1. Concepts – 12 points

1. The method `Blah.foo(x)` takes x milliseconds to execute. Determine the number of activations, the output and running time of the main method to the nearest second.

```
  public static void main(String[] arg) {
        System.out.println( mystery(1));
}
public static int mystery(int v) {
        if(v > 5) {
                Blah.foo(1000);
                return v;
   }
        return mystery(v*2) + mystery(v*2 + 2);

}
```

```
    1
  2   4
4 6 8 10
```

Number of activations of *mystery*: 9 , Output: 42, Running time: 5 seconds.

2. The function below claims to compute the factorial function, `f(x) = x!`
Which one of the following is the best explanation?

E) Factorial values larger than 0x7fffffff suffer from integer overflow.

3. Which one of the following is true?

E) None of the above.

4. "Inserting an item at the beginning of the linked list is a O(1) operation". Explain what this means (where N is length of the list) AND why using a linked list of sprites might make your viral Facebook game require less CPU time than using an array of sprites implementation.

The time required to insert an item into the linked list is constant – it is independent of the length of the linked list. A common alternative implementation is to use an array. To insert a new value would require copying all of the existing items (in a naive implementation) – an O(N) operation that will get slower the larger the list. Using a linked list implementation will require fewer CPU operations (and time) if sprites need to be regularly inserted into the list.

2. Don't look up (Continued)

```
public class PacmanVsTA {
public static void main(String[] args) {
      int tx,ty,px,py;
      boolean bad;
      do {
            tx = TextIO.getlnInt();
            ty = TextIO.getlnInt();
            px = TextIO.getlnInt();
            py = TextIO.getlnInt();
            bad =  tx<0 || ty <0 || px<0 || pz<0;
            if(bad) TextIO.putln("No!"); else TextIO.putln("OK");
      } while(bad);
      int[][] count = new int[12][12];
      int[][] food = new int[12][12];
      for(int i=0;i<12;i++)
            for(int j = 0; j <12;j++)
                  food[i][j] = TextIO.getlnInt();
      int max = solve(count,food,px,py);
      System.out.println("MAX="+max);
      if(count[tx][ty]>0) TextIO.putln("Lookup!")
      else TextIO.putln("SAFE");
}
      public static int solve(int[][] count, int[][] food, int x,int y) {
            if(x<0 || y <0 || x>11 >y>11 || food[x][y] ==0)
                  return 0;
            food[x][y] --;
            int endings = 1 + Math.max(
              Math.max(solve(count,food,x+1,y) , solve(count,food,x-1,y)),
              Math.max(solve(count,food,x,y+1) + solve(count,food,x,y-1)));
            food[x][y] ++;

            if(endings==1)
                  count[x][y]++;
```

```
        return endings;
    }
```

3. Recursion on Linked Lists – 16 points

```
public boolean foo(int acc) {
      if(next == null) return acc*2 == value;
      if(value > acc) return next.foo(value);
      else return next.foo(acc);
}
public int bar() {
      if(value == 0) return 0;
      value *= 2;
      if(next == null) return 1;
      return 1 + next.bar();
}
public Link find(int key) {
      Link temp = null;
      if(next != null) temp = next.find(key);
      if(temp == null & this.value == key) temp = this;
      return temp;
}
public boolean isCopy(Link other) {
      if(other == null || value != other.value) return false;
      if(next==null && other.next != null) return false;
      return next.isCopy(other.next);
}
```

4. Algorithm Analysis – 15 points (3 points each)

```
public static double f1(int N) {
    double result = N;

    for(int i = 1; i < 7; i++)
        for(int j = 0; j < 3; j= j + N*N)
           result  + = j;

    for(int p = N; p >0; p = p / 2)
       result += N * N - p;

    return result;
}
```

O(lg N)

```
 public static int f2(int[] arr, int lo, int hi) {
   // N = initial value of (hi-lo+1) of 1st activation of f2
   if(lo == hi) return lo;
   if(arr[hi] >= arr[lo])
        return f2(arr, lo+1, hi);
   return f2(arr, lo, hi-1);
}
```

O(N)

```
public static int f3(int N) {
   int i = N/5;
   while(i >0)
       i = i - 1;
   return N * N;
}
```

O(N)

```
public static void f4(int N) {
   int a = 0;
   while(a < N * N * N) {
       a = a + N * N;
   }
}
```

O(N)

```
public static void f5(int[] a, int[] b) {
  // N is a.length.
  // Assume a.length = b.length
  int i=0;
  while(i<a.length) {
```

O(N)

```
    if(a[i] < b[i]) {
       b[i] = a[i];
    } else {
        i++;
    }
  } // while
}
```

5. Sorting Algorithms – 12 points

Sorting Algorithm	**Running Time Best Case**	**Running Time Worst Case**
i) **SELECTION**	$O(N^2)$	$O(N^2)$
ii) **MERGE**	iii) **O(N lg N)**	O(N lg N)
iv) **QUICK**	O(N lg N)	$O(N^2)$

1. I use mergesort to sort 32 floating point values, initially in random order, stored in a double array. How many total calls (activations) to mergesort will there be? Your answer: **31**
2.

Initial values:	19	4	40	26	21	17	3	39
Values after 3rd merge:	**4**	**19**	**26**	**40**	21	17	3	39

3. Which one of the following best describes the standard quicksort's pivot choice?

   d) Choosing a median of 3 values reduces the chance of quadratic running time. : **d**

4. I use CS125's selection sort to sort 100 *reverse-sorted* integers {100,99,98...1}. How many 'useless' swaps will our selection sort perform? A useless swap is where the array contents are unchanged because a value is swapped with itself. Your answer: **50**

5. I test my selection sort Java code on a smart phone. I can sort 400,000 randomly ordered integers in ten milliseconds. If I now have 800,000 integers initially in reverse order, approximately how long will the same code take to execute?

   b) 21 – 99 ms Your answer: b (40ms)

6. Quicksort, Mergesort multiple choice – 6 points

i) Uses *findMinimum* [Mergesort only] [Quicksort only] [Both] **[Neither]**
ii) Requires O(N) space. **[Mergesort only]** [Quicksort only] [Both] [Neither]
iii) Worst case has $O(N^2)$ **levels** (depth) of recursion [Mergesort only] [Quicksort only] [Both] **[Neither]**
iv) Best case has O(N) **levels** (depth) of recursion [Mergesort only] [Quicksort only] [Both] **[Neither]**

v) Data is reverse sorted (decreasing values): [A only] [B only] [Both A&B] **[Neither]**
iv) Data is unsorted (randomly shuffled values): [A only] [B only] **[Both A&B]** [Neither]

7. Conditionals and instance variables – 6 points

```
public String choose() {
        if(time<200 && ( purpose =='s' || purpose =='g')) return "Roger";
        if(purpose == 's' || ! food) return "Pete";
        return "John";
}
} // class
```

8. PictureList – 8 Points

```
public PictureList toDeepReverse() {
     PictureList list = new PictureList(size);
     for(int i =0; i < size;i++)
          list.data[i] = new Picture(this.data[size-1-i]);
     return list;
}
public PictureList find(String name) {
     PictureList list = new PictureList(size);
     for(int i =0;i<size;i++)
          if(data[i].getName().equals(name)) list.add( data[i] );
     return list;
}
```

Tricky Bonus Challenge – 3 points

```
public static void main(String[] args) {
        Neuron n = Brain.getBrain();
        n.visitAll();
        n.zeroAll();
}
public boolean visitAll() { // returns true if I'm a keeper
        if(value <0) return value == -42;
        value = - value; // now either -42 or -99
        for(int i =0; i< to.length;i++)
                if( ! to[i].target.visitAll() ) {
                        to[i].target = null;
                                to[i] = null;
                }
        return value == -42;
}
public void zeroAll() {
        if(this.value ==0) return;
        this.value = 0;
        for(int i =0; i< to.length;i++)
                if( to[i].target != null)
                        to[i].target.zeroAll();
}
}
```