Practical 4 – Recursion and Runtime

Q1. Here is an R implementation of Bubble sort.

```
bubblesort<-function(x) {
	if ( (n<-length(x))<2) return(x)
	sorted<-FALSE
	while (!sorted) { #continue if last pass had a swap
		sorted<-TRUE
		for (i in 2:n) { #pass through all adjacent pairs
			if (x[i]<x[i-1]) { #is the pair out of order
				x[i:(i-1)]<-x[(i-1):i] #swap them
				sorted<-FALSE #this pass had a swap
			}
		}
	}
	return(x)
}
```

Modify the `bubblesort` function so that the number of pairs of elements that are compared and the number of pairs that are swapped are returned in a list with the sorted vector. Call this new function `bubblesort1`.

For each of the following 4 vectors use `bubblesort1 to find` the number of pairs of elements that are compared and the number of pairs that are swapped

(a) v1 = c(16, 12, 4, 6, 11, 19, 5, 2, 15, 1, 3, 18, 14, 8, 20, 10, 7, 13, 9, 17)
(b) v2 = 1:2000
(c) v3 = 2000:1
(d) v4 = sample(v2, 2000) [note : this samples 2000 integers without replacement from the vector v2 so you will get a different vector each time you run it.]

How many pairs of elements are compared in the worst case, as a function of the input length n?

Q2. Suppose A,B are n x n matrices and x is an n x 1 vector, and we need the matrix product ABx. How many multiplications are n A(Bx)? How many in (AB)x? Which of these does R use to evaluate `A%*%B%*%x`?

Q3. Pascal's triangle is a geometric arrangement of the binomial coefficients in a triangle.

```
        1
       1 1
      1 2 1
     1 3 3 1
    1 4 6 4 1
  1 5 10 10 5 1
```

….

Entries in each row are determined by summing adjacent numbers in the previous row. The start and end of each row is always 1. Write an R function to calculate the nth row of Pascal's triangle using the idea of recursion.

Q4. Here is an algorithm converting a non-negative integer x to binary.

> [0] If x is one or zero then return x. Otherwise, proceed as follows.
>
> [1] Take the remainder $B0$ when $x0 = x$ is divided by 2. This is the first digit (coefficient of 2^0). (Hint – what do `%%` and `%/%` do?)
>
> [2] Now set $x1 = (x0 - B0)/2$. Repeat this collecting $B1$,$B2$ etc.
>
> [3] the algorithm stops when we have divided x down to $xn = 1$. Set $Bn = 1$ and return the binary number with digits $BnBn{-}1...B1B0$ (Hint if b=f(x) is the decimal-to-binary conversion function you are writing then b=c(f(x1),B0)).

Write a recursive R function implementing this algorithm. Your function should take as input a non-negative integer `x` and return the corresponding binary number. Represent the binary number as a vector, so for example 10 is `c(1,0,1,0)`.

Q5 Implement the following sorting algorithm.

Insertion sort: sort `(x_1,...,x_{n-1})` then take `x_n` and insert it in correct position in the sorted vector. Sort `(x_1,...,x_{n-1})` using Insertion sort!

Show that the worst case number of comparisons in insertion sort is `O(n^2)`.

Q6

(i) Write an R function which simulates birthdays for n people. Assume 365 days in a year, represent dates as integers from 1 to 365, and assume birth-dates are uniformly distributed over the year. Your function should take as input the number n of people and return a vector of length n giving the n dates.

(ii) Write an R function which tests to see if any date is repeated r times or more in a vector of n birthdays. How does the runtime of your function depend on n?

(iii) Write an R function which estimates the probability that r or more people share a birthday in a group of n people, using m simulated sets of n birthdays. Your function should take as input the two integers n and m and return an estimate for the probability that r or more people share a birthday.