COMP 4804 — Questions from previous years

Question 1: Your friend Jim takes a permutation of $1, 2, \ldots, n$, stores it in boxes $B_1, B_2, \ldots, B_n$ (so that each box stores exactly one number), and then closes all boxes. You have no idea what the permutation is.

Jim opens the boxes $B_1, B_2, \ldots, B_n$, one after another. For each i with $1 \leq i \leq n$, just before opening box B_i, you have to guess which number is stored in it.

(1.1) Assume that, when you guess the number in box B_i, you do not remember the numbers stored in $B_1, B_2, \ldots, B_{i-1}$. Then, the only reasonable thing you can do is taking a random element in $\{1, 2, \ldots, n\}$ and guessing that this random element is stored in B_i.

Assume that you do this for each i with $1 \leq i \leq n$. Let X be the random variable whose value is equal to the number of times that your guess is correct. Compute the expected value $E(X)$ of X.

(1.2) Now assume that your memory is perfect, so that, when you guess the number in box B_i, you know the numbers stored in $B_1, B_2, \ldots, B_{i-1}$.

How would you make the n guesses such that the following is true: If Y is the random variable whose value is equal to the number of times that your guess is correct, then the expected value $E(Y)$ of Y satisfies $E(Y) = \Omega(\log n)$. Justify your answer.

Question 2: Recall algorithm InsertionSort, which sorts any input array $A[1 \ldots n]$:

```
Algorithm InsertionSort(A[1...n])
for i = 2 to n
do j = i;
    while j > 1 and A[j] < A[j − 1]
    do swap A[j] and A[j − 1];
        j = j − 1
    endwhile
endfor
```

In the rest of this question, we consider an input array $A[1 \ldots n]$, where each element $A[i]$ is chosen independently and uniformly at random from the set $\{1, 2, \ldots, m\}$.

(2.1) Let i and j be two indices with $1 \leq i < j \leq n$, and consider the values $A[i]$ and $A[j]$ (just before the algorithm starts). Prove that

$$\Pr(A[i] > A[j]) = \frac{1}{2} - \frac{1}{2m}.$$

(2.2) Let X be the random variable whose value is equal to the number of times the swap-operation is performed when running algorithm InsertionSort$(A[1 \ldots n])$. Determine the expected value $E(X)$ of X.

Question 3: You are given a sorted array $A[1 \ldots n]$ of n distinct numbers. The following randomized algorithm receives as input a number x with $A[1] \leq x \leq A[n]$. If x occurs in the array A, then the algorithm returns the location of x in A. Otherwise, the algorithm returns the message "not present".

Algorithm RandomBinSearch(A, x)
$i = 1$;
$j = n$;
while $i < j$
do $k =$ random element in $\{i, i+1, i+2, \ldots, j\}$;
  if $x = A[k]$
  then return k
  else if $x < A[k]$
    then $j = k - 1$
    else $i = k + 1$
    endif
  endif
endwhile;
if $x = A[i]$
then return i
else return "not present"
endif

(3.1) In one iteration of the while-loop, the algorithm chooses a random index k in the set $\{i, i+1, i+2, \ldots, j\}$. We say that the index k is *good*, if

$$i + (j - i + 1)/4 \leq k \leq i + 3(j - i + 1)/4.$$

In words, k is good if it is in the second or third quarter of the subarray $A[i \ldots j]$.

What is the probability that the index k is good?

(3.2) For any integer $\ell \geq 0$, we say that algorithm RandomBinSearch(A, x) is in *phase* ℓ, if

$$(3/4)^{\ell+1} n < j - i + 1 \leq (3/4)^{\ell} n.$$

Thus, at the start of the while-loop, the algorithm is in phase 0.

For a fixed integer $\ell \geq 0$, prove that the expected number of calls in phase ℓ is $O(1)$.

(3.3) Prove that the expected running time of algorithm RandomBinSearch(A, x) is $O(\log n)$.

Question 4: Jim has two children, but we do not know their gender. One day, we meet Jim in the pub, and he tells us that one of his children is a boy. What is the probability that Jim's other child is also a boy? Justify your answer. (In this question, it is assumed that a child is a girl with probability 1/2.)

Question 5: Let S be a set of n distinct numbers and let R be a subset of S. The sorted elements $y_1 < y_2 < \ldots < y_{|R|}$ of R partition the set $S \setminus R$ into $|R| + 1$ subsets, which we call *open intervals*:

$$S_0 = \{x \in S : x < y_1\},$$

$$S_i = \{x \in S : y_i < x < y_{i+1}\}, i = 1, 2, \ldots, |R| - 1,$$

and

$$S_{|R|} = \{x \in S : x > y_{|R|}\}.$$

(If $R = \emptyset$, then there is one open interval S_0, which is equal to S.) If we regard R as being a sample that "represents" S, then the "ideal" sample would have the property that all open intervals have (approximately) the same number of elements. Given an integer r with $1 < r < n$, we can obtain such an "ideal" sample R of size r, in $O(n \log n)$ time: First sort the elements of S, then add each (n/r)-th element to R. (Rounding is ignored here. In fact, using an $O(n)$–time median finding algorithm, R can be computed in $O(n \log r)$ time.)

In this question, we will see a simple randomized algorithm that computes a "good" sample (to be defined below) with probability at least $1/2$, if r is not too small.

For the rest of this question, we fix an integer r with $1 < r < n$. Consider the following algorithm:

Algorithm RandomSample(S, r)
$p = r/n$;
$R = \emptyset$;
for each $x \in S$
do with probability p, add x to R
endfor;
sort the elements of R;
compute the open intervals $S_0, S_1, \ldots, S_{|R|}$;
return $R, S_0, S_1, \ldots, S_{|R|}$

We say that the sample R is *good* if

1. $1 \leq |R| \leq 2r$, and
2. for each i with $0 \leq i \leq |R|$, the open interval S_i contains at most $\frac{2n \ln r}{r}$ elements of S.

Otherwise, the sample R is called *bad*. In words, a good sample R is (i) non-empty, (ii) at most twice as large as the sample size we are aiming for, and (iii) the elements of $S \setminus R$ are approximately evenly distributed over the open intervals (except for the $\ln r$ factor).

(5.1) Compute the expected size $E(|R|)$ of the set R.

(5.2) Prove that

$$\Pr(R = \emptyset) \leq e^{-r}.$$

(*Hint:* Recall that $1 - z \leq e^{-z}$ for all real numbers z.)

(5.3) Use the Chernoff bound to show that

$$\Pr(|R| > 2r) \leq e^{-r/3}.$$

(5.4) Consider the sorted sequence $x_1 < x_2 < \ldots < x_n$ of elements of S. Let k be an integer that divides n (thus, n/k is an integer and no rounding is needed below). Partition S into n/k subsets $B_1, B_2, \ldots, B_{n/k}$, each containing k elements: B_1 contains $x_1, \ldots, x_k$; B_2 contains $x_{k+1}, \ldots, x_{2k}$, etc. We call each subset B_i a *bucket* and say that it is *empty* if $B_i \cap R = \emptyset$.

Argue that the following is true:

- If each bucket is non-empty, then each open interval contains at most $2k$ elements of S.

(5.5) Prove the following:

$$\Pr(\text{ each bucket is non-empty }) \geq 1 - \frac{n}{k}(1-p)^k.$$

(5.6) Argue that

$$\Pr(\text{ each open interval contains at most } 2k \text{ elements of } S\text{ }) \geq 1 - \frac{n}{k}(1-p)^k.$$

(5.7) Recall that $p = r/n$. Let $k = \frac{n \ln r}{r}$. (You may assume that k is an integer that divides n, so that no rounding is needed.) Prove that

$$\Pr\Big(\text{ at least one open interval contains more than } \tfrac{2n \ln r}{r} \text{ elements of } S\Big) \leq \frac{1}{\ln r}.$$

(*Hint:* Recall that $1 - z \leq e^{-z}$ for all real numbers z.)

(5.8) Show that

$$\Pr(\text{ the sample } R \text{ is bad }) \leq e^{-r} + e^{-r/3} + \frac{1}{\ln r}.$$

(5.9) Show the following: If r is chosen sufficiently large, then

$$\Pr(\text{ the sample } R \text{ is good }) \geq \frac{1}{2}.$$

Question 6: Let $\mathcal{A}$ be a Monte Carlo algorithm that, for any given input, outputs either YES or NO. (For example, YES could indicate that the input is a prime number, whereas NO could indicate that the input is not a prime number.) You are told that with probability 2/3, the output is correct.

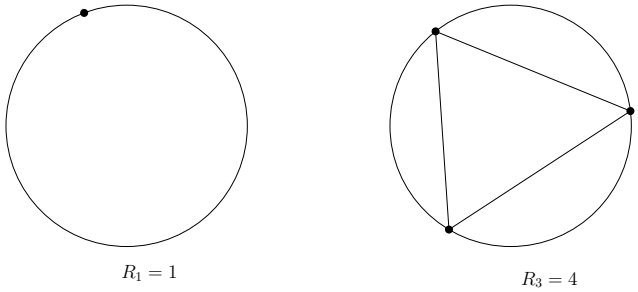
For a large *odd* integer k, consider the following algorithm $\mathcal{B}$:

- Run algorithm $\mathcal{A}$ k times.
- Return the most frequent output obtained in these k runs.

Use the Chernoff bound to show that the output of algorithm $\mathcal{B}$ is correct with probability at least

$$1 - e^{-k/48}.$$

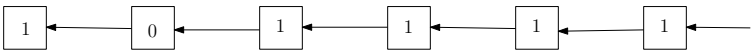
Question 7: For any integer $n \geq 1$, draw n points on a circle. Consider the complete graph on these n points; thus, any two distinct points are connected by an edge which is drawn as a straight-line segment. Assume that the n points are drawn such that no three (or more) edges intersect in one point. This graph divides the interior of the circle into regions. Denote the number of regions by R_n. As the examples below show, we have $R_1 = 1$ and $R_3 = 4$.



(7.1) Determine R_1, R_2, R_3, R_4, and R_5. If you look only at these five values, what do you guess the general formula for R_n is?

(7.2) Argue that your guess in **(7.1)** is completely wrong: Show that the values R_n grow *much slower* than the value that you guessed in **(7.1)**.

Question 8: We consider the problem of maintaining a collection of counters. Each counter C is a list whose nodes store the bits in the binary representation of the value of the counter. This list can be accessed by a pointer to the header, which stores the least significant (i.e., rightmost) bit. Thus, if the value of C is 47, then the counter is the list



Initially, the value of each counter is equal to zero (and the corresponding list is empty). We consider a sequence of operations, where each operation is one of the following:

- INCREMENT(C): Increase the value of the counter C by one (i.e., $C = C + 1$). As in class, the time for INCREMENT is defined to be the number of bits that change. For example, if the value of C is 47, then INCREMENT(C) takes time 5.
- DOUBLE(C): Double the value of the counter C (i.e., $C = 2C$). The time for DOUBLE is defined to be 1, because we just make a new header storing the bit 0.

- ADD(C, C'): Add the values of the counters C and C', store the result in C, and reset C' to zero (i.e., $C = C + C'$; $C' = 0$). In this operation, C and C' are two different counters and the value of C is least that of C'. The time for ADD(C, C') is defined to be the number of positions that are looked at when C and C' are added. Thus, in the following example, the time is equal to 9, because we look at the 9 rightmost positions:

1	0	1	0	1	1	1	1	0	1	0	1
						1	0	1	1	1	1
1	0	1	1	0	0	1	0	0	1	0	0

Use the potential method to prove that any sequence of n operations takes $O(n)$ time.

(*Hint:* The potential function should depend both on the length of a counter and the number of 1s in the counter.)

Question 9: Assume we have some data structure on which we perform a sequence of n operations. For each i with $1 \leq i \leq n$, we denote by c_i the cost of the i-th operation. Assume that

$$c_i = \begin{cases} i & \text{if } i \text{ is a power of two,} \\ 1 & \text{otherwise.} \end{cases}$$

(9.1) Use a payment scheme to prove the best possible upper bound on the total cost of the n operations.

(9.2) Use the potential method to prove the best possible upper bound on the total cost of the n operations.

Question 10: We consider the problem of maintaining a set S under the following operations:

- INSERT(S, x): This operation inserts element x into the set S.
- DELETE50%LARGEST(S): This operation deletes the largest $\lceil |S|/2 \rceil$ elements from the set S.

Design a data structure (and explain the algorithms for INSERT and DELETE50%LARGEST) such that any sequence of n operations takes $O(n)$ time. Use the potential method to prove the time bound. You may assume that the set S is empty at the start of the sequence.

(*Hint:* Recall from COMP 3804 that the median of a set of size m can be computed in $O(m)$ time. You may use this fact. You can explain your algorithms in plain English. Before you come up with the correct potential function, it may help to come up with a payment scheme first.)

Question 11: We are given a set S of real numbers and perform a sequence of operations, each of which is one of the following three types:

- SEARCH(S, x): This operations returns a pointer to the node storing x, if $x \in S$. If $x \notin S$, then this operation returns the message "not present".
- INSERT(S, x): This operation inserts x into S.
- DELETE(S, x): This operation deletes x from S.

Of course, you all know how to obtain an efficient data structure that supports these operations: Store the elements of S in a balanced binary search tree (AVL-tree, red-black tree, ...). Then, each of the three operations takes $O(\log n)$ time, if n denotes the size of S. Thus, any sequence of n operations takes $O(n \log n)$ time, if we start with an empty set S.

As you may remember, however, implementing deletions is a pain. Therefore, we do the following. We choose our favorite balanced binary search tree, and implement the operations SEARCH and INSERT as we learned in class. The operation DELETE(x, S) is implemented in the following way:

- First call SEARCH(x, S). If $x \in S$, then this call returns a pointer to the node storing x. In this case, we *mark* this node and keep it in the data structure. (If $x \notin S$, then there is nothing to do.) If at this moment, at least $|S|/2$ nodes are marked, then we rebuild the tree for all unmarked nodes (which can be done in time which is linear in the number of unmarked nodes).

(11.1) Explain (in plain English) what changes have to be made to the functions SEARCH and INSERT so that they are still correct.

(11.2) Use the potential method to prove that in this new data structure, any sequence of n operations still takes $O(n \log n)$ time. You may assume that the set S is empty at the start of the sequence.

Question 12: Let $G = (V, E)$ be a graph with n vertices and m edges. Consider the following experiment: For each vertex u in V, uniformly at random, pick an element x_u from the set $\{1, 2, 3\}$.

An edge $e = (u, v)$ in E is called *good* if $x_u \neq x_v$. Let X be the number of good edges. Compute the expected value $E(X)$ of the random variable X.

Question 13: You are given a set S of n objects. Each object in S is either *good* or *bad*. You are told that more than $n/2$ of the objects are good; thus less than $n/2$ of them are bad.

For any given object in S, you cannot determine whether it is good or bad, just by "looking at it". Instead, you are given a function SAMETYPE which takes as input two objects x and y, and returns, in $O(1)$ time, the Boolean value

$$\text{SAMETYPE}(x, y) = \begin{cases} \textit{true} & \text{if both } x \text{ and } y \text{ are good,} \\ \textit{true} & \text{if both } x \text{ and } y \text{ are bad,} \\ \textit{false} & \text{if } x \text{ is good and } y \text{ is bad,} \\ \textit{false} & \text{if } x \text{ is bad and } y \text{ is good.} \end{cases}$$

In the questions below, you have to come up with a randomized algorithm that finds a good object in S.

(13.1) Let x be a random object in S. What is the best lower bound on the probability that x is good? Justify your answer.

(13.2) Let x be an arbitrary object in S. Describe, in plain English, a deterministic $O(n)$–time algorithm that decides whether x is good or bad. Justify your answer.

(13.3) Describe, in plain English, a randomized algorithm that returns, in $O(n)$ expected time, a good object in S. Explain why the expected running time of your algorithm is $O(n)$.

Question 14: Let S be a set of n *distinct* numbers which are stored, in random order, in an array $A[1 \ldots n]$. (Thus, this array stores each permutation of S with probability $1/n!$.)

Let m be an integer with $1 < m < n$. Consider the following Monte Carlo algorithm that (attempts to) find the largest element in the set S:

1. By scanning the subarray $A[1 \ldots m]$, compute the largest element in this subarray.
2. Let x be the element found in Step 1. Scan the subarray $A[m+1 \ldots n]$ from left to right, and return the *first* element encountered that is larger than x. In case no element in $A[m+1 \ldots n]$ is larger than x, the number x is returned. In pseudocode,

$i = m + 1$;
while $i \leq n$ and $A[i] < x$
do $i = i + 1$
endwhile;
if $i \leq n$
then return $A[i]$
else return x
endif

Let E be the event

E: "this algorithm returns the largest element in S".

For each i with $m + 1 \leq i \leq n$, let E_i be the event

E_i: "event E holds and the largest element in S is stored at $A[i]$".

(14.1) For each i with $m + 1 \leq i \leq n$, prove that

$$\Pr(E_i) = \frac{m}{n} \cdot \frac{1}{i-1}.$$

(14.2) Let p be the probability that the algorithm returns the largest element in S. Prove that

$$p = \frac{m}{n}\left(1 + H_{n-1} - H_{m-1}\right),$$

where $H_k = \sum_{j=1}^{k} \frac{1}{j}$ is the k-th harmonic number.

Question 15: You are given m balls and n boxes. The balls are thrown independently and uniformly at random in the boxes. Thus, for $1 \leq k \leq m$ and $1 \leq i \leq n$,

$$\Pr(\text{ the } k\text{-th ball falls in the } i\text{-th box }) = 1/n.$$

A box is called *empty* if it does not contain any ball.

(15.1) For a fixed index i, what is the probability that the i-th box is empty?

(15.2) Give a non-trivial upper bound on the probability that at least one box is empty.

(15.3) Give a non-trivial lower bound on the probability that all boxes are non-empty.

(15.4) Assume that $m = 2n \ln n$. Prove that

$$\lim_{n\to\infty} \Pr(\text{ all boxes are non-empty }) = 1.$$

(*Hint:* $1 - x \leq e^{-x}$ for all real numbers x.)

Question 16: Let n be a large positive integer and consider the *complete graph* K_n. This graph has vertex set $V = \{1, 2, \ldots, n\}$, and each pair of distinct vertices is connected by an undirected edge. (Thus, K_n has $\binom{n}{2}$ edges.) Consider the following randomized algorithm, which gives each edge of K_n a random direction:

Algorithm DIRECTEDGES(n):
$\vec{E} = \emptyset$;
for $i = 1$ **to** $n - 1$
do for $j = i + 1$ **to** n
  do flip a fair coin;
    if the coin comes up head
    then add the directed edge (i, j) to $\vec{E}$
    else add the directed edge (j, i) to $\vec{E}$
    endif
  endfor
endfor;
let $\vec{K}_n$ be the directed graph with vertex set V and edge set $\vec{E}$;
return $\vec{K}_n$

We say that three pairwise distinct vertices i, j, and k define a *directed triangle* in $\vec{K}_n$, if

- (i, j), (j, k), and (k, i) are edges in $\vec{K}_n$ or
- (i, k), (k, j), and (j, i) are edges in $\vec{K}_n$.

Let X be the random variable whose value is equal to the total number of directed triangles in $\vec{K}_n$.

(16.1) Prove that $E(X) = \frac{1}{4}\binom{n}{3}$.

(16.2) Argue that this implies that there *exists* a way to direct the edges of K_n, in such a way that the number of directed triangles is at least $\frac{1}{4}\binom{n}{3}$.

(16.3) Let $N = \frac{1}{4}\binom{n}{3}$. Consider again the random variable X. As we have seen in **(16.1)**, we have $E(X) = N$. Define p to be the probability that $X \geq N$, i.e.,

$$p = \Pr(X \geq N).$$

Prove that

$$p \geq \frac{1}{3N + 1}.$$

(*Hint:* Write down the definition of $E(X)$, split the summation into two parts, and deduce an upper bound on each of the two parts. This gives an inequality involving N and p, which is equivalent to the inequality to be shown.)

We next consider the problem of actually computing a direction for each edge of K_n such that, in the resulting directed complete graph $\vec{K}_n$, the number of directed triangles is at least $\frac{1}{4}\binom{n}{3}$. The following algorithm computes such a directed graph:

Algorithm ManyDirectedTriangles(n):
run algorithm DirectEdges(n) and consider the output $\vec{K}_n$;
compute the number X of directed triangles in $\vec{K}_n$;
if $X \geq \frac{1}{4}\binom{n}{3}$
then return $\vec{K}_n$
else ManyDirectedTriangles(n) (i.e., try again)
endif

(16.4) Show that the *expected* running time of algorithm ManyDirectedTriangles(n) is $O(n^6)$. (*Hint:* Computing the value of X can be done, by brute-force, in $O(n^3)$ time. You may use the following fact: If the success probability of an experiment is equal to p, then the expected number of times we have to repeat the experiment until the first success is equal to $1/p$.)

(16.5) Prove that it is possible to give each edge of K_n a direction, such that the number of directed triangles is equal to zero. (*Hint:* This has nothing to do with probability theory.)

Question 17: You are given an array $A[1 \ldots n]$ of n distinct numbers and an integer k with $1 \leq k \leq n$, and you are asked to compute the k-th smallest element in A. Consider the following recursive randomized algorithm:

Algorithm FIND(A, ℓ, r, k):
(**comment:** In this call, $\ell \leq k \leq r$. Thus, the k-th smallest element in the entire array $A[1 \ldots n]$ is contained in the subarray $A[\ell \ldots r]$. This call returns the k-th smallest element in $A[1 \ldots n]$.)

if $\ell = r$
then return $A[\ell]$
else choose a random element p in $A[\ell \ldots r]$;
in $O(r - \ell + 1)$ time, rearrange $A[\ell \ldots r]$ such that $A[i] = p$, all elements in $A[\ell \ldots i-1]$ are less than p, and all elements in $A[i+1 \ldots r]$ are larger than p;
if $k = i$
then return $A[i]$
else if $k < i$
then FIND$(A, \ell, i-1, k)$
else FIND$(A, i+1, r, k)$
endif
endif
endif

The call FIND$(A, 1, n, k)$ returns the k-th smallest element in the entire array $A[1 \ldots n]$.

(17.1) In the call FIND(A, ℓ, r, k), a random pivot p is chosen. We say that the pivot is *good* if, after the elements in $A[\ell \ldots r]$ have been rearranged,

$$\ell + (r - \ell + 1)/4 \leq i \leq \ell + 3(r - \ell + 1)/4.$$

In words, the pivot p is good if, after the rearrangement, p is in the second or third quarter of the subarray $A[\ell \ldots r]$.

What is the probability that the pivot p is good?

(17.2) The call FIND$(A, 1, n, k)$ generates recursive calls of the form FIND(A, ℓ, r, k). We say that such a call is in *phase* j, if $n/2^{j+1} < r - \ell + 1 \leq n/2^j$. (Thus, the initial call FIND$(A, 1, n, k)$ is in phase 0.)

For a fixed integer $j \geq 0$, prove that the expected number of calls in phase j is $O(1)$.

(17.3) Prove that the expected running time of algorithm FIND$(A, 1, n, k)$ is $O(n)$.

Question 18: You are given a list L of elements and want to choose a random element in this list. Each element of L should have the same probability of being chosen. Unfortunately, you do not know the number of elements in L. You are allowed to make only one pass over the list. Consider the following algorithm:

Algorithm ChooseRandomElement(L):
u = first element of L;
$i = 1$;
while u exists
do with probability $1/i$, set $x = u$;
    u = successor of u in L;
    $i = i + 1$
endwhile;
return x

Prove that the output x of this algorithm is indeed a random element of L. In other words, prove the following: Let v be an arbitrary element of L. Then, the probability that $x = v$ after ChooseRandomElement(L) has terminated is equal to $1/n$, where n is the number of elements in L.

Question 19: An *independent set* in a graph is a subset of the vertex set, such that no two vertices in the subset are connected by an edge. Let $G = (V, E)$ be a graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, and assume that every vertex has degree d. Consider the following randomized algorithm which selects a subset I of the vertex set V:

1. Initialize $I := \emptyset$.
2. Flip n fair coins (independently) and denote the outcomes (head or tail) by $f_1, f_2, \ldots, f_n$.
3. For $i = 1, 2, \ldots, n$, do the following: If
   (a) f_i = head and
   (b) f_j = tail for every edge $(v_i, v_j) \in E$ having v_i as a vertex,

   then add vertex v_i to the set I.
4. Return the set I.

(19.1) Argue that the set I that is returned by the algorithm is an independent set in G.

(19.2) Compute the expected value $E(|I|)$ of the size of the independent set I.

For each vertex v_i with $1 \leq i \leq n$, let E_i be the event

$$E_i = \text{“vertex } v_i \text{ is contained in } I\text{”}.$$

(19.3) Let i and j be indices such that (v_i, v_j) is an edge in G. Are the events E_i and E_j independent?

(19.4) Let i and j be indices such that the shortest path in G between v_i and v_j contains two edges. Are the events E_i and E_j independent?

(19.5) Let i and j be indices such that the shortest path in G between v_i and v_j contains at least three edges. Are the events E_i and E_j independent?

(19.6) Let ϵ be a real number with $0 < \epsilon < 1$. Use the Chernoff bound to show that, with high probability, the algorithm returns an independent set I of size at least

$$\frac{(1-\epsilon)}{2^{d+1}(1+d+d^2)}\, n.$$

What is a good lower bound on this probability?

Question 20: Let G be a graph with n vertices and m edges. Consider the value $d = 2m/n$, which is the average vertex degree. Consider the following algorithm:

Step 0: Set $H = G$.

Step 1: For each vertex v of H, with probability $1 - 1/d$, delete (from H) the vertex v together with its incident edges.

Step 2: As long as the graph H contains edges, do the following: Pick an edge (u, v) in H, and remove (from H) the vertex u together with its incident edges.

Step 3: Let I be the vertex set of the graph H. Return I.

(20.1) Argue that the set I that is returned by the algorithm is an independent set in G.

(20.2) Let X be the number of vertices in the graph H after Step 1, and let Y be the number of edges in the graph H after Step 1. Prove that $E(X) = n^2/(2m)$ and $E(Y) = n^2/(4m)$.

(20.3) Let Z be the size of the independent set I that is returned by the algorithm. Prove a lower bound on Z in terms of X and Y. Use this lower bound to show that $E(Z) \geq n^2/(4m)$.

Question 21: Assume we generate a sequence S of n bits by flipping a fair coin n times (the coin flips are independent). A *run* of length k is a consecutive subsequence of S, all of whose bits are the same. (In other words, k times in a row, you generate the same bit.)

(21.1) Assume that n is a power of 2, so that $\log n$ is an integer. Let X be the random variable whose value is equal to the number of runs in S having length $1 + \log n$. Prove that $E(X) = 1 - o(1)$, i.e., $E(X) = 1 - f(n)$, for some function $f(n)$ which converges to zero if n goes to infinity.

(21.2) Assume that n is a power of a power of 2, so that both $\log n$ and $\log\log n$ are integers. Let E be the event

$$E = \text{ “there is no run of length } \log n - 2\log\log n\text{”}.$$

Prove that

$$\Pr(E) \leq 1/n.$$

(*Hint:* Split the sequence of n bits into blocks of length $\log n - 2\log\log n$, and observe that these blocks are independent of each other. Then fool around with the log's and loglog's. You do not need the Chernoff bound.)

Question 22: You are given an array $F[0 \ldots n-1]$ which contains n integers in the range $[0 \ldots m-1]$. This array has the following linearity-property:

- For all integers x and y with $0 \leq x < n$ and $0 \leq y < n$,

$$F[(x+y) \bmod n] = (F[x] + F[y]) \bmod m.$$

One night, Mr. E. Vil shows up and changes $n/5$ of the values in the array $F[0 \ldots n-1]$. You have no idea which values have been changed.

Consider an integer z with $0 \leq z < n$. You would like to know the value of $F[z]$ in the original array. Of course, you have no idea whether or not this value is equal to $F[z]$ in the corrupted array.

Give an algorithm that looks at a small number of locations in the (corrupted) array and returns the correct value of $F[z]$ with probability at least $1/2$. Try to minimize the number of array locations that you access. Justify your answer.

Question 23: We are given a set S of real numbers and perform a sequence of operations, each of which is one of SEARCH(S, x), INSERT(S, x), and DELETE(S, x). We are lazy and store the set S in the following way:

- Let n be the size of the current set S. Write n in binary as

$$n = \sum_{i \geq 0} a_i 2^i,$$

  where each a_i is either zero or one. Partition the set S into subsets such that there is one subset S_i of size 2^i, for each i for which $a_i = 1$. Thus, if

$$n = 87 = 2^0 + 2^1 + 2^2 + 2^4 + 2^6,$$

  then S is partitioned into five subsets having sizes 1, 2, 4, 16, and 64.

- For each i for which $a_i = 1$, we sort the elements of the subset S_i and store the sorted sequence in an array $A_i[1 \ldots 2^i]$.

(23.1) Explain (in plain English) how to perform the operation SEARCH(S, x) in this data structure, in $O(\log^2 n)$ worst-case time.

(23.2) Explain (in plain English) how to perform the operation INSERT(S, x) in this data structure. Use a payment scheme to prove that any sequence of n INSERT-operations takes $O(n \log n)$ time. You may assume that the set S is empty at the start of the sequence.

(*Hint:* Recall that two sorted arrays of lengths m and m' can be merged into one sorted array in $O(m + m')$ time. Set up a payment scheme in which each element pays part of the

costs. If there is an expensive operation that costs T dollars and in which m elements are involved, then let each of these elements pay T/m dollars. Now analyze the total amount of dollars that is paid by each element during the sequence of INSERT-operations.)

Question 24: Recall that an array $A[1 \ldots n]$ is called a *max-heap* if

- $A[i] \geq A[2i]$ for all i with $1 \leq i \leq n/2$, and
- $A[i] \geq A[2i+1]$ for all i with $1 \leq i \leq (n-1)/2$.

The best way to think about a max-heap is by viewing it as a tree; see Section 6.1 in the textbook.

Recall the function MAX_HEAPIFY(A, i) which does the following (see Section 6.2 in the textbook):

- This function assumes that the subtree rooted at the left child of node i is a max-heap, and it assumes that the subtree rooted at the right child of node i is a max-heap.
- This function lets element $A[i]$ float down the tree until the subtree rooted at i is a max-heap.
- If the subtree rooted at i has height h, then the function MAX_HEAPIFY(A, i) takes $O(h)$ time.

Now consider an arbitrary array $A[1 \ldots n]$. The following algorithm converts A into a max-heap (See Section 6.3 in the textbook):

Algorithm BUILD_MAX_HEAP$(A[1 \ldots n])$:
for $i = \lfloor n/2 \rfloor$ **downto** 1
do MAX_HEAPIFY(A, i)
endfor

In COMP 3804, you have seen that the running time of this algorithm is $O(n)$. In this question, you are asked to prove this fact using amortized analysis.

Assume that one call to the function MAX_HEAPIFY(A, i) costs h dollars, where h is the height of the subtree rooted at i.

At the start, we put 2 dollars on each node of the tree; thus, at this moment, the amount of dollars in the tree is $2n$. We are going to show that this amount is sufficient to pay for the entire algorithm.

(24.1) Set up a payment scheme that satisfies the following property: For any node i, let h be the height of its subtree. Then just after the call MAX_HEAPIFY(A, i) has terminated, the subtree rooted at i contains at least h dollars.

For your payment scheme, prove that it satisfies this property (for example, by induction on h).

(24.2) Argue that the $2n$ dollars are sufficient to pay for the entire algorithm.

Question 25: Consider the SETCOVER problem:

- Given a set B of size n, and subsets $S_1, S_2, \ldots, S_m$ of B such that $\cup_{i=1}^{m} S_i = B$.
- Compute a subset I of $\{1, 2, \ldots, m\}$ such that $\cup_{i \in I} S_i = B$.
- The goal is to minimize the size of I.

In class, we have seen that the greedy algorithm computes a set I whose size is at most $\lceil \ln n \rceil$ times the size of the optimal solution. In this question, you have to show that the worst-case approximation factor of *this algorithm* is $\Omega(\log n)$.

Assume that n is a power of two, and let $B = \{1, 2, \ldots, n\}$. Show that there is an input sequence $S_1, S_2, \ldots, S_m$ (for some m) having the following properties:

- The greedy algorithm returns a set I of size $\Omega(\log n)$. (If in one iteration, the greedy algorithm has more than one choice to make, you may assume that it makes the worst possible choice.)
- The optimal solution has size two.

Question 26: Consider the SubsetSum problem:

- Given a sequence $s_1, s_2, \ldots, s_n$ of n positive real numbers and a positive real number B such that
$$\max(s_1, s_2, \ldots, s_n) \leq B$$
and
$$\sum_{i=1}^{n} s_i > B.$$
- Compute a subset I of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in I} s_i \leq B$.
- The goal is to minimize the value
$$D := \max\left(0, B - \sum_{i \in I} s_i\right).$$
In other words, we consider only subsets I for which $\sum_{i \in I} s_i \leq B$.

The problem of deciding if there exists a subset I for which $D = 0$ is **NP**-complete.

Give an $O(n)$–time algorithm that computes an index set I for which $D \leq B/2$. Does this algorithm give a 2-approximation to SubsetSum?

Question 27: In the optimization version of the independent set problem, you are given a graph $G = (V, E)$ and want to find a largest independent set in G. Thus, you want to find a largest possible subset V' of V such that no two vertices in V' are connected by an edge.

Assume that the maximum degree of any vertex in G is equal to d. Let K be the number of vertices in a largest independent set in G. Give a polynomial-time algorithm that, when

receiving the graph G as an input, computes an independent set V' in G whose size is at least $K/(d+1)$. Justify your answer.

Question 28: You are asked to write an article for the popular magazine *The Beer Connoisseur*. In the article, you are supposed to present a fair review of n different beers $B_1, B_2, \ldots, B_n$. You decide to do this in the following way: You try beers in pairs. For each pair (B_i, B_j) that you try, you decide which one you like better, and give your decision a positive weight, where a higher weight indicates that you are more confident of your decision. After having tried many (but not necessarily all $\binom{n}{2}$) pairs of beers, you represent the results by a *directed* graph $G = (V, E)$ whose vertex set is $V = \{B_1, B_2, \ldots, B_n\}$.

- A directed edge $e = (B_i, B_j)$ in E means that beer B_j is better than beer B_i. The weight w_e of this edge e indicates your confidence.

For example, if you try the pair $B_1 =$ Holy Smoke[1] and $B_2 =$ Bud Light, then G will contain the directed edge (B_2, B_1) and this edge has a very large weight.

You plan to include the graph G in your article, but suddenly you notice that G contains directed cycles.

(28.1) Why are readers of the *The Beer Connoisseur* not impressed with a review graph G containing directed cycles?

Rather than doing the entire beer-tasting again, you decide to do the following: You take your directed graph $G = (V, E)$ and compute a subgraph $G' = (V, E')$ such that

1. G' is acyclic (i.e., G' does not contain any directed cycle) and
2. the total weight $\sum_{e \in E'} w_e$ of the edges in G' is maximum.

Thus, the comparisons between pairs of beers that are implied by the subgraph G' are non-conflicting and your confidence for these is as large as possible.

Unfortunately, the decision version of this problem is **NP**-complete, so that you do not have time to compute this acyclic subgraph G' of maximum weight. Since the deadline is approaching, you do the following:

- Recall that the vertex set of the directed graph G is $V = \{B_1, B_2, \ldots, B_n\}$ and the edge set is denoted by E.
- You compute
$$E_1 := \{(B_i, B_j) \in E : i < j\}$$
and
$$E_2 := \{(B_i, B_j) \in E : i > j\}.$$
- If $\sum_{e \in E_1} w_e \geq \sum_{e \in E_2} w_e$, then you define $E'' := E_1$. Otherwise, you define $E'' := E_2$.

[1] available at the Kingston Brewpub, Kingston, Ontario

- You include the graph $G'' = (V, E'')$ in your article. (Note that G'' is acyclic.)

Consider the graph $G'' = (V, E'')$ that is obtained in this way. Let W be the total weight of all edges in a maximum-weight acyclic subgraph of G.

(28.2) Prove that

$$\sum_{e \in E''} w_e \geq W/2.$$

In other words, show that this simple algorithm approximates the optimal solution within a factor of two.

Question 29: Consider the following randomized algorithm:

take a random permutation of $1, 2, \ldots, n$ and store it in the array $A[1 \ldots n]$;
$min = n + 1$;
$x = 0$;
for $i = 1$ **to** n
do if $A[i] < min$
  then $min = A[i]$;
    $x = x + 1$
  endif
endfor;
return x

Consider the variable x that is returned by this algorithm. What is the expected value of x?

Question 30: Let $A[1 \ldots n]$ be an array containing n distinct real numbers, where n is a multiple of 10, and let ϵ be a real number with $0 < \epsilon < 1$. Let x be the $(n/10)$-th smallest element in this array; assume that you do *not know* x.

Give a randomized algorithm that computes, with probability at least $1 - \epsilon$, an element in $A[1 \ldots n]$ which is less than or equal to x. The running time of your algorithm must be $O(\log(1/\epsilon))$.

Question 31: Let $G = (V, E)$ be an undirected graph with n vertices in which each vertex has degree d. Consider the following randomized algorithm, which gives each edge of G a random direction:

Algorithm DIRECTEDGES(G):
$\vec{E} = \emptyset$;
for each edge $\{u, v\}$ in E
do flip a fair coin;
  if the coin comes up head
  then add the directed edge (u, v) to $\vec{E}$

else add the directed edge (v, u) to $\vec{E}$
endif
endfor;
let $\vec{G}$ be the directed graph with vertex set V and edge set $\vec{E}$;
return $\vec{G}$

(31.1) A vertex u is called a *sink* if its outdegree in $\vec{G}$ is equal to zero. What is the probability that u is a sink in $\vec{G}$? (Just give the answer.)

(31.2) Let X be the random variable whose value is equal to the number of sinks in $\vec{G}$. Compute $E(X)$.

(31.3) Assume that $d > \log n$. Prove that

$$\Pr(\ \vec{G} \text{ contains at least one sink }) < 1.$$

(*Hint:* $\Pr(A \text{ or } B) \leq \Pr(A) + \Pr(B)$.)

(31.4) Assume that $d > \log n$. Using **(3.3)**, argue that there *exists* a way to direct the edges of G, in such a way that the resulting directed graph $\vec{G}$ does not contain any sink.

Question 32: You start with an empty set S of real numbers and perform a sequence of INSERT operations. The data structure that you use is randomized and the time for an operation INSERT(S, x) is equal to

$$\begin{cases} |S| & \text{with probability } 1/|S|, \\ c & \text{with probability } 1 - 1/|S|, \end{cases}$$

where $c > 0$ is some constant. Let T be the random variable whose value is equal to the total time for a sequence of n INSERT operations. Compute the expected value $E(T)$ of T.

Question 33: You are given a set S of real numbers and perform a sequence of operations, each of which is of the following two types:

- INSERT(S, x): This operation inserts x into S.
- DELETERANGE(S, a, b): This operation deletes from S all elements x that satisfy $a \leq x \leq b$. (The values a and b need not be the same for all such operations.)

Using a balanced binary search tree, the cost for INSERT(S, x) is equal to $\log |S|$, and the cost for DELETERANGE(S, a, b) is equal to $(1 + k) \log |S|$, where k is the number of elements in S that are between a and b.

Use a payment scheme or the potential method to prove that using this data structure, any sequence of n operations has total cost $O(n \log n)$. You may assume that the set S is empty at the start of the sequence.

Question 34: You are given a set $S = \{s_1, s_2, \ldots, s_n\}$ of real numbers in the interval $(0, 1]$. You are also given identical bins, each having capacity 1. We say that S fits in m bins, if every element of S can be put in one of the m bins in such a way that for each bin, the sum of the elements in this bin is at most 1.

The problem of computing the minimum value of m such that S fits in m bins is **NP**-hard. Consider the following greedy algorithm:

- Start with zero bins.
- For $i = 1, 2, \ldots, n$, do the following:
  - If there is a bin that has space for s_i, then add s_i to this bin. (Thus, after s_i has been added to this bin, the sum of the elements in it is still at most 1.)
  - Otherwise (i.e., if none of the current bins has space for s_i), take a new bin and add s_i to it.

In this question, you will show that the approximation ratio of this algorithm is 2.

Let m be the number of bins used by the greedy algorithm, and let m^* be the optimal solution (i.e., m^* is the smallest number of bins such that S fits in them).

(34.1) Argue that $m^* \geq \sum_{i=1}^{n} s_i$.

(34.2) Consider the m bins that are used by the greedy algorithm. Argue that at most one of these bins is filled to at most 50% of its capacity.

(34.3) Prove that $m^* > (m-1)/2$.

(34.4) Prove that $m \leq 2m^*$.

Question 35: Let $x_1, x_2, \ldots, x_n$ be a sequence of n variables. You are given k equations

$$\begin{aligned}(x_{i_1} + x_{j_1}) \bmod 2 &= b_1\\ (x_{i_2} + x_{j_2}) \bmod 2 &= b_2\\ &\vdots\\ (x_{i_k} + x_{j_k}) \bmod 2 &= b_k\end{aligned}$$

where

- $i_1, \ldots, i_k, j_1, \ldots, j_k \in \{1, 2, \ldots, n\}$,
- $i_1 \neq j_1$, $i_2 \neq j_2$, $\ldots$, $i_k \neq j_k$, and
- $b_1, \ldots, b_k \in \{0, 1\}$.

If you give each of $x_1, \ldots, x_n$ a value in $\{0, 1\}$, then a certain number of the above k equations are satisfied. Let c^* be the maximum number of equations that can be satisfied. For example, if the equations are

$$\begin{aligned}(x_1 + x_2) \bmod 2 &= 0\\ (x_2 + x_3) \bmod 2 &= 1\\ (x_1 + x_3) \bmod 2 &= 0\end{aligned}$$

then $c^* = 2$.

Give a randomized algorithm that gives each of $x_1, \ldots, x_n$ a value in $\{0, 1\}$ such that the following is true: If c is the random variable whose value is the number of equations that are satisfied, then $E(c) \geq c^*/2$. Justify your answer.

Question 36: Let $A[1 \ldots n]$ be an array of n numbers. Consider the following two algorithms, which take as input the array A and a number x. If x is not present in A, then these algorithms return the message "not present". Otherwise, they return an index i such that $A[i] = x$. The first algorithm runs linear search from left to right, whereas the second algorithm runs linear search from right to left.

Algorithm LinearSearchFromLeft(A, x)
$i := 1$;
while $i \leq n$ and $A[i] \neq x$ **do** $i := i + 1$ **endwhile**;
if $i = n + 1$ **then** return "not present" **else** return i **endif**

Algorithm LinearSearchFromRight(A, x)
$i := n$;
while $i \geq 1$ and $A[i] \neq x$ **do** $i := i - 1$ **endwhile**;
if $i = 0$ **then** return "not present" **else** return i **endif**

Consider the following algorithm, which again take as input the array A and a number x. If x is not present in A, then it returns the message "not present". Otherwise, it returns an index i such that $A[i] = x$.

Algorithm RandomLinearSearch(A, x)
flip a fair coin;
if the coin comes up head
then LinearSearchFromLeft(A, x)
else LinearSearchFromRight(A, x)
endif

Assume that the number x occurs exactly once in the array A and let k be the index such that $A[k] = x$. Let T be the random variable whose value is the number of times the test "$A[i] \neq x$" is made in algorithm RandomLinearSearch(A, x). (In words, T is the number of comparisons made by algorithm RandomLinearSearch(A, x).) Determine the expected value $E(T)$ of T.

Question 37: Assume we have n balls and m boxes. We throw the balls randomly in the boxes. Thus, for $1 \leq k \leq n$ and $1 \leq i \leq m$,

$$\Pr(\text{ the } k\text{-th ball falls in the } i\text{-th box }) = 1/m.$$

Define the following three random variables:

1. X: the number of boxes that do not contain any ball.
2. Y: the number of boxes that contain at least one ball.
3. Z: the number of boxes that contain exactly one ball.

(37.1) Determine the expected values $E(X)$, $E(Y)$, and $E(Z)$.

(37.2) Assuming that $m = n$, determine the limits

1. $\lim_{n\to\infty} E(X)/n$,
2. $\lim_{n\to\infty} E(Y)/n$,
3. $\lim_{n\to\infty} E(Z)/n$.

(*Hint:* You may use the fact that $\lim_{n\to\infty}(1 - 1/n)^n = 1/e$.)

Question 38: In this question, you are asked to do the following:

- Prove that every graph with m edges contains a bipartite subgraph with at least $m/2$ edges.
- Give an efficient randomized algorithm that finds such a bipartite subgraph.

Here are the details. Let $G = (V, E)$ be an undirected graph with n vertices and m edges. The edges of E are written as (u, v). Since the edges of E are undirected, (u, v) and (v, u) denote the same edge. If A and B form a partition of V (thus, $V = A \cup B$ and $A \cap B = \emptyset$), then the graph $G' = (V, E')$, where

$$E' = \{(u, v) \in E : u \in A, v \in B \text{ or } u \in B, v \in A\},$$

is bipartite, because every edge in E' connects a vertex of A with a vertex of B.

Using this terminology, this question asks you to do the following:

- Prove that there exists such a partition $A \cup B$ of V for which E' contains at least $m/2$ edges.
- Give an efficient randomized algorithm that finds such a partition.

Consider the following randomized algorithm:

Algorithm RandomPartition(V, E)
$A := \emptyset$;
$B := \emptyset$;
for each vertex u in V
do flip a fair coin;
  if the coin comes up head

then add u to A
else add u to B
endif
endfor;
return the sets A and B

Define the following random variable X:

$$X = |\{(u, v) \in E : u \in A, v \in B \text{ or } u \in B, v \in A\}|.$$

In words, X is equal to the number of edges in the bipartite graph G' that is defined by the sets A and B that are computed by this algorithm.

(38.1) Prove that $E(X) = m/2$.

(38.2) Argue that this implies that there *exists* a partition $A \cup B$ of V for which the corresponding bipartite graph G' contains at least $m/2$ edges. (*Remark:* Notice what we have done. Our goal was to prove that every graph with m edges contains a bipartite subgraph with at least $m/2$ edges. This problem is purely combinatorial, it has *nothing* to do with probability theory. Still, we have *used* probability theory to prove that such a large bipartite subgraph exists.)

We next consider the problem of actually computing a partition $A \cup B$ of V for which the corresponding bipartite graph G' contains at least $m/2$ edges. The following algorithm computes such a partition:

Algorithm LargeBipartiteSubgraph(V, E)
run RandomPartition(V, E) and consider the output (A, B);
compute $X = |\{(u, v) \in E : u \in A, v \in B \text{ or } u \in B, v \in A\}|$;
if $X \geq m/2$
then return the sets A and B
else LargeBipartiteSubgraph(V, E) (i.e., try again)
endif

(38.3) Assume that m is even. Consider again the random variable X in the algorithm. As we have seen in **(38.1)**, we have $E(X) = m/2$. Define p to be the probability that $X \geq m/2$, i.e.,

$$p = \Pr(X \geq m/2).$$

Prove that

$$p \geq \frac{1}{1 + m/2}.$$

(*Hint:* Write down the definition of $E(X)$, split the summation into two parts, and deduce an upper bound on each of the two parts. This gives an inequality involving m and p, which is equivalent to the inequality to be shown.)

(38.4) Show that the *expected* running time of algorithm LargeBipartiteSubgraph(V, E) is $O((n+m)m)$. (*Hint:* You may use the following fact: If the success probability of an experiment is equal to p, then the expected number of times we have to repeat the experiment until the first success is equal to $1/p$.)

Question 39: Consider a sequence $x_0, x_1, x_2, \ldots, x_n, x_{n+1}$ of items. Let

$$L = (x_1, x_2, \ldots, x_n).$$

(Thus, L does not contain the items x_0 and x_{n+1}.) Consider the following randomized algorithm which selects a subsequence L' of L:

Algorithm SelectSubsequence$(x_0, x_1, x_2, \ldots, x_n, x_{n+1})$
flip $n+2$ fair coins;
$L' := \emptyset$;
for $i := 1$ **to** n
do if the i-th coin comes up head and
the $(i-1)$-st coin comes up tail and
the $(i+1)$-st coin comes up tail
then add x_i to L'
endif
endfor;
return L'

Define X to be the random variable whose value is equal to the size of L'. Thus, $X = |L'|$.

(39.1) Compute the expected value $E(X)$.

For each i with $1 \leq i \leq n$, let E_i be the event

$$E_i = \text{ (item } x_i \text{ is contained in } L').$$

(39.2) Let $1 \leq i \leq n-1$. Are the events E_i and E_{i+1} independent? Justify your answer (i.e., give a proof).

(39.3) Let $1 \leq i \leq n-2$. Are the events E_i and E_{i+2} independent? Justify your answer (i.e., give a proof).

(39.4) Let $1 \leq i \leq n-3$. Are the events E_i and E_{i+3} independent? Justify your answer (i.e., give a proof).

(39.5) Let ϵ be a real number with $0 < \epsilon < 1$. Use the Chernoff bound to show that, with high probability, algorithm SelectSubsequence returns a list L' of size at least $(1-\epsilon)n/24$. What is a good lower bound on this probability?

Question 40: Michiel's Bank of Ottawa (MBoO) has installed a new system, which allows customers to access their bank accounts on-line. The innovative feature of the system is the

following: If a customer has a password consisting of n bits, then she can identify herself by sending a string consisting of only $O(\log n)$ bits. The system works as follows:

- MBoO and all customers fix a large integer n and a prime number q with $10^6 n < q < 10^9 n$. (Everybody uses the same values for n and q.)
- Each customer C chooses a password, which is a binary string $P_C = c_0 c_1 \dots c_{n-1}$ of length n. Of course, each customer should not share her/his password with other people.
- MBoO knows the passwords of all customers.

When customer Lingling Qi wants to access his bank account, the following protocol is followed:

Step 1: Mr. Qi sends the following message to MBoO: "The name's Qi, Lingling Qi, I would like to access my account".

Step 2: MBoO chooses a random integer x in $\{0, 1, 2, \dots, q-1\}$ and sends it to Mr. Qi, together with the message "Please send us the number

$$\left(\sum_{i=0}^{n-1} a_i x^i \right) \bmod q,$$

where the coefficients $a_0, a_1, \dots, a_{n-1}$ are the bits in your password".

Step 3: Mr. Qi takes his password $P_{007} = m_0 m_1 \dots m_{n-1}$, computes

$$P_{007}(x) = \left(\sum_{i=0}^{n-1} m_i x^i \right) \bmod q,$$

and sends the number $P_{007}(x)$ to MBoO.

Step 4: MBoO computes $(\sum_{i=0}^{n-1} m_i x^i) \bmod q$ (which they can do, because they known Mr. Qi's password), and compares the result with the number received in Step 3. If the result is the same, then Mr. Qi gets access to his account; otherwise, access is denied.

Observe that during this protocoll, all information that is sent over the internet consists of only $O(\log n)$ bits (even though the password itself consists of n bits). Since Mr. Qi knows his own password, he is obviously able to access his account.

Here is the question: Assume that Dr. No wants to access Mr. Qi's account. Being a good citizen, Mr. Qi has stored his password in a secure place. Thus, Dr. No does not know Mr. Qi's password. All that Dr. No can do is choose some bitstring $k_0 k_1 \dots k_{n-1}$ and use it in Step 3 of the protocoll. Prove that the probability that Dr. No will access Mr. Qi's account is at most $1/10^6$.

(*Hint:* The only place where randomization is used is in Step 2. You may use the following fact: If Q is a polynomial of degree less than n and having integer coefficients, then there are less than n many integers y in $\{0, 1, 2, \dots, q-1\}$ for which $Q(y) = 0 \bmod q$.)

Question 41: Let $G = (V, E)$ be a graph with n vertices and m edges. Consider the following experiment: For each vertex u in V, uniformly at random, pick an element x_u from the set $\{1, 2, 3\}$, and assign x_u to u. An edge $e = (u, v)$ in E is called *good* if $x_u \neq x_v$. Let X be the number of good edges.

(41.1) Compute the expected value $E(X)$ of the random variable X.

(41.2) The Markov inequality states that $\Pr(X \geq t \cdot E(X)) \leq 1/t$. What does this inequality give for $\Pr(X \geq m/2)$? Is this a useful upper bound?

(41.3) Can the Chernoff bound be used to obtain an estimate for $\Pr(X \geq m/2)$? Justify your answer (in one sentence).

(41.4) Prove that $\Pr(X \geq m/2) \geq 1/3$. (*Hint:* Write down the definition of $E(X)$, split the summation into two parts, and deduce an upper bound on each of the two parts.)

Question 42: You are given a Monte Carlo algorithm $\mathcal{A}$ that solves some problem $\mathcal{P}$. This algorithm has the property that, on any input of length n,

- the expected running time is at most $T_{\mathcal{A}}(n)$ and
- the probability that the output of algorithm $\mathcal{A}$ is *wrong* is equal to $3/4$.

You are also given an algorithm $\mathcal{V}$ that does the following:

- $\mathcal{V}$ takes an input of length n, together with the output of algorithm $\mathcal{A}$ on this input. In time $T_{\mathcal{V}}(n)$, algorithm $\mathcal{V}$ decides whether or not $\mathcal{A}$'s output is correct.

Describe a Las Vegas algorithm $\mathcal{B}$ that solves problem $\mathcal{P}$ in $O(T_{\mathcal{A}}(n) + T_{\mathcal{V}}(n))$ expected time. Justify your answer. (*Remark:* The output of $\mathcal{B}$ must always be correct. When describing $\mathcal{B}$, you may use $\mathcal{A}$ and $\mathcal{V}$ as black boxes.)

Question 43: You have won the first prize in the lottery and receive a cheque worth P dollars that you can spend in Michiel's Famous Booze Store (MFBS). As you all know, MFBS sells n beers $B_1, B_2, \ldots, B_n$. For $1 \leq i \leq n$, the price for beer B_i is p_i dollars.

Which beers are you going to buy? Being a beer connoisseur, you take a detailed look at all beers and, for $1 \leq i \leq n$, assign a value v_i to beer B_i. The more you like a beer, the higher the value you assign to it. For example, you give Rochefort[2] a very high value, whereas Bud Light gets a very low value.

Since you want to maximize the total value of the beers that you buy, you want to compute a subset I of $\{1, 2, \ldots, n\}$ for which

- $\sum_{i \in I} p_i \leq P$ (you can spend at most P dollars), and
- $\sum_{i \in I} v_i$ is maximum.

[2] a famous Belgian trappist beer, available in Pub Italia on Preston Street

Let I_{opt} denote the set of indices in an optimal solution, and let V_{opt} denote the corresponding total value; thus,

$$V_{opt} = \sum_{i \in I_{opt}} v_i.$$

How to compute I_{opt}? Having paid attention in COMP 3804, you come up with a dynamic programming algorithm $\mathcal{A}$ that computes I_{opt} in $O(nV)$ time, where $V = v_1 + v_2 + \ldots + v_n$.

Unfortunately, your value of V is very large, so that $O(nV)$ is not polynomial in n. Thus, running $\mathcal{A}$ on the sequence $v_1, \ldots, v_n, p_1, \ldots, p_n, P$ will take too long. After some effort, you manage to prove that computing I_{opt} is **NP**-hard, so that computing I_{opt} exactly is probably out of reach.

After some more effort, you have a brilliant idea: You can use your algorithm $\mathcal{A}$, by running it on smaller numbers obtained by scaling the values $v_1, \ldots, v_n$. Here is your new algorithm:

Algorithm $\mathcal{B}$:
Input: Positive integers $v_1, \ldots, v_n, p_1, \ldots, p_n, P$ and a small real number $\epsilon > 0$. You may assume that $p_i \leq P$ for all $1 \leq i \leq n$.

Step 1: Compute $v_{max} = \max(v_1, v_2, \ldots, v_n)$.

Step 2: For $i = 1, 2, \ldots, n$, compute

$$v'_i = \left\lfloor v_i \cdot \frac{n}{\epsilon \cdot v_{max}} \right\rfloor.$$

Step 3: Run your exact algorithm $\mathcal{A}$ on the sequence $v'_1, \ldots, v'_n, p_1, \ldots, p_n, P$. Let I be the index set returned by this algorithm.

Step 4: Return the index set I computed in Step 3.

(43.1) Show that the running time of algorithm $\mathcal{B}$ is $O(n^3/\epsilon)$.

(43.2) Consider the optimal index set I_{opt} and the corresponding value V_{opt} for the original input $v_1, \ldots, v_n, p_1, \ldots, p_n, P$. Prove that

$$\sum_{i \in I_{opt}} v'_i \geq \frac{n \cdot V_{opt}}{\epsilon \cdot v_{max}} - n.$$

(*Hint:* For all real x, $\lfloor x \rfloor > x - 1$.)

(43.3) Consider the index set I that is returned by algorithm $\mathcal{B}$. Prove that

$$\sum_{i \in I} v_i \geq (1 - \epsilon) V_{opt}.$$

(Thus, in this question, we have obtained a polynomial-time algorithm that computes a $(1 - \epsilon)$-approximation for the original input sequence $v_1, \ldots, v_n, p_1, \ldots, p_n, P$.)

Question 44: Let $G = (V, E)$ be a graph and let $s_1, s_2, \ldots, s_k$ be distinct vertices of V. We want to compute the smallest number of edges in E, whose removal separates $s_1, s_2, \ldots, s_k$, in the following sense: We want the smallest subset E' of E such that, in the graph $(V, E \setminus E')$, no two of $s_1, s_2, \ldots, s_k$ are in the same connected component.

You are given that this problem can be solved in polynomial time for the case when $k = 2$.

Give a polynomial-time algorithm that computes a 2-approximation for the case when $k = 3$. Justify your answer.

Question 45: Recall that an *independent set* in a graph is a subset of the vertex set, such that no two vertices in the subset are connected by an edge.

Let $G = (V, E)$ be a graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, and let d be the maximum degree of any vertex. Consider the following randomized algorithm which selects a subset I of the vertex set V:

1. Initialize $I = \emptyset$.
2. Flip n fair coins (independently) and denote the outcomes (head or tail) by $f_1, f_2, \ldots, f_n$.
3. For $i = 1, 2, \ldots, n$, do the following: If
   (a) $f_i =$ head and
   (b) $f_j =$ tail for every edge $(v_i, v_j) \in E$ having v_i as a vertex,

   then add vertex v_i to the set I.
4. Return the set I.

(45.1) Argue that the set I that is returned by the algorithm is an independent set in G.

(45.2) Let K_{opt} be the size of a largest independent set in G. Prove that the expected size of the independent set I that is computed by the algorithm satisfies

$$E(|I|) \geq \left(\frac{1}{2}\right)^{d+1} K_{opt}.$$

Question 46: Let $G = (V, E)$ be a graph with n vertices. Recall that a *vertex cover* is a subset C of the vertex set V, such that for each edge e in E, at least one of the vertices of e is in C.

- In the *Minimum Vertex Cover Problem*, we want to compute a vertex cover of *minimum* size.

Recall that an *independent set* is a subset I of the vertex set V, such that no two vertices in I are connected by an edge of E.

- In the *Maximum Independent Set Problem*, we want to compute an independent set of *maximum* size.

(46.1) Prove that C is a vertex cover in G if and only if $I = V \setminus C$ is an independent set in G.

Let C_{opt} be a vertex cover in G of minimum size, and let I_{opt} be an independent set in G of maximum size.

(46.2) Argue that $|I_{opt}| = n - |C_{opt}|$.

(46.3) In class, we have seen a polynomial-time algorithm that computes a vertex cover C in G whose size is at most $2 \cdot |C_{opt}|$. Consider the complement $I = V \setminus C$ of this vertex cover, which is an independent set in G.

Is the size of I a good approximation to the size of the largest independent set I_{opt}? In other words, is there a non-trivial lower bound on the ratio

$$\frac{|I|}{|I_{opt}|}?$$

Justify your answer.

Question 47: You are given a large integer n and processes $P_1, P_2, \ldots, P_n$, each of which tries to access a single shared database. We assume that time is being divided into discrete *rounds*. The database can be accessed by at most one process in a single round. Thus, if two or more processes try to access the database simultaneously, then all processes are "locked out" for the duration of that round.

Consider the following algorithm (where T is a large integer):

- In round t, where t runs from 1 to T:
  - Each process P_i ($1 \leq i \leq n$) flips a coin that comes up head with probability $p = 1/n$. Let f_i denote the outcome ("head" or "tail") of P_i's coin.
  - If there is exactly one i such that f_i is "head", then the corresponding process P_i accesses the database.

(47.1) Consider a fixed index i and a fixed round t. Let A_{it} be the event

$$A_{it}\text{: process } P_i \text{ accesses the database in round } t.$$

Determine $\Pr(A_{it})$.

(47.2) Consider a fixed index i. Let NA_i be the event

$$NA_i\text{: process } P_i \text{ does not access the database in any of the rounds } 1, 2, \ldots, T.$$

Prove that

$$\Pr(NA_i) \leq e^{-T/(en)}.$$

(*Hint:* $1 - x \leq e^{-x}$ and for large n, $(1 - 1/n)^{n-1} \geq 1/e$.)

(47.3) Let $T = 2en \ln n$ and let E be the event

E: each process P_i accesses the database at least once during the rounds $1, 2, \ldots, T$.

Prove that

$$\Pr(E) \geq 1 - 1/n.$$

Question 48: When I was a little boy, I collected cards of famous Dutch soccer players. These cards were contained in chewing gum packs, so I had to buy chewing gum until I had all n different cards.

Assume that each pack contains one card, which is randomly chosen out of the total of n different cards. You buy one pack at a time, until you have at least one card of each of the n players.

(48.1) Let j be an integer with $0 \leq j < n$. Assume that you have already j different cards (it may happen that you have more than one card of a particular player). Let X_j be the random variable whose value is equal to the number of packs you buy until you get a card of a new player (i.e., a card that is different from the j cards that you already have). Prove that

$$E(X_j) = \frac{n}{n - j}.$$

(48.2) Let X be the random variable whose value is equal to the total number of packs you buy until you have at least one card of each of the n players. Prove that

$$E(X) = \Theta(n \log n).$$

Question 49: We are given a set S of real numbers and perform a sequence of operations, each of which is one of the following three types:

- SEARCH(x): This operations returns a pointer to the node storing x, if $x \in S$. If $x \notin S$, then this operation returns the message "not present".
- INSERT(x): This operation inserts x into S. You may assume that, prior to this operation, x is not contained in S.

Since we do not want to implement the (complicated) algorithm for inserting an element into a balanced tree, we do the following:

- We store a subset of S in a balanced binary search tree T.
- We store the remaining elements of S in a linked list L (the elements in this list are not sorted).

The operation SEARCH(x) is implemented as follows:

- First search for x in T. If we find x in T, then we return the node storing x. Otherwise, we do a linear search in L. If we find x in L, then we return the node storing x. Otherwise, we return "not present".

The operation INSERT(x) is implemented as follows:

- We add x at the end of the list L.
- If, at this moment, L contains at least $\sqrt{|S|}$ many elements, then we collect all elements in T and L, build a new balanced binary search tree T for all these elements, and set L to the empty list.

Use a payment scheme **or** the potential method to prove that in this new data structure, any sequence of n operations takes $O(n\sqrt{n})$ time. You may assume that the set S is empty at the start of the sequence; thus, both T and L are empty as well.

Question 50: A $(2, 5)$-tree is a tree having the following properties:

- Each non-leaf has 2, 3, 4, or 5 children.
- All leaves are at the same level.

We consider the following two operations:

- ADD(u, v): Given a pointer to a node u, which is the parent of some leaf, give u a new child v.
- REMOVE(v): Given a pointer to a leaf v, delete this leaf.

Describe algorithms (in plain English and using figures) that implement the operations ADD and REMOVE on a $(2, 5)$-tree. The total time for any sequence of n such operations must be $O(n)$.

Use a payment scheme **or** the potential method to prove the $O(n)$ bound on the running time for a sequence of n operations. You may assume that the tree is empty at the start of the sequence.

Question 51: You are given a sequence $s_1, s_2, \ldots, s_n$ of positive integers and a collection $B_1, B_2, \ldots, B_m$ of identical bins, each one having an unbounded capacity.

Assume we distribute the numbers $s_1, s_2, \ldots, s_n$ over the bins, and let T_k be the sum of all numbers that are in bin B_k, $1 \leq k \leq m$. We say that T_k is the *size* of the bin B_k. The problem of computing a distribution for which $\max_{1 \leq k \leq m} T_k$ is minimimum is **NP**-hard. We denote the optimal solution by T_{opt}. Thus, T_{opt} is the smallest possible size of the largest bin in any distribution.

Consider the following greedy algorithm:

- For $k = 1, 2, \ldots, m$, set $T_k = 0$. (Thus, initially, all bins are empty.)
- For $i = 1, 2, \ldots, n$, do the following:
  - Compute an index k for which T_k is minimum.
  - Add element s_i to bin B_k.
  - Set $T_k = T_k + s_i$.

Let T_{opt} be the optimal solution, as defined above, and let T denote the size of the largest bin in the distribution computed by our greedy algorithm. In this question, you will show that $T \leq 2 \cdot T_{opt}$; thus, the approximation ratio of this algorithm is 2.

(51.1) Argue that

$$T_{opt} \geq \frac{1}{m} \sum_{i=1}^{n} s_i.$$

(51.2) Argue that

$$T_{opt} \geq \max(s_1, s_2, \ldots, s_n).$$

(51.3) For $1 \leq k \leq m$, let T_k be the size of bin B_k after our greedy algorithm has terminated. Let B_k be a bin whose size T_k (again, at the end of our algorithm) is equal to T. Let s_i be the last element that was added to bin B_k. Prove that

$$T_1 + T_2 + \ldots + T_m \geq m(T - s_i).$$

(51.4) Prove that $T \leq 2 \cdot T_{opt}$.

Question 52: Consider a list $L = (x_1, x_2, \ldots, x_n)$ of n nodes and assume that each node x_i has a weight w_i, which is a positive real number.

A subset S of the nodes is called an *independent set*, if the following is true for all i:

- If x_i is in S, then x_{i-1} is not in S and x_{i+1} is not in S.

The *weight* of such a subset S is defined to be $\sum_{x_i \in S} w_i$.

Let W_{opt} denote the maximum weight of any independent set S of L.

Give a polynomial-time greedy algorithm that computes an independent set S of L, whose weight is at least $\frac{1}{2} \cdot W_{opt}$. Justify your answer.