / (x − 1))
1. Solving a problem requires running an O(N) algorithm and then afterwards a second O(N) algorithm. What is the total cost of solving the problem? Why?

O(N), since 2N has order N.

1. Solving a problem requires running an O(N) algorithm and then afterwards an O(N2) algorithm. What is the total cost of solving the problem? Why?

O(N2), since N2+N has order N2

1. In terms of n, what is the running time of the following algorithm to compute x to the power n (xn)? Can you think of a faster approach?

public static double power( double x, int n ) {

 double result = 1.0;

 for( int i = 0; i < n; i++ ) {

 result = result \* x;

 }

 return result;

}

O(n), since the loop is executed n times.

There *is a* faster approach – it used a “binary splitting” approach, x2m = xm \* xm. This solution will be O(log n).

1. Which of the following statements make sense or not ? Why?
	1. My algorithm is O(2N2)

Instead write O(N2) as the simplest function for f.

* 1. My algorithm is > O(N2)

Does not make sense, since big O gives an upper bound.

* 1. My algorithm is O(N2 + N)

Instead write O(N2) as the simplest function for f.

* 1. A method with one loop nested inside another must be O(N2)

Not necessarily true (though often it will be).

- It may not be the case that both loops run over all the input (or if they do, that they fully complete)

- Once could write a method with nested loops where no statements were performed inside the loop

- In both these cases, complexity could be *less* than O(N2).

But additionally:

- This doesn’t rule out methods where further loops are nested inside the outer two; so the complexity could be *more* than O(N2).

* 1. If method A is O(N) and method B is O(N) then an algorithm which performs A followed by B is also O(N)

 True

[See Weiss Ch 6 Common Errors (at the end of the chapter) for more explanation of these points]

1. Consider an **array implementation** of the stack ADT. Give a short description of an implementation for each of its functions in words. What is the big Oh of each of these operations, and why?

isEmpty

isFull

pop

push

isEmpty: check whether number of elements is 0

isFull: check whether number of elements is same size as the array

pop: return the element on the top of the stack, and update the top-of-stack index.

push: add an element to the top of the stack, and update the top of stack index.

All these operations are O(1) because they take constant time. They do not depend on the size of the stack.

1. The following method searches an array (stored in block) to see whether the same item appears twice.   If so, it returns true. If no duplicates are found it returns false.

public boolean hasMatch (int[] block) {

 boolean found = false;

 for (int i=0; i < block.length; i++) {

 for (int j=0; j < block.length; j++) {

found = found ||
 (i != j && block[i]==block[j]);

 }

 }

 return found;

}

If the function f(x) describes the time performance of the hasMatch method, where x denotes the size of the parameter block, which of the following is the smallest big O for f(x)? Why?

1. f(x) is O(1)
2. f(x) is O(log n)
3. f(x) is O(n)
4. f(x) is O(n2)

This is because the inner statement (found = …) is executed n times for the (j) loop and each of those n times for the (i) loop

1. Write the simplest algorithm you can think of to determine whether an integer i exists such that Ai = i in an array, A, of increasing integers. Now, try to give a more efficient algorithm, explaining your reasoning. What is the big O running time for each of your algorithms?

A is increasing means, A[i+1]>A[i] for all i.

The “simplest” solution is to do a sequential search. This algorithm is O(n)

for (int i=0; i<a.length; i++) {

if (a[i]==i) { return i; }

}

return false;

For a faster algorithm, you need to notice two characteristics of this problem:

if A[j]>j then for all k>j, A[k]>k and

if A[j]<j then for all k<j, A[k]<k

That means, if A[j]>j then no need to search the top part of the array (because the answer cannot be there since we know A[k]>k).

And if A[j]<j then no need to search the lower part of the array (because the answer cannot be there).

Therefore, we can solve the A[i]==i problem using an algorithm similar to binary search. This is O(log n).

**public** **static** **int** Problem13(**int**[] a) {

 **int** low = 0;

 **int** high = a.length - 1;

 **int** mid;

 **while** (low <= high) {

 mid = (low + high) / 2;

 if (a[mid] == mid) {

// we've found it

return mid;

} // otherwise keep looking

 **if** (a[mid] < mid) {

//not in lower so now search only upper

 low = mid + 1;

 } **else** **if** (a[mid] > mid) {

//not in upper so now search only lower

 high = mid - 1;

 }

 }

 // if we get here, the item was not found

 **return** *NOT\_FOUND*; // NOT\_FOUND = -1

 }

1. Method hasTwoTrueValues returns true if at least two values in an array of Booleans are true. What is the Big-O running time for all three implementations proposed below?

// Version 1

public boolean hasTwoTrueValues( boolean [ ] arr ) {

int count = 0;

for( int i = 0; i < arr.length; i++ )

 if( arr[ i ] )

 count++;

return count >= 2;

}

Running time:

O(N) where N is the length or arr

// Version 2

public boolean hasTwoTrueValues( boolean [ ] arr ) {

for( int i = 0; i < arr.length; i++ )

 for( int j = i + 1; j < arr.length; j++ )

 if( arr[ i ] && arr[ j ] )

 return true;

return false;

}

Running time:

O(N) where N is the length or arr

// Version 3

public boolean hasTwoTrueValues( boolean [ ] arr ) {

for( int i = 0; i < arr.length; i++ )

 if( arr[ i ] )

 for( int j = i + 1; j < arr.length; j++ )

 if( arr[ j ] )

 return true;

return false;

}

Running time:

O(N2) where N is the length or arr. Version 3 will be faster than Version 2, because the inner loop runs only when arr[i] is true. But they have the same big O.

1. The function methodX searches an array as follows.

public boolean methodX (int[] block) {

 boolean found = false;

 for (int i=0; i<block.length; i++) {

 for (int j=0; j<block.length; j++)

 found = found || block[i]==block[j];

 }

 return found;

}

Which of the following is true of this function? Why?

1. It never returns true.
2. It returns true only if the same item appears twice.
3. It returns true if the last two items compared are the same.
4. It always returns true.

For more Java revision questions see

<http://teaching.csse.uwa.edu.au/units/CITS2200/Tutorials/tutorial02.html>

## Tuesday

### Abstract data types

1. *(from Weiss)* A combination lock has the following basic properties
* the combination (a sequence of three numbers) is hidden.
* the lock can be opened by providing the combination.
* the lock can be closed without requiring the combination.
* the combination can be changed, but only by someone who knows the current combination.

 Suppose you are designing an ADT to represent a combination lock.

1. What operations would you expect this ADT to have ?

The aim here is to implement *your own* idea of the ADT, and to get practice writing Java code – so sample solutions are not given.

1. Specify the behaviour of the ADT by describing (in words) the complete behaviour of each operation.
2. Highlight the error cases that need to be considered for each operation?

Read: <http://teaching.csse.uwa.edu.au/units/CITS2200/Labs/labsheet02.html> which explains how to write and implement interface classes in Java.

1. Write a Java interface for the Lock ADT you have specified.
2. Write a Java implementation for your lock ADT using **an integer** to store the combination.
	1. // An implementation of a Lock using an integer to store the combination
	2. public class LockInt implements Lock {
		1. private int combination;
		2. ...
	3. }
3. **Challenge**: Write a second implementation (LockString) which stores the hidden combination as a String.

### Stacks and Queues

1. On a stack, the result of the sequence push(21) push(14) pop() push(7) pop() is an empty stack?

TRUE or FALSE? Why?

After 3 push and 2 pop there will be 1 element on the stack

1. With a queue, the value 3 is returned by enqueue(1) enqueue(2) enqueue(3) dequeue()?

TRUE or FALSE? Why?

Items are added to the end of the queue, so first dequeue will return 1 (not 3)

1. Show the results of the following sequence:

add(4), add(8), add(1), add(6), remove(), and remove()

when the add and remove operations correspond to the basic operations (push, pop or enqueue, dequeue) in the following:

* 1. Stack

(in a stack you add and remove from front)

4 then 84 then 184 then 6184 then 184 then 84

* 1. Queue

(in a queue you add to back and remove from front)

4 then 48 then 481 then 4816 then 816 then 16

## Wednesday

### First Sorting Algorithms

1. Use the array a to answer the following questions.

int[] a = new int[] { 2,8,9,1,6,3,4,5 }

a. The first element swapped by selection sort is a[4]

TRUE or FALSE ? Why ?

 1 swap with 2 and 1 is at a[3] (arrays start at 0)

 new array is 1,8,9,2,6,3,4,5

b. What is the index of the second element to be swapped?

3

new array is 1,8,9,2,6,3,4,5

swap 8 and 2 (the next smallest)

index of 2 is 3

new array is 1,2,9,8,6,3,4,5

c. What is the index of the third element to be swapped?

5

new array is 1,2,9,8,6,3,4,5

swap 9 and 3 (the next smallest)

index of 3 is 5

1. Which of the following statements about Insertion sort and Selection sort are TRUE and which are FALSE.

(NB: this year, selection sort is not covered)

* 1. There is no difference in running time of Selection sort and Insertion sort since they are both O(N2)   FALSE
	2. Insertion sort is faster than Selection sort because it makes fewer comparisons TRUE
	3. For arrays of the same size, Selection sort has the same number of comparisons for any array order

TRUE because always have to search the whole remaining array to check for the minimum

* 1. For arrays of the same size, Insertion sort has the same number of comparisons for any array order

FALSE if array is ordered then only 1 comparison (and no swaps)

1. **Challenge Question:** Write code to swap the values of two integers in Java **without** using any additional variables (such as temp)?

Ordered

suppose our variables are a and b.

We can write:

 a = a + b;

 b = a – b;

and the variables will be swapped.

### Recursion

***Some Common Errors when using recursion***

*1. The most common error in the use of recursion is forgetting a base case.*

*2. Be sure that each recursive call progresses toward a base case. Otherwise, the recursion is incorrect.*

*3. Overlapping recursive calls must be avoided because they tend to yield exponential algorithms.*

*4. Using recursion in place of a simple loop is bad style.*

*5. Recursive algorithms are analyzed by using a recursive formula. Do not assume that a recursive call takes linear time.*

1. Write a recursive method that calculates factorial of a positive number. Choose a suitable exception for its error cases.

See Java code for week 2 and lecture notes

1. The n-th harmonic number is the sum of the reciprocals of the first n natural numbers. So Hn = 1 + 1/2 + 1/3 + 1/4 + ... + 1/n.

Explain what is *wrong* with each of the following three definitions of a recursive method to calculate the nth Harmonic number? Then write a correct Java implementation and test it.

public static double H(int N) {

 return H(N-1) + 1.0/N;

}

No base case given so this will not terminate

public static double H(int N) {

 if (N == 1) return 1.0;

 return H(N) + 1.0/N;

}

No error checking for N<1 so this will not terminate if N<1.

public static double H(int N) {

 if (N == 0) return 0.0;

 return H(N-1) + 1.0/N;

}

H only defined for N>=1, this has the wrong base case

1. Write a recursive method that returns the number of 1s in the binary representation of N. Use the fact that this number equals the number of 1s in the representation of N/ 2, plus 1, if N is odd.

First: what is the base case ? what is the step case?

Second: express this recursion in a Java method.

Third: write some test cases to test your code.

*[ source: Princeton intro to cs]*

The aim here is to give you practice using Java – no sample solutions are given.

## Thursday

### Recursive Sorting Algorithms

1. You should always choose quicksort or mergesort instead of selection sort or insertion sort, because the first two are faster.

 TRUE or FALSE? Why?

FALSE. For short lists, selection or insertion are fast enough and they are simpler. Also, quicksort and mergesort have some cases with poor performance.

1. Quicksort has many traps for the unwary programmer.
2. What happens if the array is already sorted ?

This is the worst case performance for Quicksort. The pivot always splits the list into 1 and n-1 length so you do not get the benefit of sorting shorter lists.

1. What happens with duplicate elements?

It is ambiguous what should happen to elements equal to the pivot. Part of a good Java implementation is handling this case as efficiently as possible. Both i and j should stop if they encounter an element equal to the pivot. This action turns out to be the only one of the four possibilities that does not take quadratic time for this input. [ See Weiss 8.6.5 Keys equal to the pivot ]

1. Why is finding the pivot a critical step?

Because we want it to split the list into 2 nearly equal parts. A poor choice of pivot can make QuickSort run really slowly.

1. Design and run a set of experiments to compare the running times of selection, insertion sorting algorithms under the following conditions. Using the Java class, SortingAlgorithms.java, provided with your course materials. Add new code to run your experiments.

a. Lists containing the same value in every position.

b. Lists that are already sorted.

c. Lists that have been sorted in reverse order (ie highest to lowest).

d. Integers

e. Real numbers of type double

f. Strings

Write a lab report on the results of your experiments. Which algorithms perform best, under what conditions, and why? And which perform worst, under what conditions and why?

You can base your code on the JUnit tests provided, or use Java's System timing for your experiments like this:

long startTime = System.currentTimeMillis();

callOperationToTime();

long endTime = System.currentTimeMillis();

long totalTime = endTime - startTime;

The aim here is to give you practice designing and testing experiments – no sample solution is provided.

1. **Challenge Problem:** The implementation of Quicksort provided in your notes chooses the fence as the first element in the array. Implement additional code to call QuickSort using one of the (better) methods suggested in the text book for choosing the fence: a. Choose the middle element as pivot  b. Median-of-three pivot. Include the running times of the new version in your experiments above.

Solution TBA.

##  Java revision Questions

*Sources: UWA tutorial01, Weiss*

1. What kind of "thing" is each part of the following Java statement? How does it work?

System.out.println(myObject);

1. What is meant by each part of the statement:

public static void main(String[] args)

1. What is meant by each part of the statement:

import java.util.ArrayList;

1. What is meant by the statement:

import java.io.\*;

1. What is the difference between a *checked* exception and an *unchecked* exception?
2. What options exist for dealing with exceptions?

1. What is the value of red == xxx after these two statements and why?

java.awt.Color red = new java.awt.Color(255,0,0);

java.awt.Color xxx = new java.awt.Color(255,0,0);

1. A reference with the same value as xxx, because an assignment expression has the same value as the expression on the right hand side of the assignment.
2. We cannot tell, because it depends on details of the implementation that are hidden from the users.
3. A runtime error occurs, because == can only be used for primitive types.
4. FALSE, because they are references to different objects, even though the objects contain identical data.
5. TRUE, because they both represent the colour red.
6. Assume there are *size* items in a suitably large array. Which of the following moves all items in an array one place to the right? Why?
	1. for (int i=0; i<size; i++) block[i+1]=block[i];
	2. for (int i=1; i<=size; i++) block[i+1]=block[i];
	3. for (int i=size; i>0; i--) block[i]=block[i-1];
	4. for (int i=size; i>0; i--) block[i-1]=block[i];
7. An array contains N numbers, and you want to determine whether two of the numbers sum to a given number K. For instance, if the input is 8, 4, 1, 6 and K is 10, the answer is yes (4 and 6). A number may be used twice.
	1. Describe (in words) an O(N2) algorithm to solve this problem.
	2. Suppose the list **is sorted**. Describe (in words) an O(N) algorithm to solve this problem.
	3. Describe (in words) an O(N log N) algorithm to solve the problem when the list **is not sorted.**
	4. Write Java code for your solutions for parts a. and c. and compare the running times of your algorithms.
8. **Challenge:** Write a method that removes all duplicates in an array A of N items. The array is passed as a parameter to the method. Return the number of items that remain in A.

Hint: see Q24 for ideas for an O(N log N) algorithm for this problem.