COMP 2804 — Solutions Assignment 2

Question 1: On the first page of your assignment, write your name and student number.

Solution:

- Name: James Bond
- Student number: 007

Question 2: Use Newton's Binomial Theorem to prove that for every integer $n \geq 1$,

$$\sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n.$$

Solution: Recall Newton's Binomial Theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

If we take $x = 1$ and $y = 2$, then we get

$$3^n = (1+2)^n = \sum_{k=0}^{n} \binom{n}{k} 1^{n-k} \cdot 2^k = \sum_{k=0}^{n} \binom{n}{k} 2^k.$$

Question 3: In this question, you will give a different proof of the fact that

$$\sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n.$$

Let $A = \{1, 2, 3, \ldots, n\}$ and $B = \{a, b, c\}$. In class, we have see that there are 3^n many functions $f : A \to B$.

- Consider a fixed integer k with $0 \leq k \leq n$ and a fixed subset S of A having size k. How many functions $f : A \to B$ are there for which
  - for all $x \in S$, $f(x) \in \{a, b\}$, and
  - for all $x \in A \setminus S$, $f(x) = c$.

  Solution: For each element x of S, there are 2 choices for $f(x)$. For each element x in $A \setminus S$, there is only 1 choice for $f(x)$. Thus, since S has size k and, therefore, $A \setminus S$ has size $n - k$, the answer is
  $$2^k \cdot 1^{n-k} = 2^k.$$

- Argue that the summation $\sum_{k=0}^{n} \binom{n}{k} 2^k$ counts all functions $f : A \to B$.

  Solution: Consider an arbitrary function $f : A \to B$. For some elements x in A, the value of $f(x)$ is equal to a or b, whereas for the other elements x in A, the value of $f(x)$ is equal to c. Define K_f to be the number of elements x in A for which $f(x)$ is equal to a or b:

  $$K_f = |\{x \in A : f(x) \in \{a, b\}\}|.$$

  This number K_f can be any integer in $\{0, 1, 2, \ldots, n\}$.

  Our first step is to divide all functions $f : A \to B$ into $n + 1$ groups, depending on the value of K_f:

  - Group 0: All functions $f : A \to B$ for which $K_f = 0$.
  - Group 1: All functions $f : A \to B$ for which $K_f = 1$.
  - $\vdots$
  - Group k: All functions $f : A \to B$ for which $K_f = k$.
  - $\vdots$
  - Group n: All functions $f : A \to B$ for which $K_f = n$.

  Observe that any two groups are disjoint. Therefore, by the Sum Rule, the number of functions $f : A \to B$ is equal to

  $$\sum_{k=0}^{n} (\text{the number of functions in group } k). \tag{1}$$

  In the next step, we are going to count how many functions are in group k. Consider a function $f : A \to B$ in this group. We know that there are exactly k elements x in A for which $f(x) \in \{a, b\}$; for all other elements x, we have $f(x) = c$. Define S_f to be the set of all elements x in A for which $f(x) \in \{a, b\}$:

  $$S_f = \{x \in A : f(x) \in \{a, b\}\}.$$

  Since f is in group k, the set S_f has size k. Moreover, S_f can be any subset of A having size k.

  We are going to divide all functions in group k into subgroups:

  - For any subset S of A having size k, there is one subgroup containing all functions in group k for which $S_f = S$.

  In this way, each function in group k belongs to exactly one subgroup.

  - How many subgroups are there: There is one subgroup for every subset of A having size k. Thus, there are $\binom{n}{k}$ many subgroups.

– How many functions are there in one subgroup? From the first part of the question, we know that the answer is 2^k.
– How many functions are in group k? The answer is $\binom{n}{k}2^k$.

This, together with (1) implies that the number of functions $f : A \to B$ is equal to

$$\sum_{k=0}^{n} \binom{n}{k} 2^k.$$

Question 4: In class, we have seen an algorithm for the Gossip Problem; see page 63 of the notes. This recursive algorithm, denoted by GOSSIP(n), computes, for any $n \geq 4$, a sequence of phone calls for the persons $P_1, P_2, \ldots, P_n$.

Give an iterative (i.e., non-recursive) version of this algorithm in pseudocode. Your algorithm must produce exactly the same sequence of phone calls as GOSSIP(n). Justify your answer.

Solution: For $n \geq 5$, the recursive algorithm GOSSIP(n) produces the following sequence of phone calls:

- P_{n-1} calls P_n,
- the sequence of phone calls for $P_1, \ldots, P_{n-1}$ produced by the recursive call GOSSIP($n-1$),
- P_{n-1} calls P_n.

It follows that for $n \geq 4$, GOSSIP(n) produces the following sequence of phone calls ((P_i, P_j) means that P_i calls P_j):

- (P_{n-1}, P_n), (P_{n-2}, P_{n-1}), (P_{n-3}, P_{n-2}), $\ldots$, (P_6, P_7), (P_5, P_6), (P_4, P_5),
- (P_1, P_2), (P_3, P_4), (P_1, P_3), (P_2, P_4),
- (P_4, P_5), (P_5, P_6), (P_6, P_7), $\ldots$, (P_{n-3}, P_{n-2}), (P_{n-2}, P_{n-1}), (P_{n-1}, P_n).

This leads to the following iterative algorithm:

Algorithm ITERATIVE-GOSSIP(n): ($*$ requirement: $n \geq 4$ $*$)
for $k = n$ **downto** 5
do P_{k-1} calls P_k
endfor;
P_1 calls P_2;
P_3 calls P_4;
P_1 calls P_3;

P_2 calls P_4;
for $k = 5$ **to** n
do P_{k-1} calls P_k
endfor

Question 5: The Fibonacci numbers are defined as follows: $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

The following recursive algorithm FIB takes as input an integer $n \geq 0$ and returns the n-th Fibonacci number f_n:

Algorithm FIB(n):
if $n = 0$ or $n = 1$
then $f = n$
else $f =$ FIB$(n-1) +$ FIB$(n-2)$
endif;
return f

Let a_n be the number of additions made by algorithm FIB(n), i.e., the total number of times that the +-function in the else-case is called. Prove that for all $n \geq 0$,

$$a_n = f_{n+1} - 1.$$

Hint: Derive a recurrence for a_n.

Solution: If $n = 0$ or $n = 1$, the algorithm does not enter the else-case and, therefore, the +-function is never called. This means that $a_0 = 0$ and $a_1 = 0$.

Assume that $n \geq 2$.

- There is one call to FIB$(n-1)$, in which the +-function is called a_{n-1} times.
- The +-function is called once in the else case.
- There is one call to FIB$(n-2)$, in which the +-function is called a_{n-2} times.

It follows that $a_n = 1 + a_{n-1} + a_{n-2}$. Thus, we obtain the recurrence

$$\begin{cases} a_0 &= 0, \\ a_1 &= 0, \\ a_n &= 1 + a_{n-1} + a_{n-2} \text{ for } n \geq 2. \end{cases}$$

We now prove by induction that

$$a_n = f_{n+1} - 1$$

for all $n \geq 0$.

If $n = 0$, then $a_n = a_0 = 0$ and $f_{n+1} - 1 = f_1 - 1 = 1 - 1 = 0$. The claim is true for $n = 0$.

If $n = 1$, then $a_n = a_1 = 0$ and $f_{n+1} - 1 = f_2 - 1 = 1 - 1 = 0$. The claim is true for $n = 1$.

Let $n \geq 2$ and assume that for all k with $0 \leq k < n$, $a_k = f_{k+1} - 1$. Then

$$\begin{aligned} a_n &= 1 + a_{n-1} + a_{n-2} \\ &= 1 + (f_n - 1) + (f_{n-1} - 1) \\ &= (f_n + f_{n-1}) - 1 \\ &= f_{n+1} - 1. \end{aligned}$$

Question 6: In this question, we consider finite bitstrings that do not contain 00. Examples of such bitstrings are 0101010101 and 11110111.

For any integer $n \geq 2$, let B_n be the number of bitstrings of length n that do not contain 00.

- Determine B_2 and B_3.
- Prove that $B_n = B_{n-1} + B_{n-2}$ for each $n \geq 4$.
- For each $n \geq 2$, express B_n in terms of a Fibonacci number.

Solution: To determine B_2: There are four bitstrings of length two: 00, 01, 10, and 11. We see that $B_2 = 3$.

To determine B_3: There are eight bitstrings of length three: The ones that contain 00 are: 001, 100, and 000. Thus, $B_3 = 5$.

Let $n \geq 4$. There are B_n bitstrings of length n that do not contain 00. Each of these starts with a 0 or a 1. Let us see how many there are of each type:

- Start with 1: The rest of the string has length $n - 1$ and does not contain 00. Thus, there are B_{n-1} of these.
- Start with 0: Then the second bit is 1. The rest of the string has length $n - 2$ and does not contain 00. Thus, there are B_{n-2} of these.

This shows that

$$B_n = B_{n-1} + B_{n-2}.$$

The Fibonacci numbers are

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

whereas the sequence B_n is

$$3, 5, 8, 13, \ldots$$

The B-sequence is a shifted version of the Fibonacci sequence. We see that, for all $n \geq 2$,

$$B_n = f_{n+2}.$$

Question 7: Let S be the set of ordered triples of integers that are defined in the following way:

- $(66, 55, 1331) \in S$.
- If $(a, b, c) \in S$ then $(a + 7, b + 5, 14a - 10b + c + 24) \in S$.

Prove that for every element (a, b, c) in S,

$$a^2 - b^2 = c.$$

Solution:
Basis: The "base element" in S is $(66, 55, 1331)$. Since $66^2 - 55^2 = 1331$, the base case of the induction is true.

Induction Step: We assume that $(a, b, c) \in S$ and $a^2 - b^2 = c$.

According to the definition of S, $(a + 7, b + 5, 14a - 10b + c + 24)$ is an element of S. We have to show that

$$(a + 7)^2 - (b + 5)^2 = 14a - 10b + c + 24.$$

Here we go:

$$(a + 7)^2 - (b + 5)^2 = (a^2 + 14a + 49) - (b^2 + 10b + 25) = a^2 - b^2 + 14a - 10b + 24.$$

Using $a^2 - b^2 = c$, we obtain

$$(a + 7)^2 - (b + 5)^2 = 14a - 10b + c + 24,$$

which is exactly what we wanted to show.

Question 8: Consider the following recursive algorithm BEER(n), which takes as input an integer $n \geq 1$:

Algorithm BEER(n):
if $n = 1$
then eat some peanuts
else drink one pint of beer;
    choose an arbitrary integer m with $1 \leq m \leq n - 1$;
    BEER(m);
    BEER($n - m$)
endif

(8.1) Explain why, for any integer $n \geq 1$, algorithm BEER(n) terminates.

(8.2) Let $B(n)$ be the number of pints of beer you drink when running algorithm BEER(n). Determine the exact value of $B(n)$.

Solution: We start with the first part. If $n \geq 2$, a call to BEER(n) generates two recursive calls, one to BEER(m) and the other to BEER($n-m$). Since $1 \leq m \leq n-1$, we have

$$1 \leq m < n$$

and

$$1 \leq n - m < n.$$

Thus, each recursive call is on an input which is a positive integer that is strictly less than n. Therefore, after a finite number of calls (deeper in the recursion), the algorithm will reach the base case (which is when the input is 1), in which case it terminates.

Here is an alternative proof by induction: If $n = 1$, then BEER(n) terminates. Let $n \geq 2$ and assume that for each k with $1 \leq k \leq n-1$, BEER(k) terminates. We show that BEER(n) terminates. Running BEER(n) generates two recursive calls:

- BEER(m), where m is an integer with $1 \leq m \leq n-1$. By the induction hypothesis (with $k = m$), this recursive call terminates.
- BEER($n-m$), where m is an integer with $1 \leq m \leq n-1$. Since $1 \leq n-m \leq n-1$, it follows from the induction hypothesis (with $k = n-m$) that this recursive call terminates.

We conclude that BEER(n) terminates.

Now the second part. We are asked to determine the exact value of $B(n)$. The actual computation (i.e., recursion tree) of BEER(n) depends on the value of m that is chosen. As we will see, the number of pints that you drink when running BEER(n) does not depend on the value of m that is chosen.

If you go to your neighborhood pub and run algorithm BEER with several values of n and m, then you will probably guess that

$$B(n) = n - 1.$$

So we think that we know the answer and we can try to prove it by induction:

Base case: If $n = 1$, then you do not drink any pint, so $B(1) = 0$. Since $n - 1 = 0$, the base case is true.

Induction step: Let $n \geq 2$ and assume that $B(k) = k - 1$ for all k with $1 \leq k \leq n-1$. We have to show that $B(n) = n - 1$. Run algorithm BEER(n) and consider the integer m that is chosen. Then it follows from the algorithm that

$$B(n) = 1 + B(m) + B(n-m).$$

- Since $1 \leq m \leq n-1$, we can apply the induction hypothesis with $k = m$. Thus, $B(m) = m - 1$.
- Since $1 \leq m \leq n-1$, we have $1 \leq n-m \leq n-1$. Thus, we can apply the induction hypothesis with $k = n-m$ and obtain $B(n-m) = n-m-1$.

By combining these results, we obtain

$$B(n) = 1 + B(m) + B(n-m) = 1 + (m-1) + (n-m-1) = n-1.$$

By the way, we have just proved the following result: Define a binary tree to be a rooted tree in which each node has zero or two children. A node with zero children is called a leaf, whereas a node with two children is called an internal node. Let T be an arbitrary binary tree with n leaves. Then the value of $B(n)$ is equal to the number of internal nodes of T. Thus, each binary tree with n leaves has exactly $n-1$ many internal nodes.

Question 9: The Hadamard matrices $H_0, H_1, H_2, \ldots$ are recursively defined as follows:

$$H_0 = (1)$$

and for $k \geq 1$,

$$H_k = \left(\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right).$$

Thus, H_0 is a 1×1 matrix whose only entry is 1,

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

and

$$H_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

Observe that H_k has 2^k rows and 2^k columns.

If x is a column vector of length 2^k, then $H_k x$ is the column vector of length 2^k obtained by multiplying the matrix H_k with the vector x.

Describe a recursive algorithm $\textsc{Mult}(k, x)$ that does the following:

Input: An integer $k \geq 0$ and a column vector x of length $n = 2^k$.

Output: The column vector $H_k x$ (having length n).

The running time $T(n)$ of your algorithm must be $O(n \log n)$. Derive a recurrence for $T(n)$. (You do not have to solve the recurrence, because we have done that in class.)
Hint: The input only consists of k and x. The matrix H_k, which has n^2 entries, is not given as part of the input. Since you are aiming for an $O(n \log n)$–time algorithm, you cannot compute all entries of the matrix H_k.

Solution: We will write the vector x as

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

Algorithm $\textsc{Mult}(k, x)$ is a recursive algorithm and does the following:

- If $k = 0$, return the vector (x_1).
- Assume that $k \geq 1$.
  - Split the vector x into two vectors x' and x'', both of length $n/2 = 2^{k-1}$:

$$x' = \begin{pmatrix} x_1 \\ \vdots \\ x_{n/2} \end{pmatrix}$$

    and

$$x'' = \begin{pmatrix} x_{1+n/2} \\ \vdots \\ x_n \end{pmatrix}.$$

  - Run $\textsc{Mult}(k-1, x')$ and let the output be y'.
  - Run $\textsc{Mult}(k-1, x'')$ and let the output be y''.
  - Compute the vector

$$y = \begin{pmatrix} y' + y'' \\ y' - y'' \end{pmatrix}.$$

  - Return the vector y.

Let $T(n)$ denote the running time of algorithm $\textsc{Mult}(k, x)$, where $n = 2^k$. If $k \geq 1$, there are two recursive calls, both of which take time $T(n/2)$, whereas the rest of the algorithm takes $O(n)$ time. Thus, we obtain the "merge-sort recurrence"

$$T(n) = \begin{cases} \text{constant} & \text{if } n = 1, \\ 2 \cdot T(n/2) + O(n) & \text{if } n \geq 2. \end{cases}$$

We have seen in class that this recurrence solves to $T(n) = O(n \log n)$.