Data Structures and Algorithms
Week 2 problem sheet

A. Linked Lists, stacks and queues

1. Give three points of differences between an array and a linked list.

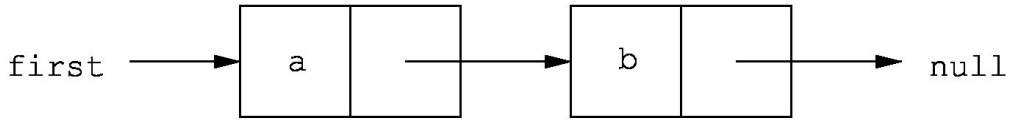
2. Draw the **stack** and **queue** data structures with linked list implementations for each step in the following sequence:

   add(1), add(2), remove, add(3), add(4),
   remove, remove, add(5).

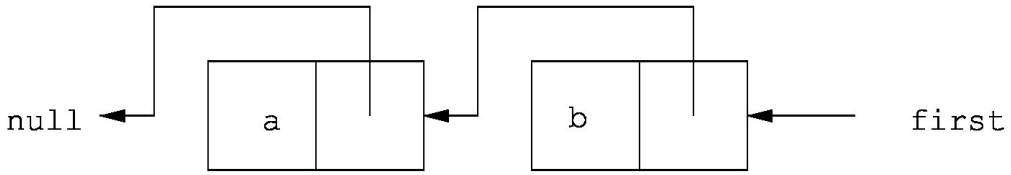
3. Let Link be an object with two member variables:

```
public class Link {
  char item;
  Link next;
}
```

   Assume that the variable `first` is a reference to a Link object containing `'a'`, whose next is a link object containing `'b'`, whose next is `null`.



   Which of the following code snippets successfully reverses this structure, so as to give the following list?



   Draw a picture of the result of each code snippet to explain your answer.

   a.
```
first.next = null;
first.next.next = first;
first = first.next;
```

   b.
```
first.next.next = first;
first = first.next;
first.next.next = null;
```

   c.
```
Link temp = first;
first = temp.next;
first.next = temp;
temp = null;
```

   d.
```
Link temp = first.next;
temp.next = first;
first.next = null;
```

4. A Cyclic Linked Implementation of a Queue

   An (unbounded) queue can be implemented cyclically based on a linked representation. In this case, rather than referencing "`null`", the successor of the last item in the queue references the beginning of the queue. While this is not necessary to prevent memory erosion, it does mean that rather than having two references to the beginning and the end of the queue, only a *single* reference is needed.

   Write a cyclic linked implementation of a queue called `QueueLinked` using this approach.

   Your queue ADT must implement the `QueueADT` interface.

   Fully document your code.

5. A double-ended queue (*deque*) of characters differs from a standard queue in that it allows objects to be added and deleted from both ends of the queue. Contrast this to a standard queue, where objects can only be added to the end of the queue and removed from the front.

   Write a singly linked-list implementation of the deque ADT in Java.

   Your implementation should contain the following methods:

   - `DequeCharCyclic(s)`: create an empty deque of size s.
   - `isEmpty()`: return true iff the deque is empty, false otherwise.
   - `isFull()`: return true iff the deque is full, false otherwise.
   - `pushLeft(c)`: add character c as the left-most character in the deque, or throw an Overflow exception if the deque is full.
   - `pushRight(c)`: add character c as the right-most character in the deque, or throw an Overflow exception if the deque is full.
   - `peekLeft()`: return the left-most character in the deque, or throw an Underflow exception if the deque is empty.
   - `peekRight()`: return the right-most character in the deque, or throw an Underflow exception if the deque is empty.
   - `popLeft()`: remove and return the left-most character in the deque, or throw an Underflow exception if the deque is empty.
   - `popRight()`: remove and return the right-most character in the deque, or throw an Underflow exception if the deque is empty.

6. **Challenge:** See extra questions on dequeues at http://teaching.csse.uwa.edu.au/units/CITS2200/Tutorials/tutorial04.html

B. Big "O" notation

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

   x^2, x, x^2+x, x^2-x, and $(x^{3^i}(x-1))$.

2. 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?

3. Solving a problem requires running an O(N) algorithm and then afterwards an $O(N^2)$ algorithm. What is the total cost of solving the problem? Why?

4. In terms of n, what is the running time of the following algorithm to compute x to the power n (x^n)? 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;
}
```

5. Which of the following statements make sense or not ? Why?

   a. My algorithm is $O(2N^2)$

   b. My algorithm is $O(N^2)$

   c. My algorithm is $O(N^2+N)$

   d. A method with one loop nested inside another must be $O(N^2)$

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

6. 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

7. 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?

a. f(x) is $O(1)$
b. f(x) is $O(\log n)$
c. f(x) is $O(n)$
d. f(x) is $O(n^2)$

8. Write the simplest algorithm you can think of to determine whether an integer i exists such that $A_i = i$ 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?

9. 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?

a. It never returns true.
b. It returns true only if the same item appears twice.
c. It returns true if the last two items compared are the same.
d. It always returns true.

For more Java revision questions see

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

C. Sorting and searching

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?

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

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

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

a. There is no difference in running time of Selection sort and Insertion sort since they are both $O(N^2)$

b. Insertion sort is faster than Selection sort because it makes fewer comparisons
c. For arrays of the same size, Selection sort has the same number of comparisons for any array order
d. For arrays of the same size, Insertion sort has the same number of comparisons for any array order

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

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

   TRUE or FALSE? Why?

D. 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.

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

   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.

```java
public static double H(int N) {
  return H(N-1) + 1.0/N;
}

public static double H(int N) {
  if (N == 1) return 1.0;
  return H(N) + 1.0/N;
}

public static double H(int N) {
  if (N == 0) return 0.0;
  return H(N-1) + 1.0/N;
}
```

3. 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)