

DATA STRUCTURES AND ALGORITHMS COLLECTION OF EXAMS

Albert Atserias

Amalia Duch

Enric Rodríguez Carbonell

(Editors)

February 7, 2017



Departament de Ciències de la Computació
Universitat Politècnica de Catalunya

Contents

1	Mid Term Exams	1
2	Lab Exams	45
3	Final Exams	51
4	Solutions to Mid Term Exams	99
5	Solutions to Lab Exams	107
6	Solutions to Final Exams	147

1

Mid Term Exams

Midterm EDA

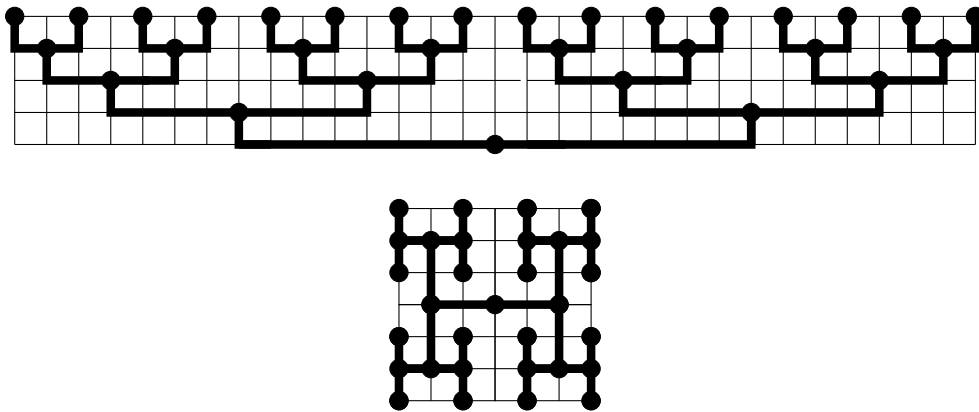
Length: 2 hours

24/3/2014

Problem 1

(3 points)

As graphs, all trees are planar. In particular this means that they are all representable on a sufficiently large rectangular grid in such a way that the edges do not intersect (except at the vertices, obviously). Here are two planar representations of a full binary tree with 16 leaves:



(1 point) Following the recursive pattern of the **first** representation, give recurrences for the height $H(n)$ and width $W(n)$, respectively, of the grid that is needed to represent a full binary tree with 2^n leaves in this representation.

(1 point) Following the recursive pattern of the **second** representation, give the recurrence for the length $L(n)$ of the square grid that is needed to represent a full binary tree with 4^n leaves in this representation.

(1 point) If your goal is to minimize the area of the grid for a full binary tree with 4^n leaves, explain which of the two representations you will choose (to relate both representations note that $4^n = 2^{2n}$). You do not need to solve the recurrences exactly; an asymptotic estimation is enough.

Problem 2**(2 points)**

Consider the following four pieces of code:

```
void f1 (int n) {  
    int x = 2;  
    int y = 1;  
    while (y ≤ n) {  
        y = y + x;  
        x = x + 1;  
    }  
}
```

```
void f3 (int n) {  
    int x = 2;
```

```
void f2 (int n) {  
    int x = 2;  
    int y = 1;  
    while (y ≤ n) {  
        y = y + x;  
        x = 2*x;  
    }  
}
```

```
void f4 (int n) {  
    int x = 2;
```

```
int y = 1;
while (y ≤ n) {
    y = y*x;
    x = 2*x;
}
```

```
int y = 1;
while (y ≤ n) {
    y = y*x;
    x = x*x;
}
```

What is the cost of f_1 ? Answer: $\Theta(\text{ })$

What is the cost of f_2 ? Answer: $\Theta(\text{ })$

What is the cost of f_3 ? Answer: $\Theta(\text{ })$

What is the cost of f_4 ? Answer: $\Theta(\text{ })$

A correct answer without proper reasoning **will not get any points**.

(0,5 points) Reasoning for f_1 :

(0,5 points) Reasoning for f_2 :

(0,5 points) Reasoning for f_3 :

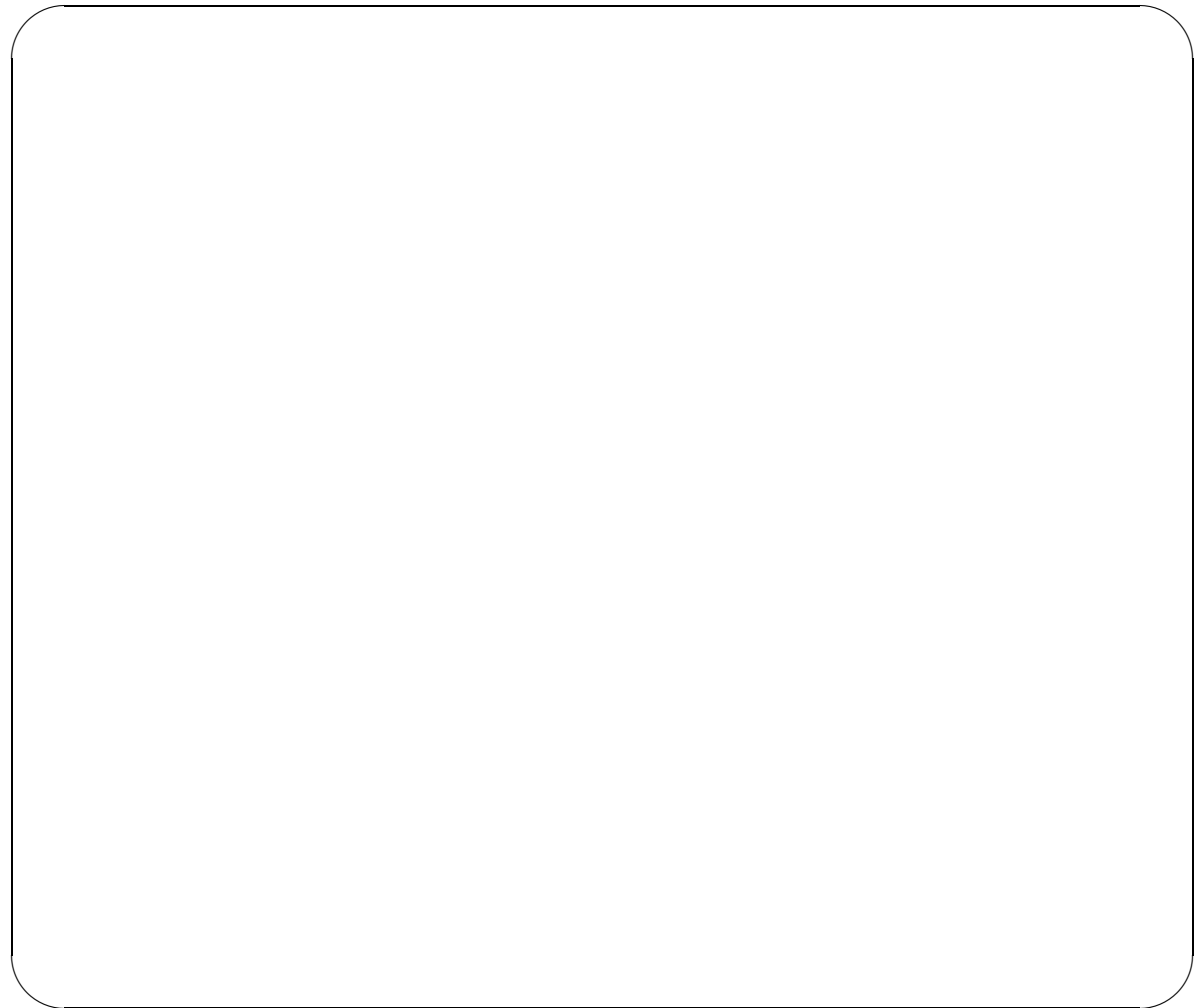


(0,5 points) Reasoning for f_4 :

**Problem 3****(3 points)**

An *inversion* in a vector of integers $T[0 \dots n - 1]$ is a pair of positions storing values that are not ordered, that is, a pair (i, j) such that $0 \leq i < j < n$ and $T[i] > T[j]$.

(1.5 points) Let $T[0 \dots n - 1]$ be a vector of integers. Show that if there is an inversion (i, j) in T , then T has at least $j - i$ inversions.



(1.5 points) Let us fix a value N such that $N \geq 0$ (that is, N is a constant in this exercise). By using Divide & Conquer, implement in C++ a function

bool *search*(**int** x , **const** **vector**<**int**>& T)

which, given an integer x and a vector of integers T of size n with at most N inversions, determines if x is in T . The function should have worst-case cost $\Theta(\log n)$. Show that indeed it has this cost.

Hint: to solve question 2, use the fact from question 1. Besides, deal with subvectors of size $\leq 2N$ in the base case of the recursion, and with the rest of subvectors in the recursive case.

Problem 4**(2 points)**

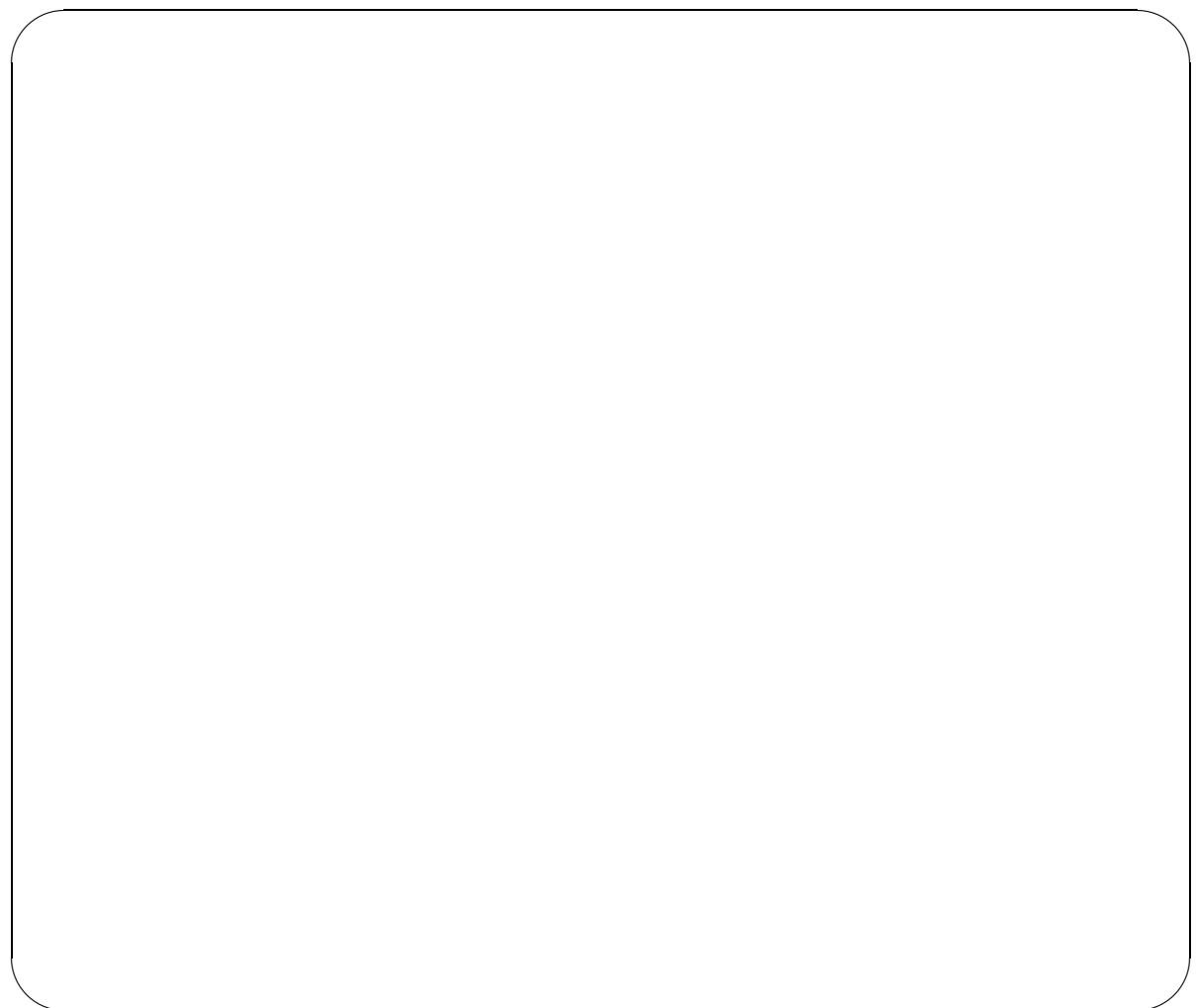
Four quickies:

1. Let $f(n) = n \log(\cos(n\pi) + 4)$. Then $f(n) = \Theta(\text{ })$.
2. Which is the last digit of 7^{1024} written in decimal notation? Answer: .
3. What is the **recurrence** for the cost of Strassen's algorithm to multiply $n \times n$ matrices? Answer $T(n) = \text{ }.$
4. An algorithm admits all 2^n vectors of n bits as possible inputs. On $2^n - 1$ of these inputs the cost of the algorithm is $\Theta(n^2)$, and on the remaining one the cost is $\Theta(n^4)$. Therefore, its worst-case cost is $\Theta(n^4)$ and its best-case cost is $\Theta(n^2)$. What is its average-case cost when the input is chosen uniformly at random (so each input has

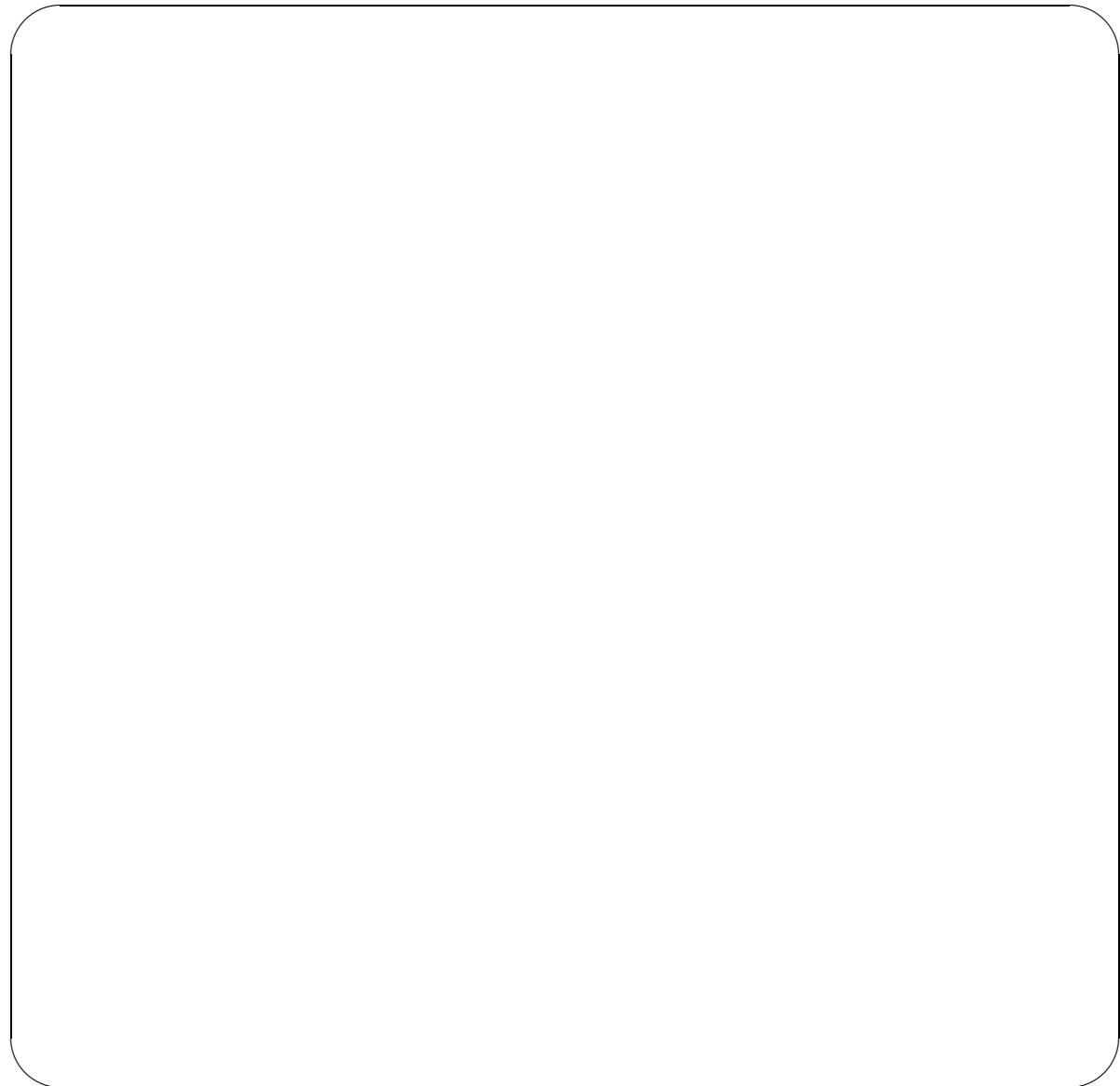
Midterm EDA**Length: 2.5 hours****13/10/2014****Problem 1****(2 points)**

(1 point) Prove that $\sum_{i=0}^n i^3 = \Theta(n^4)$.

Hint: There are several ways to do it, one of them by proving O and Ω separately.



(1 punt) Let $f(n) = 2\sqrt{\log n}$ and $g(n) = n$. Asymptotically, which one grows faster? Prove it.

**Problem 2****(3 points)**

Consider the procedure *mystery* which takes as inputs a vector A and a positive integer k such that all elements of A are between 0 and k , both included.

```
void mystery(const vector<int>& A, vector<int>& B, int k){
    int n = A.size ();
    vector<int> C(k+1, 0);
    for (int j = 0; j < n; ++j) ++C[A[j]];
    for (int i = 1; i ≤ k; ++i) C[i] += C[i-1];
    B = vector<int>(n);
    for (int j = n-1; j ≥ 0; --j){
        B[C[A[j]] - 1] = A[j];
        --C[A[j]];
    }
}
```

(a) (1 point) What is the content of B at the end of the execution of *mystery*?

(b) (1 point) If n is the length of the vector A and you are guaranteed that $k = O(n)$, what is the asymptotic cost of *mystery* as a function of n ?

(c) (1 point) Without writing any code, describe an algorithm for the following problem: given a positive integer k , a vector A of n integers between 0 and k , both endpoints included, and a vector P of m pairs of integers $(a_0, b_0), \dots, (a_{m-1}, b_{m-1})$, with $0 \leq a_i \leq b_i \leq k$ for each $i = 0, \dots, m-1$, you are asked to write the number of elements of A which are in the interval $[a_i, b_i]$ for $i = 0, \dots, m-1$ in time $\Theta(m + n + k)$, and in particular time $\Theta(m + n)$ when $k = O(n)$.

Hint: using a vector C like the one in *mystery* will be useful.

Problem 3**(3 points)**

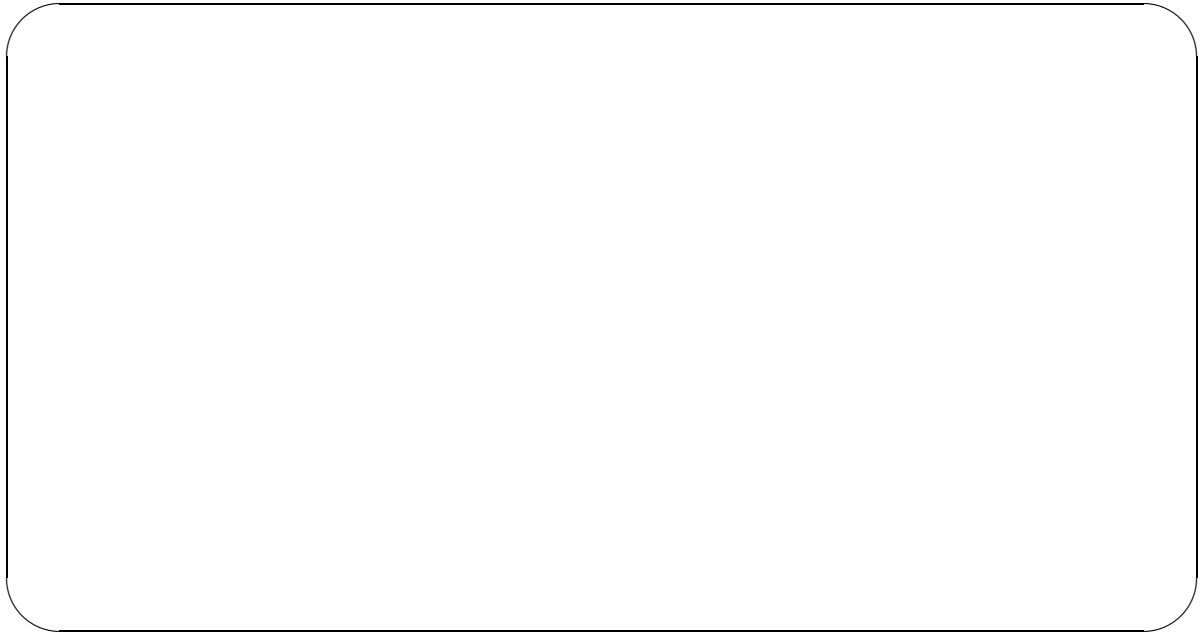
A board game of chance has 64 possible positions $1, \dots, 64$. The rules of the game are such that, from any position $i \in \{1, \dots, 64\}$, in one step a checker will hop to any other position $j \in \{1, \dots, 64\}$ with probability $P_{i,j}$. Let $P = (P_{i,j})_{1 \leq i,j \leq 64}$ be this matrix of probabilities.

(a) (1 point) Remember that if $R = P^2$, then $R_{i,j} = \sum_{k=1}^{64} P_{i,k} P_{k,j}$ for each i, j . Taking into account that $P_{i,j}$ is the probability of hopping from i to j in one step, how do you interpret $R_{i,j}$?

(b) (2 points) Design an algorithm that, given a matrix of probabilities $P = (P_{i,j})_{1 \leq i,j \leq 64}$ and given a certain number of steps $t \geq 0$, computes the matrix $(Q_{i,j})_{1 \leq i,j \leq 64}$ where each $Q_{i,j}$ is the probability that the checker, starting at position i , ends at position j after exactly t steps.

```
typedef vector<double> Row;
typedef vector<Row> Matrix;
```

```
void probabilities (const Matrix& P, int t, Matrix& Q) {
```

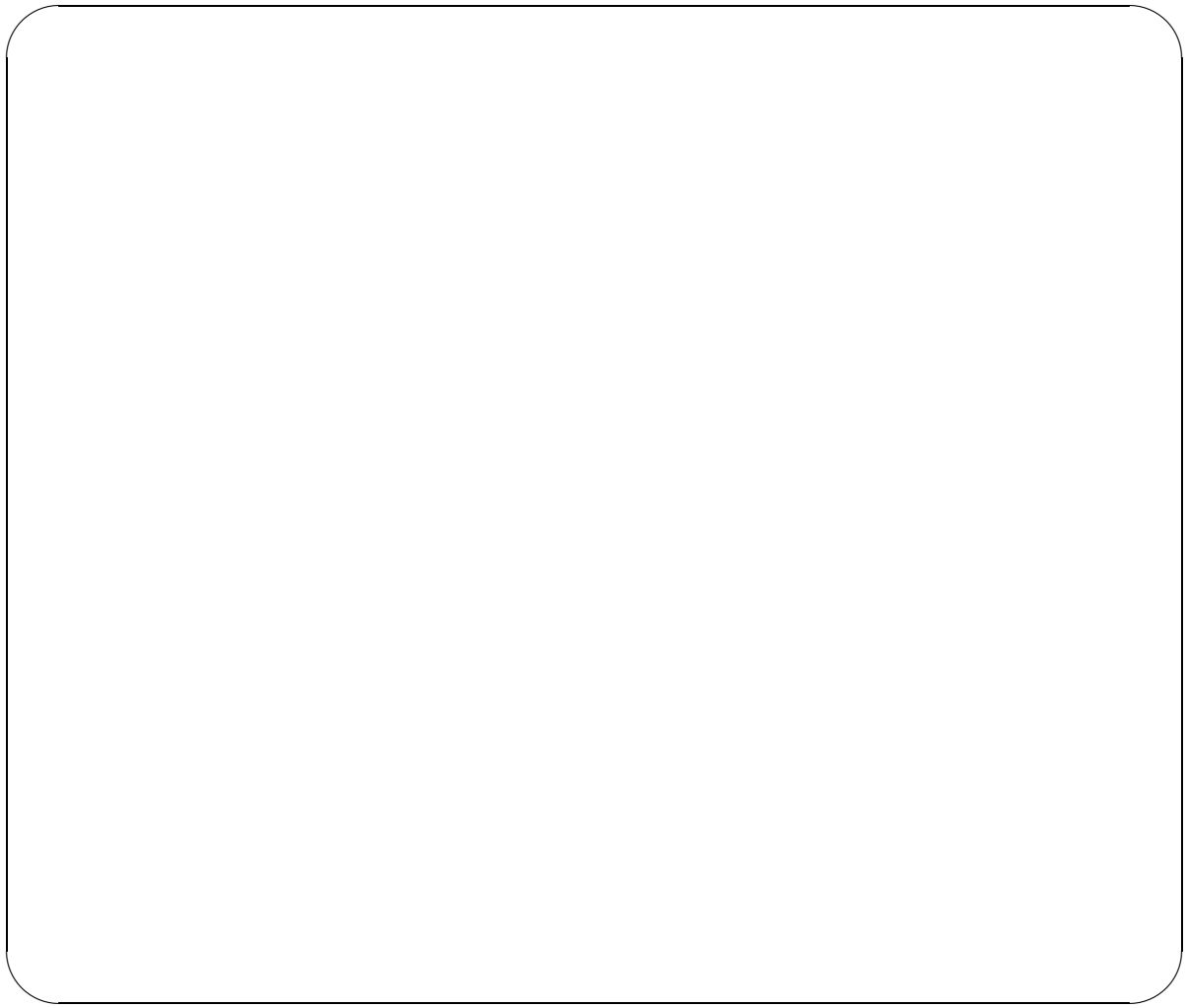
}

Note 1: In order to get full grade your algorithm must run in $\Theta(\log t)$ time.

Note 2: If you need any auxiliary function, implement on the back.

Note 3: Justify the cost of your algorithm on the back.

Auxiliary functions (if needed):



Cost of the algorithm:



Problem 4

(2 points)

(a) (1 point) Assume that an algorithm receives as input any of the 2^n vectors of n bits with equal probability. Assume that the worst-case cost of the algorithm is $\Theta(n^2)$, and that its best-case cost is $\Theta(n)$. How many inputs are required to cause cost $\Theta(n^2)$ to be sure that the average-case cost of the algorithm is $\Theta(n^2)$?

Answer: $\Omega($ $)$

Justification:

(b) (0.5 points) Consider the following algorithm that adds one unit to a natural number that comes represented by a vector **vector<int>** A of n decimal digits:

```

int  $i = n - 1$ ;
while ( $i \geq 0$  and  $A[i] == 9$ ) {  $A[i] = 0$ ;  $--i$ ; }
if ( $i \geq 0$ )  $++A[i]$ ; else cout << "Overflow" << endl;

```

If every digit $A[i]$ of the input is equally likely and independent of the rest (that is, each $A[i]$ is one of the 10 possible digits with probability $1/10$ independently of the other digits), what is the probability that this algorithm makes exactly k iterations when $0 \leq k \leq n - 1$?

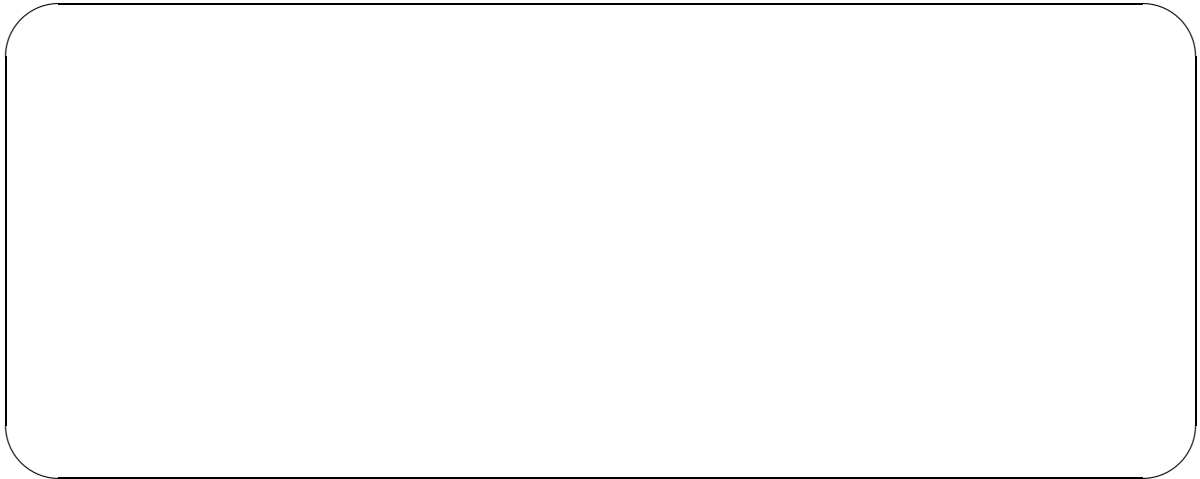
Answer:

Justification:

(c) (0.5 points) Use the master theorem for subtractive recurrences to solve the recurrence $T(n) = \frac{9}{10}T(n-1) + \Theta(1)$ with base case $T(0) = \Theta(1)$.

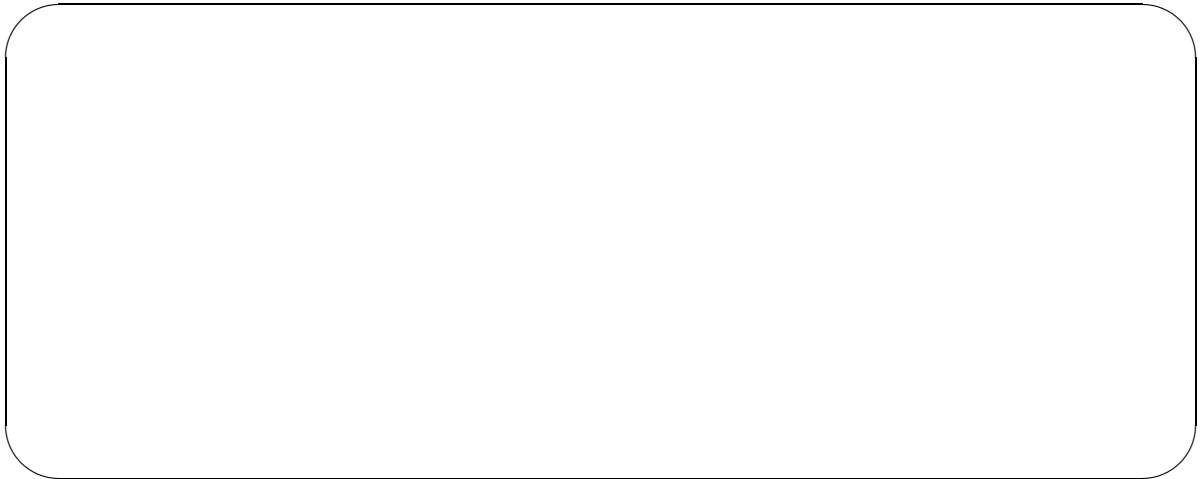
Answer: $T(n) = \Theta($ $)$

Justification:



————— END OF THE EXAM —————

(d) (**optional** extra bonus: 1 additional point to the grade) What does the recurrence $T(n)$ from section (c) compute about the algorithm from section (b)?



Midterm EDA Length: 2h30m

23/3/2015

Problem 1: Cost analysis

(2 points)

The sieve of Eratosthenes (276–194 BC) is a method to generate all prime numbers smaller or equal than a given n . The method goes as follows: traversing the sequence of numbers $2, 3, 4, \dots, n$, look for the next number x that is not yet marked, mark all its multiples $2x, 3x, 4x, \dots$ up to n , and repeat with the next x . When it finishes, those $x \geq 2$ that are not marked are the prime numbers. In C++:

```
vector<bool> M(n + 1, false);
for (int x = 2; x ≤ n; ++x) {
    if (not M[x]) {
        for (int y = 2*x; y ≤ n; y += x) M[y] = true;
    }
}
```

For the first three questions below we ask for an exact (non-asymptotic) expression *as a function of n* . If needed, use the notation $\lfloor z \rfloor$ that rounds z to the maximum integer smaller or equal than z ; for example, $\lfloor \pi \rfloor = \lfloor 3.14\dots \rfloor = 3$.

(a) (0.33 points) How many times is $M[y] = \text{true}$ executed when x is 2?

Answer: Exactly many times.

(b) (0.34 points) How many times is $M[y] = \text{true}$ executed when x is 15?

Answer: Exactly many times.

(c) (0.33 points) How many times is $M[y] = \text{true}$ executed when x is 17?

Answer: Exactly many times.

Note: More questions on the back.

(d) (0.5 points) It is known that

$$\sum_{\substack{p=2 \\ p \text{ prime}}}^n \frac{1}{p} = \Theta(\log \log n).$$

Use this, in conjunction with the answers to the previous questions, to determine the cost of the algorithm as a function of n , in asymptotic notation. Answer: $\Theta(\text{ })$.

Justification:

(e) (0.5 points) An improvement consists in replacing the condition $x \leq n$ in the external loop by $x * x \leq n$. Would this improve the asymptotic cost?

Answer and justification:

Problem 2: Strassen & company

(2 points)

The school algorithm to multiply two $n \times n$ matrices makes $\Theta(n^3)$ arithmetic operations. In 1969 Strassen found an algorithm that makes $\Theta(n^{2.81})$ arithmetic operations. Twenty one years later, Coppersmith and Winograd discovered a method that makes $\Theta(n^{2.38})$ operations.

Assuming (for simplicity) that the implicit constants in the Θ notation are 1, 10 and 100, respectively, and that they apply to every $n \geq 1$ (that is, the costs are n^3 , $10n^{2.81}$ and $100n^{2.38}$, respectively, for every $n \geq 1$), compute the least n for which one of these algorithms makes less operations than another.

(a) (1 point) For $n \geq$, Strassen improves the school method.

(b) (1 punt) For $n \geq$, Coppersmith-Winograd improves Strassen.

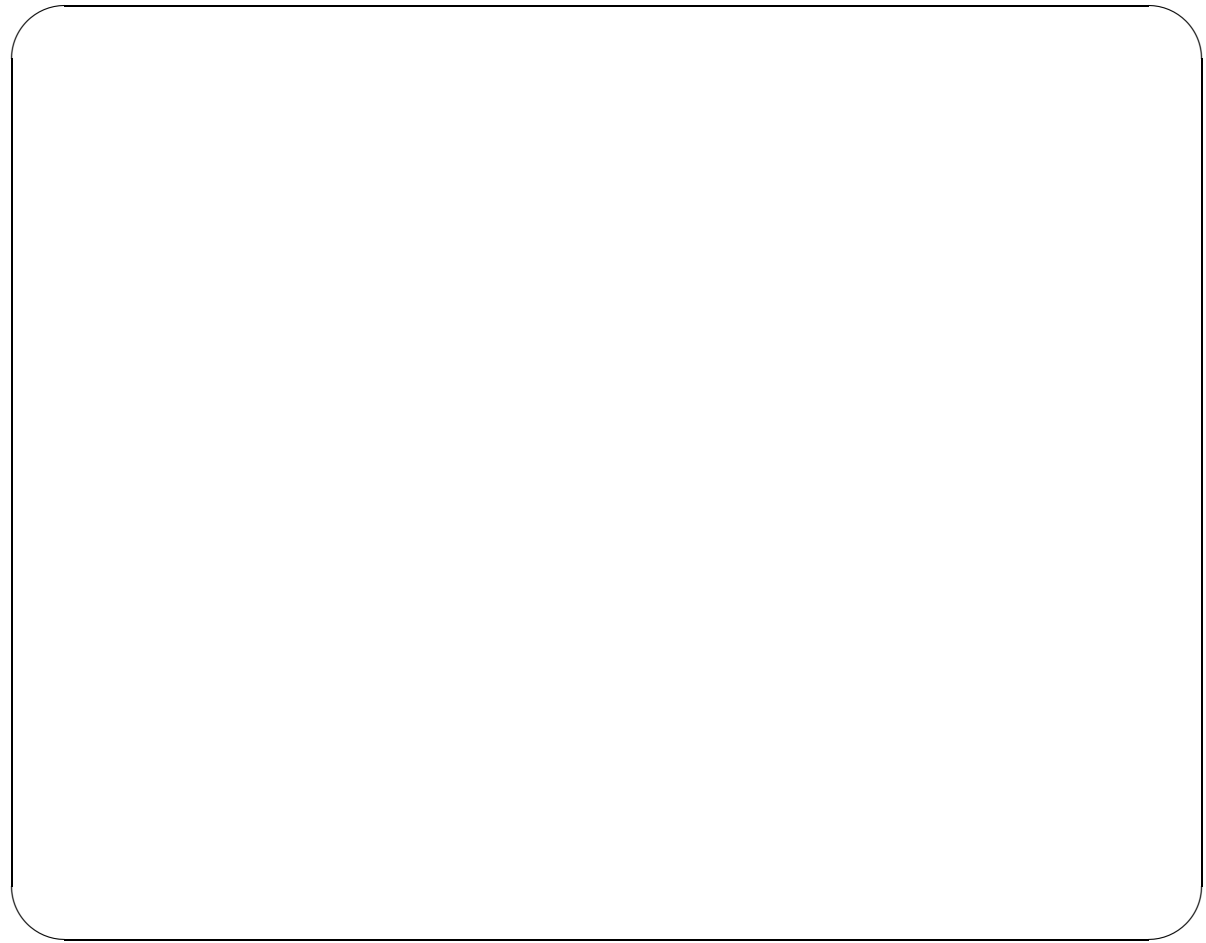
Justifications:

Problem 3: Algorithm design

(3 points)

Given a sequence of n non-empty intervals $[a_1, b_1], \dots, [a_n, b_n]$, we want to compute their union in time $O(n \log n)$. The output is represented by a sequence of disjoint intervals sorted by their left boundary. For example, if the intervals in the input were $[17, 19]$ $[-3, 7]$ $[4, 9]$ $[18, 21]$ $[-4, 15]$, then the intervals in the output would be $[-4, 15]$ $[17, 21]$.

(a) (1 point) Describe an algorithm that solves this problem. Explain the algorithm in words, without writing code, but clearly enough so that the algorithm can be implemented. Assume that the input is given by the vectors (a_1, \dots, a_n) and (b_1, \dots, b_n) , with $a_i \leq b_i$ for every $i = 1, \dots, n$.



(b) (1 point) Now, in addition to the sequence of n intervals, we are given a sequence m different reals p_1, \dots, p_m , and we want to determine how many fall in some interval of the union (only the number is needed; not which ones). Using the algorithm from the previous section, describe an algorithm that solves this problem in time $O(n \log n)$ when $m = n$. Assume that the input is given by the vectors a and b from the previous section, and the vector (p_1, \dots, p_m) with $p_i \neq p_j$ if $i \neq j$.

(c) (1 punt) If you happened to know that m is bounded by a small constant independent of n , say $m \leq 5$, would you still use the same algorithm? If not, which one would you use instead? If you choose a different algorithm, make its cost explicit.

Problem 4: Quickies**(3 points)**

- (0,5 points) Determine if they are equal ($=$) or different (\neq), and prove it:

$$\Theta(3^{\log_2(n)}) \quad \square \quad \Theta(3^{\log_4(n)}).$$

- (0,5 points) Compute $2^1 \cdot 2^2 \cdot 2^3 \cdot \dots \cdot 2^{99} \cdot 2^{100} \bmod 9$. This problem is not meant to be solved with the help of a calculator.

- (1 point) Sort the functions that follow increasingly according to their asymptotic growth rates: $n^4 - 3n^3 + 1, (\ln(n))^2, \sqrt{n}, n^{1/3}$. Exceptionally, you are not required to justify your answer.

- (1 point) Consider the following three alternatives for solving a problem:
 - A: divide an instance of size n into five instances of size $n/2$, solve each instance recursively, and combine the solutions in time $\Theta(n)$.

- B: given an instance of size n , solve two instances of size $n - 1$ recursively, and combine the solutions in constant time.
- C: divide an instance of size n into nine instances of size $n/3$, solve each instance recursively, and combine the solutions in time $\Theta(n^2)$.

Write down and solve the corresponding recurrences. Which alternative is the most efficient?



Midterm EDA Exam

Length: 2.5 hours

19/10/2015

Problem 1

(3.5 points)

Fibonacci numbers are defined by the recurrence $f_k = f_{k-1} + f_{k-2}$ for $k \geq 2$, with $f_0 = 0$ and $f_1 = 1$. Answer the following exercises:

- (a) (0.5 points) Consider the following function *fib1* which, given a non-negative integer k , returns f_k :

```
int fib1 (int k) {  
    vector<int> f(k+1);  
    f[0] = 0;  
    f[1] = 1;  
    for (int i = 2; i ≤ k; ++i)  
        f[i] = f[i-1] + f[i-2];  
    return f[k];  
}
```

Describe the asymptotic cost *in time and space* of *fib1* (k) as a function of k as precisely as possible.

- (b) (1 point) Show that, for $k \geq 2$, the following matrix identity holds:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k-1} = \begin{pmatrix} f_k & f_{k-1} \\ f_{k-1} & f_{k-2} \end{pmatrix}$$

- (c) (1 point) Fill the gaps in the following code so that function *fib2*(*k*) computes f_k , given a $k \geq 0$.

```

typedef vector<vector<int>> matrix;

matrix mult(const matrix& A, const matrix& B) {
    // Pre: A and B are square matrices of the same dimensions
    int n = A.size ();
    matrix C(n, vector<int>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                
    return C;
}

matrix mystery(const matrix& M, int q) {
    int s = M.size ();
    if (q == 0) {
        matrix R(s, vector<int>(s, 0));
        for (int i = 0; i < s; ++i) 
        return R;
    }
    else {
        matrix P = mystery(M, q/2);
        if (  ) return mult(P, P);
    }
}

```

```

    else return  ;
} }

int fib2 (int k) {
    if (k ≤ 1) return k;
    matrix M = { {1, 1}, {1, 0} };
    matrix P = mystery(M, k-1);
    return  ;
}

```

- (d) (1 point) Describe the asymptotic cost in time of $\text{fib2}(k)$ as a function of k as precisely as possible.

Problem 2

(3.25 points)

Given a vector of integers v and an integer x , the function

```

int position (const vector<int>& v, int x) {
    int n = v.size ();
    for (int i = 0; i < n; ++i)
        if (v[i] == x)
            return i;
    return -1;
}

```

examines the $n = v.size()$ positions of v and returns the first one that contains x , or -1 if there is none.

- (a) (0.75 points) Describe as a function of n the asymptotic cost in time of *position* in the best case as precisely as possible. When can this best case take place?

- (b) (0.75 points) Describe as a function of n the asymptotic cost in time of *position* in the worst case as precisely as possible. When can this worst case take place?

- (c) (0.75 points) Show that for any integer $n \geq 1$, the following identity holds:

$$\sum_{i=1}^n \frac{i}{2^i} = 2 - \frac{n}{2^n} - \frac{1}{2^{n-1}}.$$



- (d) (1 point) Assume that we have a probability distribution over the input parameters. Namely, the probability that x is element $v[i]$ is $\frac{1}{2^{i+1}}$ for $0 \leq i < n - 1$, and that it is element $v[n - 1]$ is $\frac{1}{2^{n-1}}$. In particular, these probabilities add up 1, so that x is always one of the n values of v .

Describe as a function of n the asymptotic cost in time of *position* in the average case as precisely as possible.

Problem 3**(3.25 points)**

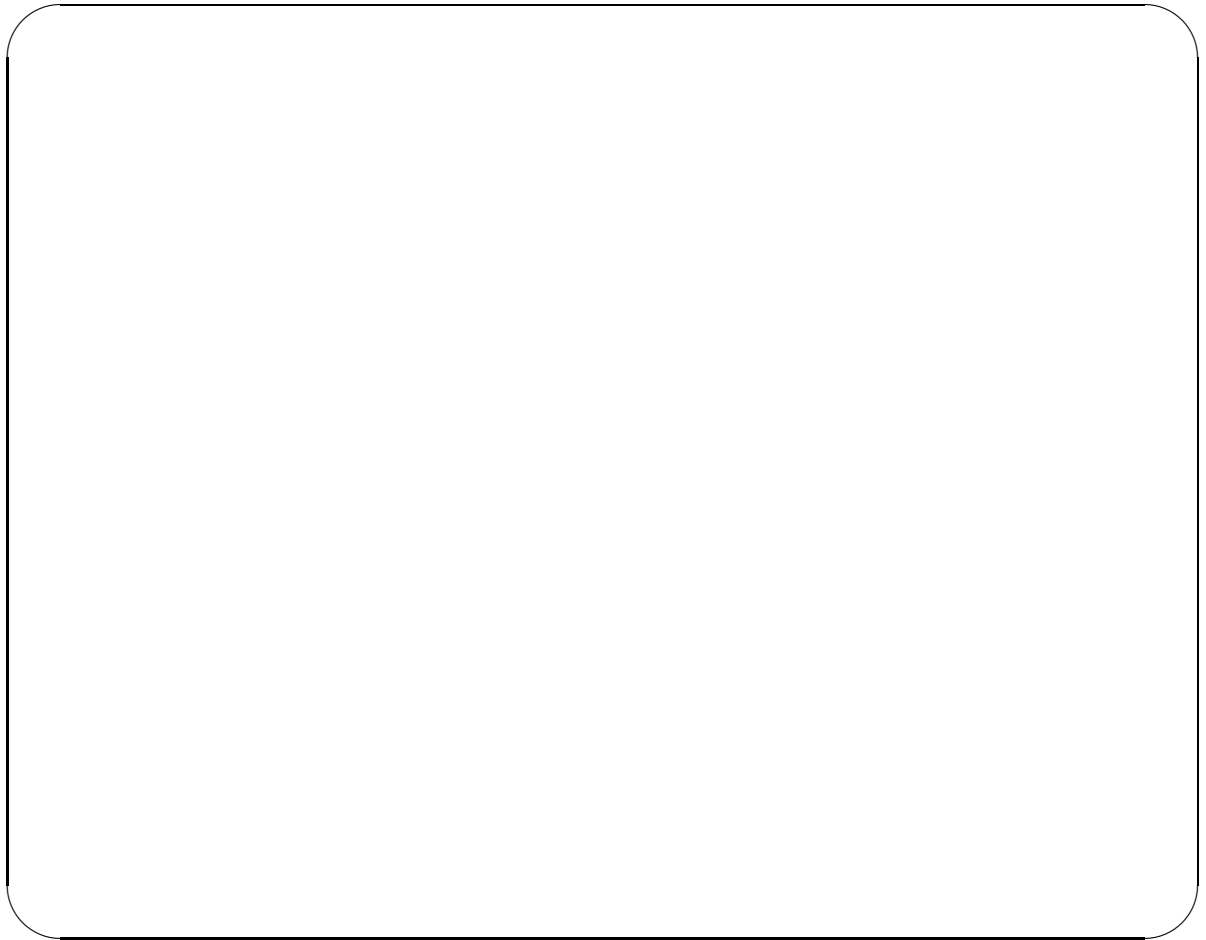
In this exercise we tackle the problem of, given two positive integers a and b , to compute their greatest common divisor $\gcd(a, b)$. Recall that $\gcd(a, b)$ is, by definition, the only positive integer g such that:

1. $g \mid a$ (g divides a),
2. $g \mid b$,
3. if $d \mid a$ and $d \mid b$, then $d \mid g$.

(a) (1.25 points) Show that the following identities are true:

$$\gcd(a, b) = \begin{cases} 2\gcd(a/2, b/2) & \text{if } a, b \text{ are even} \\ \gcd(a, b/2) & \text{if } a \text{ is odd and } b \text{ is even} \\ \gcd((a-b)/2, b) & \text{if } a, b \text{ are odd and } a > b \end{cases}$$

Hint: you can use that, given two positive integers a and b such that $a > b$, we have $\gcd(a, b) = \gcd(a - b, b)$.



- (b) (1 point) Write a function `int gcd(int a, int b)` in C++ which, by using divide and conquer and exercise (a), computes the greatest common divisor $\gcd(a, b)$ of two given positive integer numbers a, b .

Hint: you can also use that for any positive integer a , $\gcd(a, a) = a$.

- (c) (1 point) Assuming that a and b are positive integers, each of which represented with a vector of n bits, describe the cost in time in the worst case of $\gcd(a, b)$ as a function of n as precisely as possible. When can this worst case take place?

Assume that the cost of the following operations with integers of n bits: adding, subtracting, comparing, multiplying/dividing by 2 is $\Theta(n)$, and that computing the remainder modulo 2 takes time $\Theta(1)$.

Midterm EDA Length: 2.5 hours

31/03/2016

Problem 1

(1 point)

Answer the following questions. In this exercise, you do **not** need to justify the answers.

1. (0.2 points) The cost of Strassen's algorithm for multiplying two matrices of size $n \times n$ is $\Theta(\text{ } \boxed{\text{ }} \text{ })$

2. (0.6 points) The master theorem for subtractive recurrences states that given a recurrence $T(n) = aT(n - c) + \Theta(n^k)$ with $a, c > 0$ and $k \geq 0$ then:

$$T(n) = \begin{cases} \boxed{\text{ }} & \text{if } a < 1, \\ \boxed{\text{ }} & \text{if } a = 1, \\ \boxed{\text{ }} & \text{if } a > 1. \end{cases}$$

3. (0.2 points) The mergesort sorting algorithm uses $\Theta(\text{ } \boxed{\text{ }} \text{ })$ auxiliary space when sorting a vector of size n .

Problem 2

(3 points)

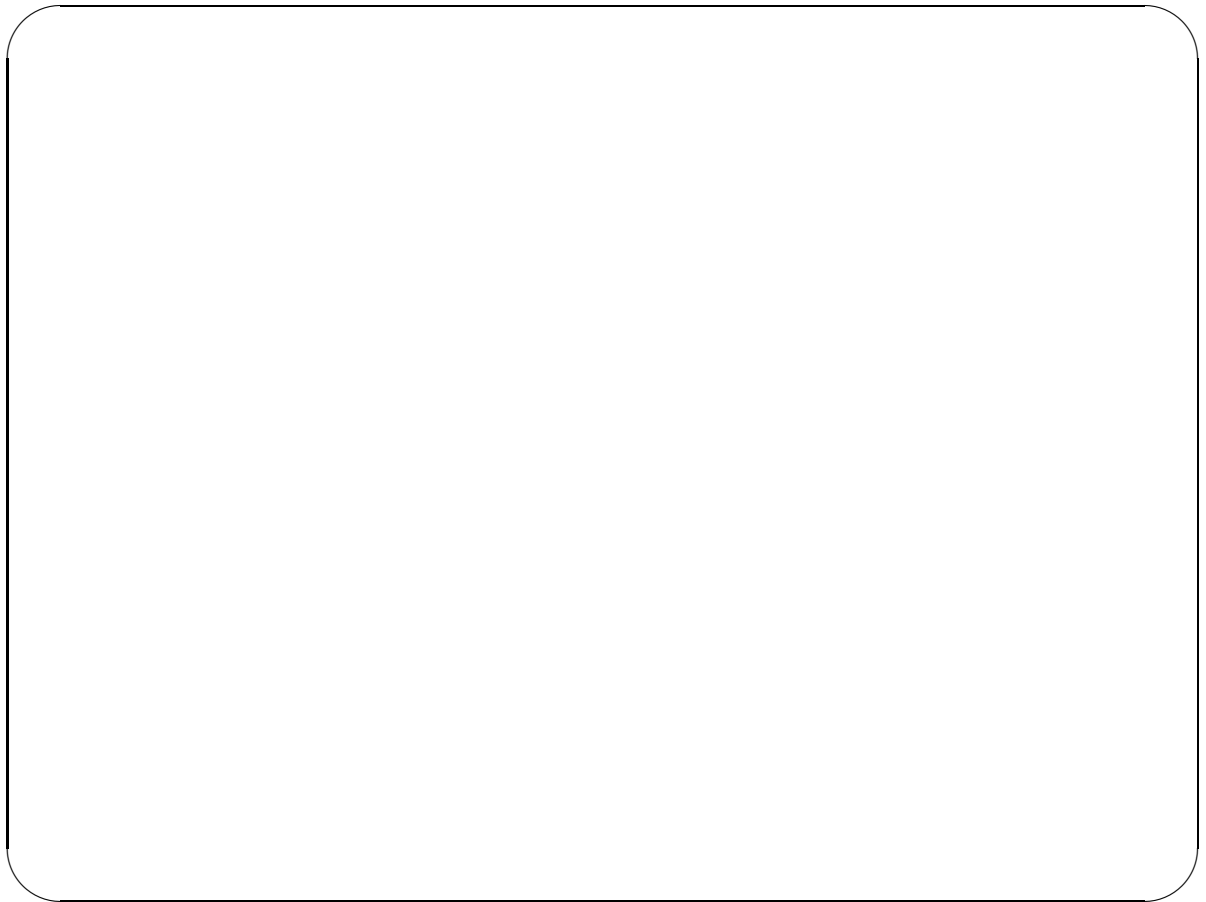
In this problem costs are only considered *in time*. Consider the following program:

```
void f(int m);
void g(int m);

void h(int n) {
    int p = 1;
    for (int i = 1; i ≤ n; i++) {
        f(i);
        if (i == p) {
            g(n);
            p *= 2;
        }
    }
}
```

- (a) (1.5 points) Answer: if the cost of $f(m)$ as well as the cost of $g(m)$ is $\Theta(m)$, then the cost of $h(n)$ as a function of n is $\Theta(\text{ } \boxed{\text{ }} \text{ })$

Justification:



(b) (1.5 points) Answer: if the cost of $f(m)$ is $\Theta(1)$ and the cost of $g(m)$ is $\Theta(m^2)$, then the cost of $h(n)$ as a function of n is $\Theta(\text{ })$

Justification:

Problem 3**(3 points)**

We say that a square matrix M of integer numbers of size $n \times n$ is *symmetric* if $M_{i,j} = M_{j,i}$ for all i, j such that $0 \leq i, j < n$.

Let us consider the following implementation of symmetric matrices with unidimensional vectors, which only stores the “lower triangle” to minimise the consumed space:

$$\begin{pmatrix} M_{0,0} & M_{0,1} & M_{0,2} & \dots & M_{0,n-1} \\ M_{1,0} & M_{1,1} & M_{1,2} & \dots & M_{1,n-1} \\ M_{2,0} & M_{2,1} & M_{2,2} & \dots & M_{2,n-1} \\ \vdots & & & \ddots & \\ M_{n-1,0} & M_{n-1,1} & M_{n-1,2} & \dots & M_{n-1,n-1} \end{pmatrix}$$



$M_{0,0}$	$M_{1,0}$	$M_{1,1}$	$M_{2,0}$	$M_{2,1}$	$M_{2,2}$	\dots	$M_{n-1,0}$	$M_{n-1,1}$	$M_{n-1,2}$	\dots	$M_{n-1,n-1}$
-----------	-----------	-----------	-----------	-----------	-----------	---------	-------------	-------------	-------------	---------	---------------

For example, the symmetric matrix 3×3

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & -2 & 3 \\ 0 & 3 & -1 \end{pmatrix}$$

would be implemented with the unidimensional vector

1	2	-2	0	3	-1
---	---	----	---	---	----

- (a) (0.5 points) Answer: the cost in space of this representation for a symmetric matrix $n \times n$ as a function of n is $\Theta(\text{ })$.

Justification:

- (b) (1 point) Given $n > 0$ and a value k , the function

```
pair<int,int> row_column(int n, int k);
```

j is the column of the coefficient pointed by k in a unidimensional vector that implements a symmetric matrix $n \times n$. If k is not a valid index, it returns $(-1, -1)$. For example, `row_column(3,2)` returns $(1,1)$, `row_column(3,3)` returns $(2,0)$, and `row_column(3,6)` returns $(-1, -1)$.

Complete the following implementation of `row_column`:

```
int p(int x) { return  ;}

int mystery(int k, int l, int r) {
    if (l+1 == r) return l;
    int m = (l+r)/2;
    if (p(m) ≤ k) return mystery(k, , r);
    else return mystery(k, l,  );
}

pair<int,int> row_column(int n, int k) {
    if (k < 0 or k ≥ p(n)) return  ;
    int i = mystery(k, 0, n);
    return {i,  };
}
```

- (c) (1 point) Analyse the cost in time in the worst case of the implementation of `row_column(int n, int k)` of the previous exercise as a function of n .

- (d) (0.5 points) Using the functions **double** *sqrt*(**double** *x*) and **int** *floor* (**double** *x*) which respectively compute \sqrt{x} and $\lfloor x \rfloor$ in time $\Theta(1)$, give an alternative implementation of *row_column* with cost $\Theta(1)$.

Problem 4

(3 points)

Assume that n is a power of 2, that is, of the form 2^k for a certain $k \geq 0$.

A square matrix A of real numbers of size $n \times n$ is a *Monge matrix* if for all i, j, k and l such that $0 \leq i < k < n$ and $0 \leq j < l < n$, it holds that

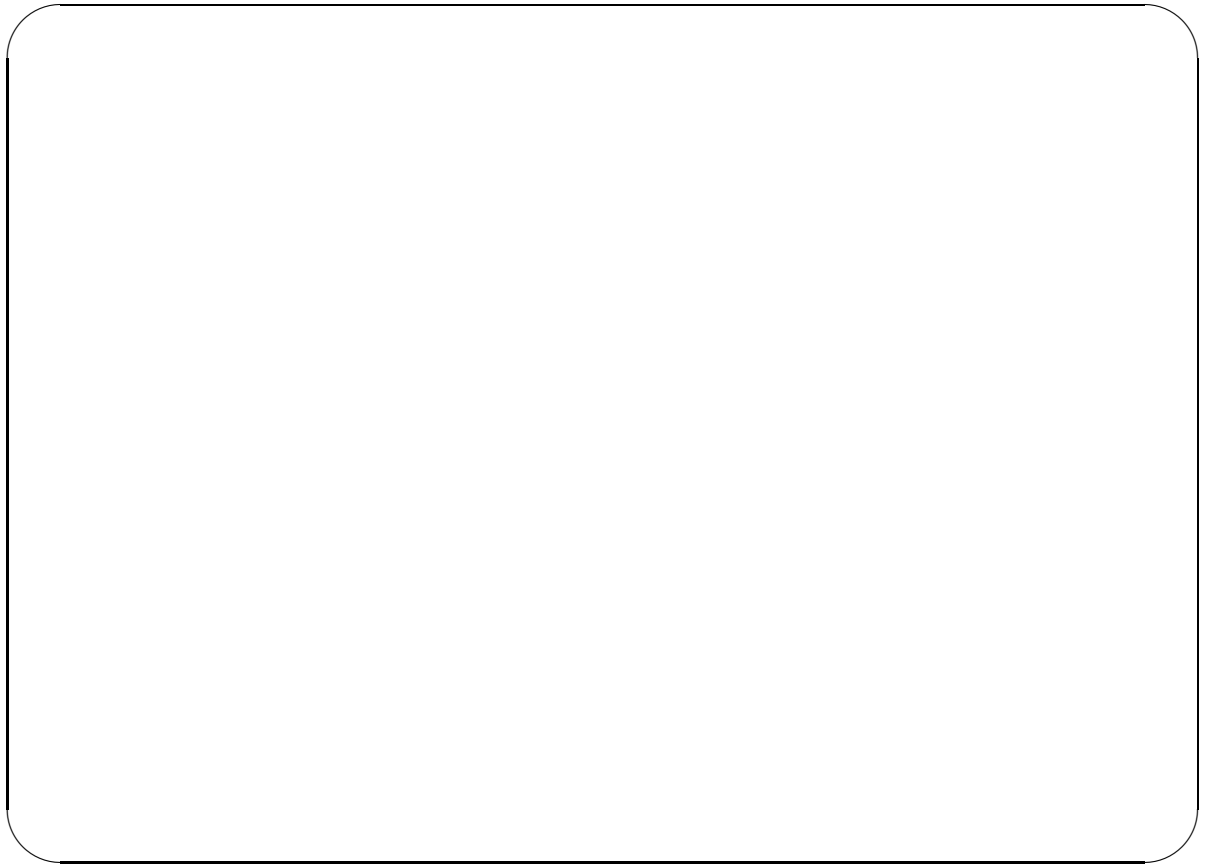
$$A_{i,j} + A_{k,l} \leq A_{i,l} + A_{k,j}$$

In other words, whenever we take two rows and two columns of a Monge matrix and consider the four elements at the intersections of the rows and the columns, the sum of the elements at the upper left and lower right corners is less than or equal to the sum of the elements at the upper right and lower left corners. For example, the following matrix is a Monge matrix:

$$\begin{pmatrix} 10 & 17 & 13 & 28 \\ 16 & 22 & 16 & 29 \\ 24 & 28 & 22 & 34 \\ 11 & 13 & 6 & 6 \end{pmatrix}$$

- (a) (1 point) Let $f_A(i)$ be the index of the column where the the minimum element of the i -th row appears (breaking ties if needed by taking the leftmost one). For example, in the matrix above $f_A(0) = 0$, $f_A(1) = 0$, $f_A(2) = 2$ i $f_A(3) = 2$.

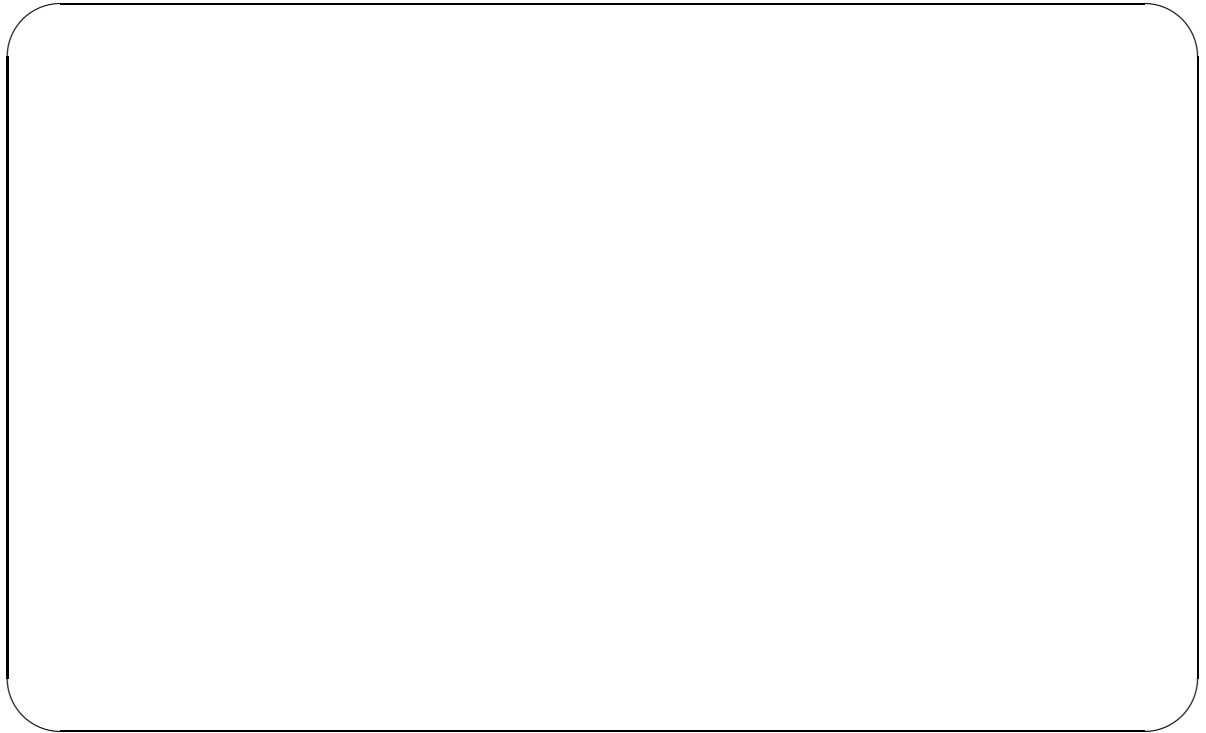
Show that $f_A(0) \leq f_A(1) \leq \dots \leq f_A(n-1)$ for any Monge matrix A of size $n \times n$.



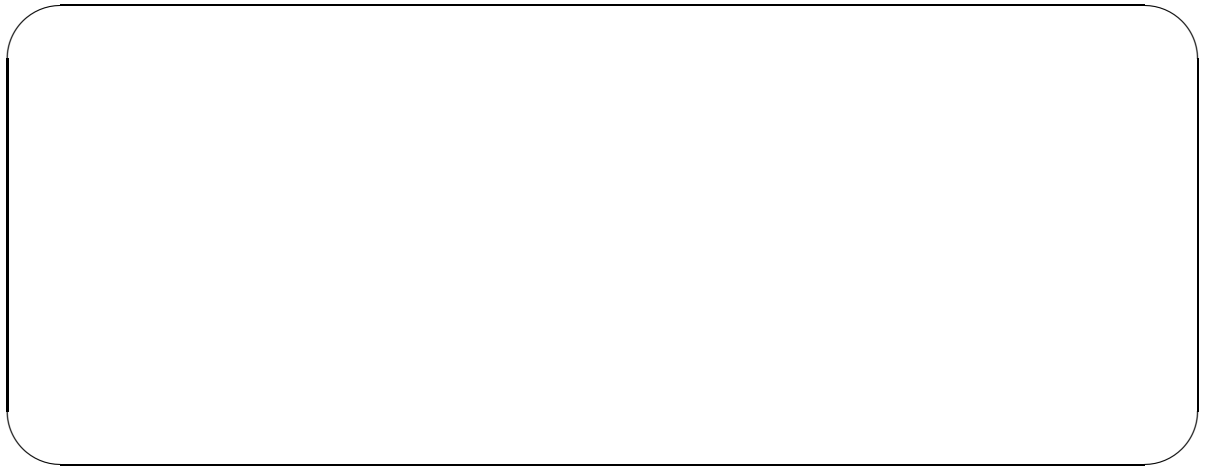
(b) (1 point) In what follows a divide-and-conquer algorithm is described which computes the function f_A for all rows of a Monge matrix A :

- (1) Build two square submatrices B_1 i B_2 of the matrix A of sizes $\frac{n}{2} \times \frac{n}{2}$ as follows: B_1 is formed by the rows of A with even index and the columns between 0 and $\frac{n}{2} - 1$, and B_2 is formed by the rows of A with even index and the columns between $\frac{n}{2}$ and $n - 1$.
- (2) Recursively determine the column where the leftmost minimum appears for any row of B_1 and for any row of B_2 .
- (3) Find the column where the leftmost minimum appears for any row of A .

Explain how, by using the result of (2), one can accomplish (3) in time $\Theta(n)$.



- (c) (1 point) Compute the cost as a function of n of the algorithm proposed in the previous exercise for computing the function f_A for all rows of a Monge matrix A of size $n \times n$. Assume that step (1) can be carried out in time $\Theta(n)$.



Midterm EDA exam

Length: 2.5 hours

07/11/2016

Problem 1

(2 points)

In this problem you do not need to justify your answers.

- (a) (0.8 pts.) Fill the gaps in the following table (except for the cell marked with **Do not fill**) with the costs in time for sorting a vector of integers of size n using the indicated algorithms. Assume uniform probability in the average case.

	<i>Best case</i>	<i>Average case</i>	<i>Worst case</i>
Quicksort (with Hoare's partition)			
Mergesort			
Insertion		Do not fill	

- (b) (0.2 pts.) The solution to the recurrence $T(n) = 2T(n/4) + \Theta(1)$ is

- (c) (0.2 pts.) The solution to the recurrence $T(n) = 2T(n/4) + \Theta(\sqrt{n})$ is

- (d) (0.2 pts.) The solution to the recurrence $T(n) = 2T(n/4) + \Theta(n)$ is

- (e) (0.3 pts.) What does Karatsuba's algorithm compute? What is its cost?

- (f) (0.3 pts.) What does Strassen's algorithm compute? What is its cost?

Problem 2**(3 points)**

Given $n \geq 2$, we say that a sequence of n integers a_0, \dots, a_{n-1} is *bi-increasing* if $a_{n-1} < a_0$ and there exists an index t (with $0 \leq t < n$) that satisfies the following conditions:

- $a_0 \leq \dots \leq a_{t-1} \leq a_t$
- $a_{t+1} \leq a_{t+2} \leq \dots \leq a_{n-1}$

For example, the sequence 12, 12, 15, 20, 1, 3, 3, 5, 9 is bi-increasing (take $t = 3$).

(a) (2 pts.) Implement in C++ a function

```
bool search(const vector<int>& a, int x);
```

which, given a vector a that contains a bi-increasing sequence and an integer x , returns whether x appears in the sequence or not. If you use auxiliary functions that are not part of the C++ standard library, implement them too. The solution must have cost $\Theta(\log(n))$ in time in the worst case.

- (b) (1 pt.) Justify that the cost in time in the worst case of your function *search* is $\Theta(\log(n))$. When does this worst case take place?

Problem 3**(2 points)**

Consider the following function:

```
int mystery(int m, int n) {  
    int p = 1;  
    int x = m;  
    int y = n;  
    while (y  $\neq$  0) {  
        if (y % 2 == 0) {  
            x *= x;  
            y /= 2;  
        }  
        else {  
            y -= 1;  
            p *= x;  
        }  
    }  
    return p;  
}
```

- (a) (1 pt.) Given two integers $m, n \geq 0$, what does $mystery(m, n)$ compute? You do not need to justify your answer.

- (b) (1 pt.) Analyse the cost in time in the worst case as a function of n of $mystery(m, n)$.

Problem 4**(3 points)**

The *Fibonacci sequence* is defined by the recurrence $F(0) = 0$, $F(1) = 1$ and

$$F(n) = F(n-1) + F(n-2) \quad \text{if } n \geq 2.$$

- (a) (0.5 pts.) Let $\phi = \frac{\sqrt{5}+1}{2}$ be the so-called *golden number*. Prove that $\phi^{-1} = \phi - 1$.

(b) (1.5 pts.) Prove that, for any $n \geq 0$, we have

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}$$

(c) (1 pt.) Prove that $F(n) = \Theta(\phi^n)$.



2

Lab Exams

Lab Exam EDA Length: 2 hours

13/12/2010

Shift 1

Problem 1

Judge.org, Problem P39846: Treasures in a map (4)
(https://www.judge.org/problems/P39846_en).

Problem 2

Judge.org, Problem P71701: Peaceful kings
(https://www.judge.org/problems/P71701_en).

Shift 2

Problem 1

Judge.org, Problem P84415: Bag of words
(https://www.judge.org/problems/P84415_en).

Problem 2

Judge.org, Problem P22295: The travelling tortoise
(https://www.judge.org/problems/P22295_en).

Lab Exam EDA Length: 2 hours

19/5/2011

Problem 1

Judge.org, Problem P69865: Picking up coins
(https://www.judge.org/problems/P69865_en).

Problem 2

Judge.org, Problem P98123: Filling the bag
(https://www.judge.org/problems/P98123_en).

Lab Exam EDA Length: 2 hours

13/12/2011

Problem 1

Judge.org, Problem P90766: Treasures in a map (3)
(https://www.judge.org/problems/P90766_en).

Problem 2

Jutge.org, Problem P67329: DNA
(https://www.jutge.org/problems/P67329_en).

Lab Exam EDA**Length: 2 hours****14/05/2012****Problem 1**

Jutge.org, Problem P47386: Gossip
(https://www.jutge.org/problems/P47386_en).

Problem 2

Jutge.org, Problem P87462: Pacman
(https://www.jutge.org/problems/P87462_en).

Lab Exam EDA**Length: 2hours****29/11/2012****Problem 1**

Jutge.org, Problem P90861: Queues of a supermarket (1)
(https://www.jutge.org/problems/P90861_en).

Problem 2

Jutge.org, Problem P14952: Topological sort
(https://www.jutge.org/problems/P14952_en).

Lab Exam EDA**Length: 2 hours****22/5/2013****Problem 1**

Jutge.org, Problem X07174: Nombres sense prefixos prohibits
(https://www.jutge.org/problems/X07174_ca).

Problem 2

Jutge.org, Problema X34032: El cavall afamat
(https://www.jutge.org/problems/X34032_en).

Lab Exam EDA Length: 2 hours

4/12/2013

Problem 1

Jutge.org, Problem P60796: Treasures in a map (2)
(https://www.jutge.org/problems/P60796_en).

Problem 2

Jutge.org, Problem X21319: Circuit Value Problem
(https://www.jutge.org/problems/X21319_en).

Lab Exam EDA Length: 2 hours

19/5/2014

Problem 1

Jutge.org, Problem P81453: Shortest path
(https://www.jutge.org/problems/P81453_en).

Problem 2

Jutge.org, Problem P89318: Forbidden words
(https://www.jutge.org/problems/P89318_en).

Lab Exam EDA Length: 2 hours

22/12/2014

Problem 1

Jutge.org, Problem X41530: Forest
(https://www.jutge.org/problems/X41530_en).

Problem 2

Jutge.org, Problem X92609: Two coins of each kind (3)
(https://www.jutge.org/problems/X92609_en).

Lab Exam EDA Length: 2.5 hours

25/05/2015

Problem 1

Jutge.org, Problem P86108: LOL
(https://www.jutge.org/problems/P86108_ca).

Problem 2

Jutge.org, Problem P27258: Monstres en un mapa
(https://www.jutge.org/problems/P27258_ca).

Lab Exam EDA**Length: 2.5 hours****22/12/2015****Problem 1**

Jutge.org, Problem P43164: Treasures in a map (5)
(https://www.jutge.org/problems/P43164_en).

Problem 2

Jutge.org, Problem P31389: Rooks inside a rectangle
(https://www.jutge.org/problems/P31389_en).

Lab Exam EDA**Length: 2.5 hours****19/05/2016****Problem 1**

Jutge.org, Problem P12887: Minimum spanning trees
(https://www.jutge.org/problems/P12887_en).

Problem 2

Jutge.org, Problem P54070: Rightmost position of insertion
(https://www.jutge.org/problems/P54070_en).

Lab Exam EDA**Length: 2.5 hours****15/12/2016****Shift 1****Problem 1**

Jutge.org, Problem P76807: Sudoku
(https://www.jutge.org/problems/P76807_en).

Problem 2

Jutge.org, Problema P99753: Bi-increasing vector
(https://www.jutge.org/problems/P99753_en).

Shift 2**Problem 1**

Judge.org, Problema P18679: Balance beam (1)
(https://www.judge.org/problems/P18679_en).

Problem 2

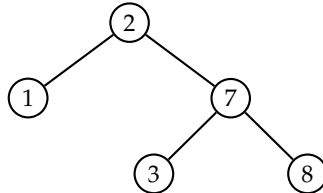
Judge.org, Problema P74219: Fibonacci numbers (2)
(https://www.judge.org/problems/P74219_en).

3

Final Exams

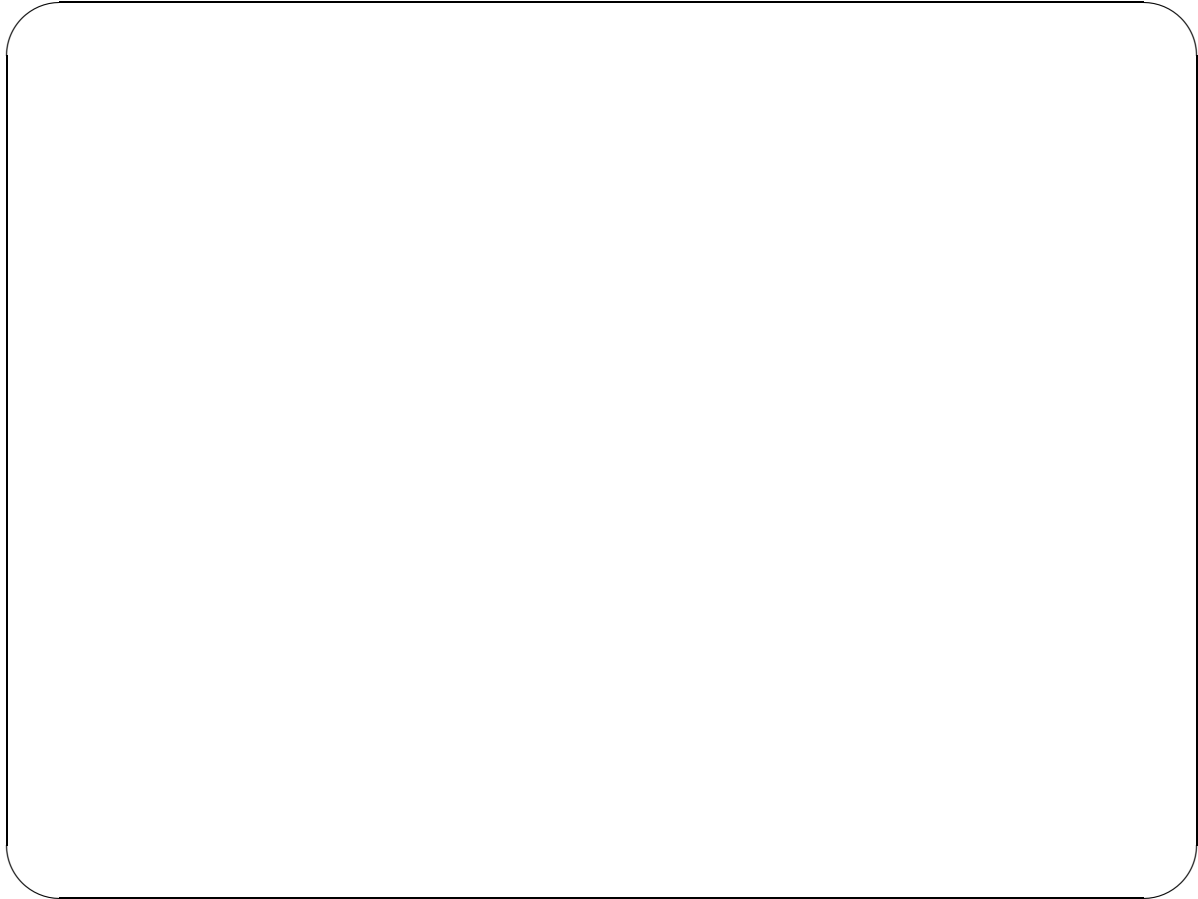

```
node(Elem x, node* lc , node* rc ) : x( x ), lc ( lc ), rc ( rc ) {}  
~node() { delete lc ; delete rc ; }  
};  
typedef node* bst ; // A BST is a pointer to the root (null if empty)
```

(a) (1.0 points) Implement a C++ function `bst redo(const vector<Elem> &v)` that reconstructs a binary search tree from its preorder traversal v . For example, on input $v = [2\ 1\ 7\ 3\ 8]$, the function should produce the following tree:




You may assume that v is the correct preorder of some BST. If you need so, you may implement an auxiliary function.

(b) (1.0 points) Implement a C++ function *bst build_AVL*(**const vector**<*Elem*> &*v*) that re-constructs a BST that satisfies the AVL property from a sorted vector of elements *v*. If you need so, you may implement an auxiliary function.



(c) (1.0 points) If we add a field *n.height* to denote the height of node *n* in the BST where it sits (with leaves at height 0 and therefore *NULL*s would be at height -1), complete the following code so that the result indicates if the BST *T* is an AVL:

```
bool is_AVL(bst T) {  
    if (T == NULL) return true;  
    else if (T→lc == NULL or T→rc == NULL) return (T→height ≤ 1);  
    else {
```



```
    }  
}
```

Problem 3: Hamiltonian paths in tournament graphs**(3 points)**

Let us represent directed graphs with adjacency matrices:

```
typedef vector<vector<bool>> Graph;
```

A *tournament graph* is a directed graph in which there is exactly one arc between every pair of different vertices, and that does not have any arc from any vertex to itself.

a) (0.5 points) Write a C++ function that tells whether a given graph with n vertices is a tournament graph in $\Theta(n^2)$ time in the worst case.

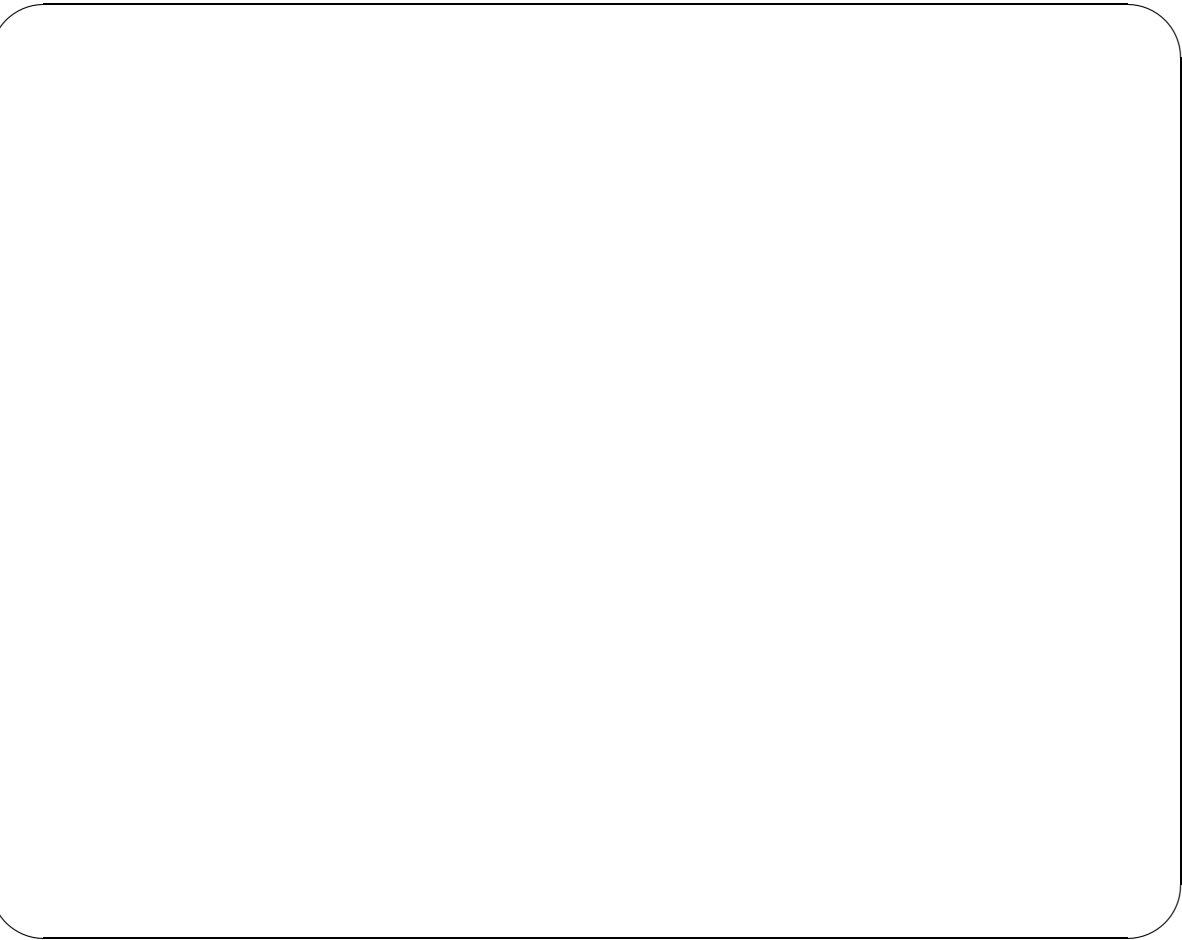
```
bool is_tournament(const Graph& G) {
```

```
}
```

b) (1.0 points) Prove, by induction on the number of vertices, that every tournament graph has a Hamiltonian path, that is, a path that visits every vertex exactly once (hint: show that a new vertex can be inserted in a path of $n - 1$ vertices to get a path of n vertices).

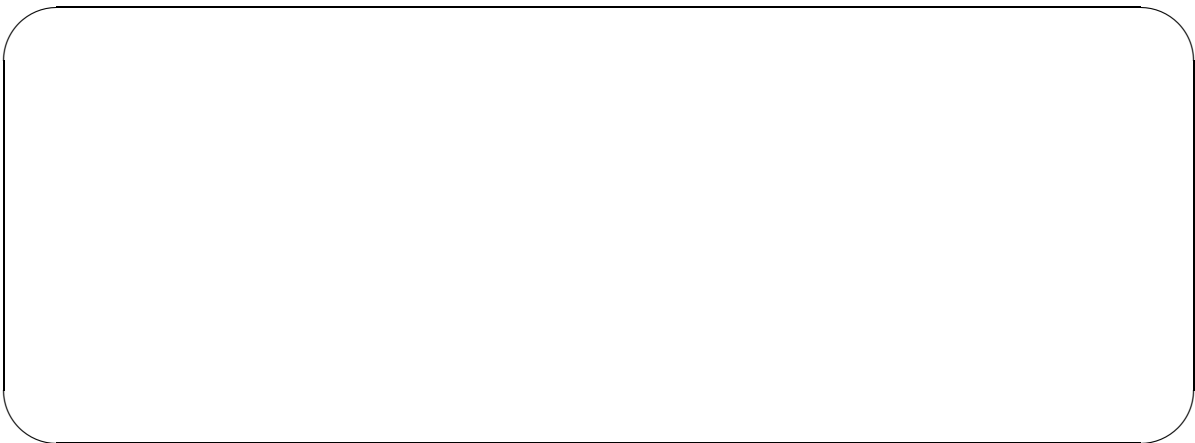
c) (1.0 points) Using the previous proof, give an algorithm that returns a Hamiltonian path of a tournament graph (clarity will be more valued than efficiency):

```
// Pre: G is a tournament graph
// Post: the result is a list of vertices that forms a Hamiltonian path in G
list <int> path(const Graph &G) {
```



```
}
```

d) (0.5 points) Analyze the cost of your solution.

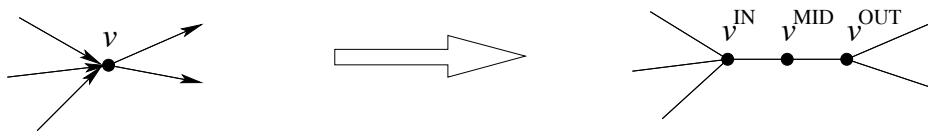


Problem 4: Reductions**(1 point)**

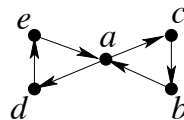
The problem of Hamiltonian cycles in directed graphs is:

DHAM: Given a directed graph, determine whether there is a cycle that visits all vertices exactly once.

The problem of Hamiltonian cycles in undirected graphs UHAM is the same, but where the input graph is undirected. Steffy knows that DHAM can be reduced to UHAM by simply applying the following transformation to each of the vertices v of the directed graph:



Roy and Johnny say that vertex v^{MID} is not actually needed: they say connecting $v^{\text{IN}} - v^{\text{OUT}}$ and ignoring v^{MID} is enough. But Steffy replies: "Guys, take a look at this input to DHAM and then we talk again":



What did Steffy mean? Is she right? (be concise but precise, please)

Problem 5: Bellman-Ford (*competència transversal*)**(1 point)**

Recall (from the practical work of the *competència transversal*) that Bellman-Ford algorithm allows one to find minimum-weight paths in a weighted directed graph, even when weights are negative.

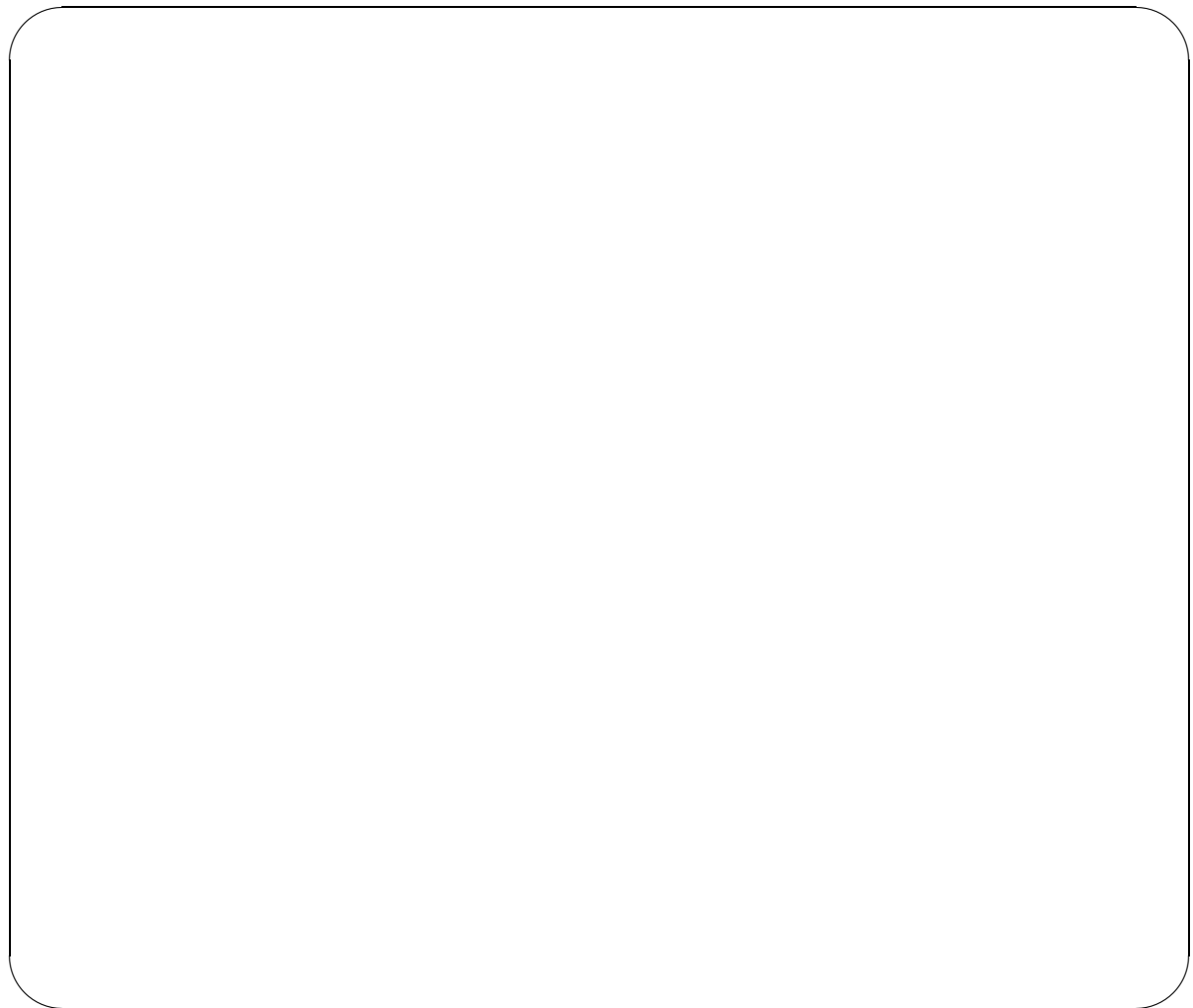
(a) (0.5 points) Which is the worst-case cost of Bellman-Ford algorithm on a graph with $|V|$ vertices and $|E|$ edges? Answer: $\Theta(\text{ })$

(b) (0.5 points) Steffy says that the statement at the beginning of this exercise cannot be completely right, since otherwise one could efficiently find a Hamiltonian cycle in a directed graph $G = (V, E)$ by simply assigning weight -1 to each edge in G , choosing any vertex s , and applying Bellman-Ford to check if the minimum-weight path from s to one of the vertices that have an edge towards s has weight $-(|V| - 1)$. How can you explain this paradox? (or have we proved that $P = NP$?)

Final exam EDA Length: 3 hours**16/1/2015****Problem 1: Asymptotic analysis (the same as in the midterm!)****(2 points)**

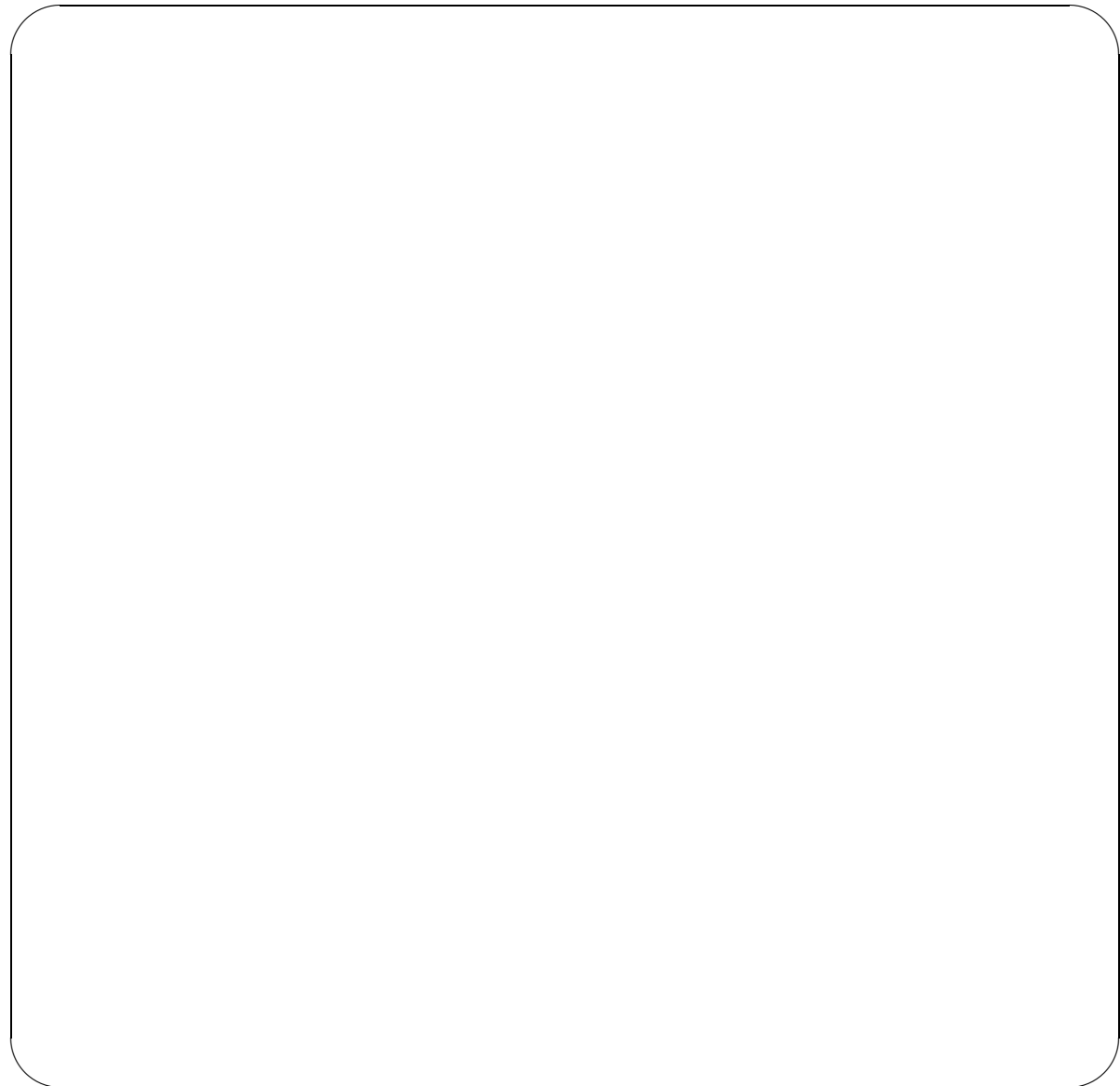
(1 point) Prove that $\sum_{i=0}^n i^3 = \Theta(n^4)$.

Hint: There are several ways to do it, and one of them is to consider the O and the Ω separately.



Note: Second part in the back.

(1 point) Let $f(n) = 2\sqrt{\log n}$ and $g(n) = n$. Which one is asymptotically faster? Prove it.

**Problem 2: Binary search trees and AVLs****(3 points)**

With the usual representation of binary search trees (this also holds for AVLs), some algorithms may be inefficient. For instance, function *smaller*(*T*, *x*) that given a tree *T* and a key *x*, returns the number of elements in *T* that have a key $\leq x$.

The following modification in the representation

```
struct node {  
    node* lft ;  
    node* rgt ;  
    Key k;      // generic type  
    Value v;    // generic type  
    int size ;  // size of the subtree rooted at this node ***** NEW *****  
};
```

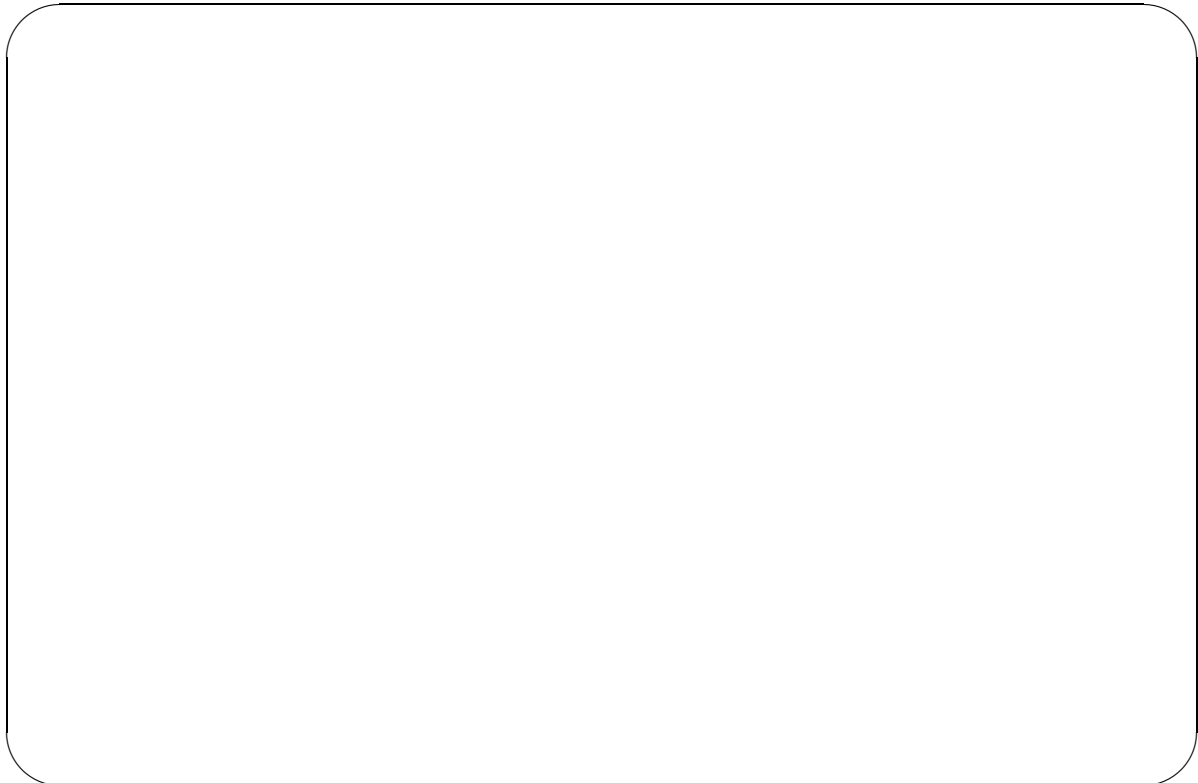


```
typedef node* bst;
```

allows, between others, an implementation of *smaller*(*T*, *x*) with cost $\Theta(h)$, where *h* is the height of *T* (which would be $\Theta(\log n)$ on average for a random BST and $\Theta(\log n)$ in the worst case for an AVL, where *n* is the size of *T*).

(1,5 points) Write in C++ an efficient implementation of the function

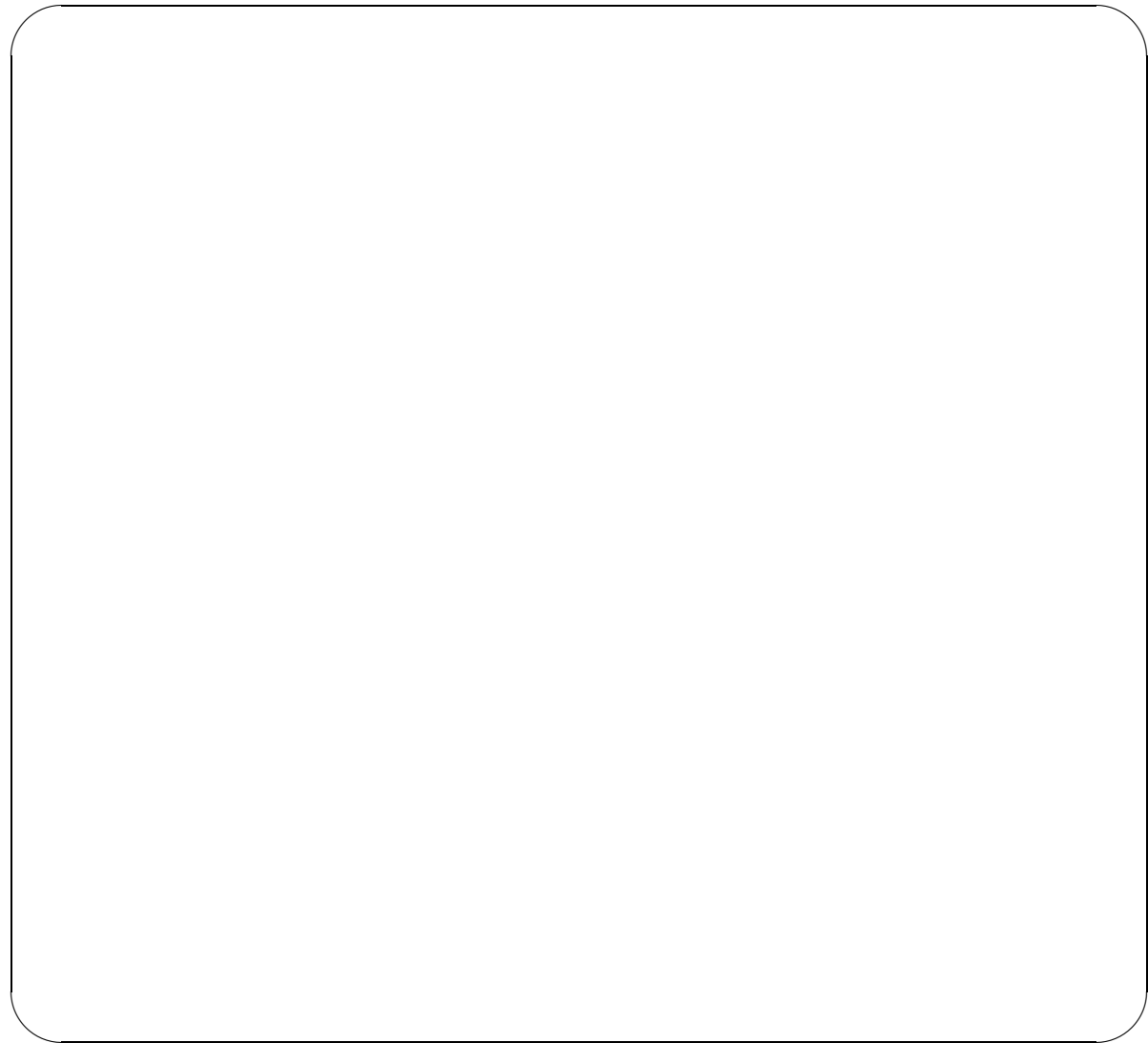
```
int smaller(bst T, const Key& x)
```



Note: Next question in the back.

(1,5 points) Write in C++ the modification of the *insert*(*T*,*k*,*v*) procedure that inserts a new pair $\langle k, v \rangle$ in a tree *T*. This procedure has as a precondition that *k* is not in *T*. Your implementation of *insert*() must have the same cost it would have under the usual representation of BSTs but, of course, somehow it must update the *size* attribute. Note that you are not asked to maintain the AVL property, nor that *T* is an AVL.

```
// Precondition: T does not contain k  
void insert(bst& T, const Key& k, const Value& v)
```

**Problem 3: Graphs and NP-completeness****(3 points)**

A mate of your is preparing the EDA lab exam and gets stuck in a problem in the “Graph Algorithms” list of the Judge. After a bit of thought, she realizes that this problem essentially asks to determine whether an undirected graph represented with adjacency lists is 2-colorable. That is, is it possible to assign a *red* or *blue* color to each vertex so that no two adjacent vertices are painted with the same color?

Unfortunately, the Judge responds Time Limit Exceeded to all her submissions! At this moment, your mate decides to look for some more efficient code in the Internet and finds the following implementation in a web forum:

```
bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
                vector<bool>& marked, bool is_red) {
    col[u] = is_red ;
    marked[u] = true;
    for (int i = 0; i < g[u].size (); ++i) {
```

```

    int v = g[u][i];
    if (not marked[v]) {
        if (not two_col_aux(g, v, col, marked, not is_red)) return false;
    }
}
return true;
}

bool two_colourable (const vector<vector<int>>& g) {
    int n = g.size ();
    vector<bool> marked(n, false);
    vector<bool> col(n);
    for (int u = 0; u < n; ++u) {
        if (not marked[u]) {
            if (not two_col_aux(g, u, col, marked, false)) return false;
        }
    }
    return true;
}

```

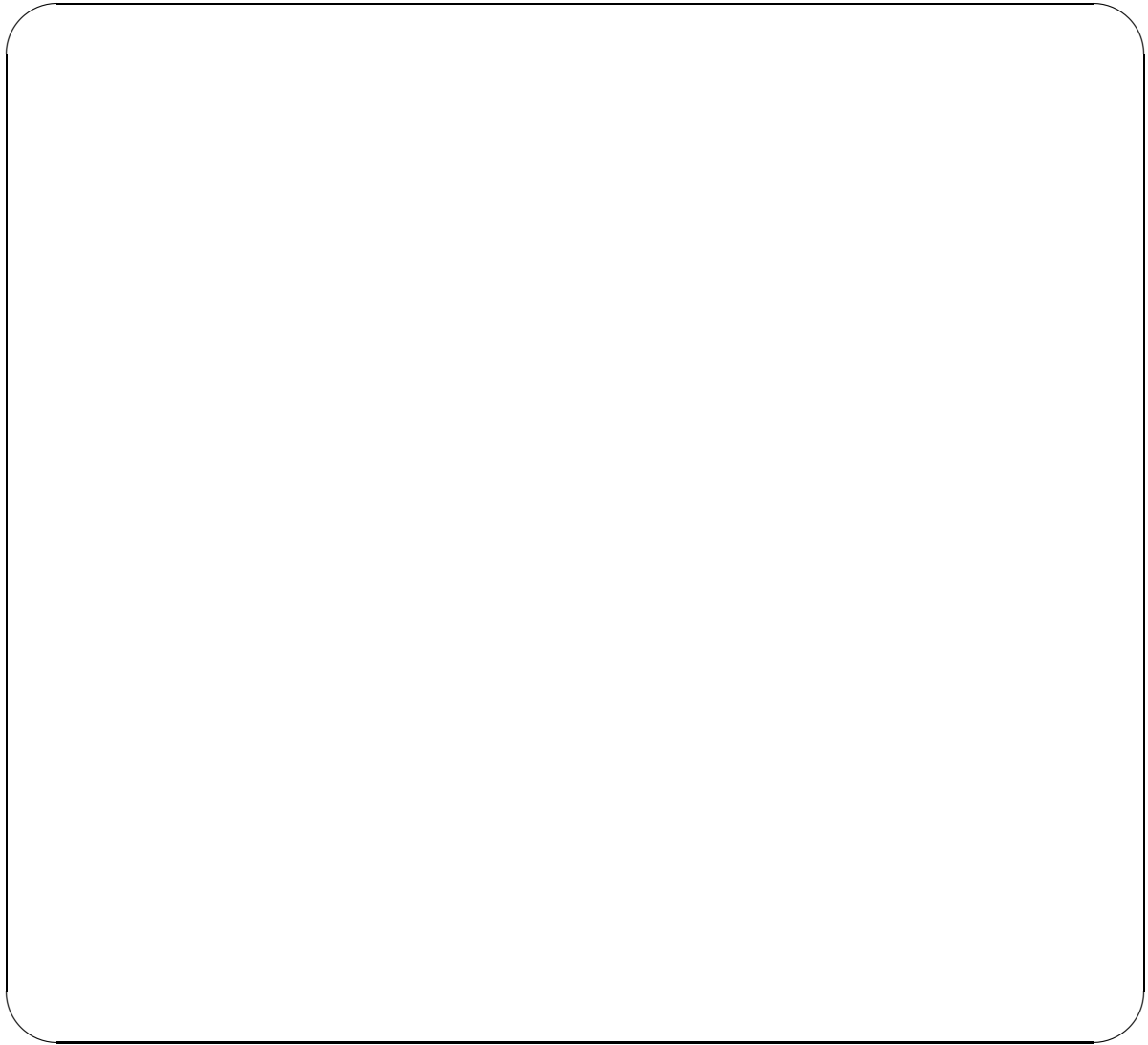
Now, she expects that the code will be efficient enough, since after a bit of analysis, she knows it takes $O(|V| + |E|)$ time. After sending it to the Judge, the verdict is... Wrong Answer!

(2 points) Write a correct version of *two_col_aux* so that *two_colorable* () properly determines if a graph is 2-colorable within $O(|V| + |E|)$ time.

```

bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
                vector<bool>& marked, bool is_red)

```



(1 point) Thanks to your help (hopefully!), your mate could fix the code and get it accepted. Irritated, she continues reading the forum. The author of the code affirms there that his program can be easily extended to solve the 3-colorability problem with the same asymptotic complexity. Do you believe him? Why or why not?



Problem 4: Algorithmic culture

(2 point)

Given a directed graph whose arcs have costs that can be negative, we wish to determine if it contains one or more negative cost cycles (just if they exist, not how many there are, nor

which are they).

(0,5 points) Give the proper name of a famous algorithm that helps solving this problem with just a little adaptation.

(0,5 points) Briefly explain (but with enough details to implement it) how this algorithm works.

(0,5 points) What is the cost of this algorithm? Express it using the number of vertices $|V|$ and the number of arcs $|E|$ in the input graph.

(0,5 point) Explain how to adapt the general algorithm to our particular problem (negative

cycle detection).



Final exam EDA Length: 3 hours**12/6/2015****One from the midterm, one from that thing, and one more****(2 points)**

(1 point) Determine if they are equal ($=$) or different (\neq), and prove it:

$$\Theta(3^{\log_2(n)}) \quad \square \quad \Theta(3^{\log_4(n)}).$$



(0.5 points) What is the algorithm from the *competencia transversal* useful for? Choose only one (reasoning not requested):

- To compute the most likely phylogenetic tree.
- To compute the optimal price of a long option over a financial product.
- To fit all 9 Beethoven symphonies into a single CD.
- To solve recurrences that do not match the form of the master theorems.

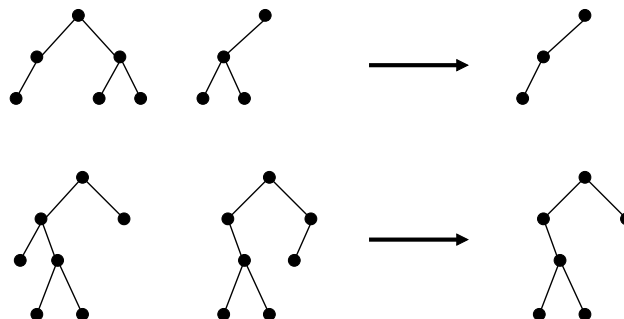
(0.5 points) The SUBSETSUM problem is the following: Given a list of natural numbers a_1, \dots, a_m and a target b , all written in decimal notation, determine if there exists a subset $S \subseteq \{1, \dots, m\}$ such that $\sum_{i \in S} a_i = b$. If d is the number of digits of the largest natural number in the input, indicate which is the only correct assertion (reasoning not requested):

- The problem is NP-complete and, therefore, there does not exist any algorithm of cost $O(m \cdot d \cdot 10^d)$ unless $P = NP$.
- The problem is NP-complete and, therefore, there does not exist any algorithm of cost polynomial in m and d unless $P = NP$.

- The problem is NP-complete but nonetheless a known algorithm exists whose cost is polynomial in both m and d .
- The problem can be solved in time $O(d \cdot m \cdot \log m)$, and hence cost polynomial in the size of the input, as follows: first, all m numbers are added to a max-heap; then, the numbers are extracted one by one until either the sum exceeds b , or all numbers are extracted without reaching b , or the sum of the extracted numbers is exactly b .

Pointers**(2 points)**

Given two binary trees, create a new binary tree that is their intersection (and leave the given trees untouched). For example:



(1.5 points) Using the type and header as given, solve this problem.


```
struct Node {           // nodes do not store data; just pointers to children
    Node* left;          // pointer to left child
    Node* right;         // pointer to right child
};
```

```
typedef Node* Tree;
```

```
Tree intersection (Tree t1, Tree t2) {
```




(0.5 points) What is its cost, in the worst case, as a function of the sizes n_1 and n_2 of t_1 and t_2 ?

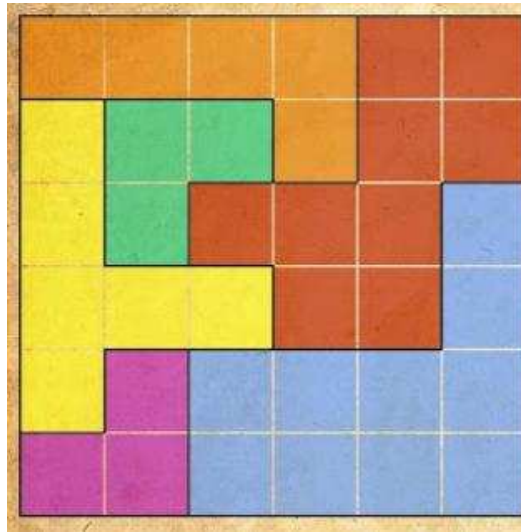


Backtracking fill-in-the-gaps

(2 points)

Onions is a logical puzzle¹ in which we are given a square matrix $n \times n$ divided into n regions, each region painted with a different color.

¹ The original puzzle is called *Alberi* (tree in Italian) and the apps (both iOS and Android) for this game were tremendously successful a few years ago.



The goal of the puzzle is to plant n onions in such a way that

- each row contains exactly one onion,
- each column contains exactly one onion,
- each region contains exactly one onion,
- there are no two horizontally, vertically, or diagonally adjacent onions; in other words, all cells surrounding an onion must be free.

Using the following definition

```
struct Cell {
    int color; // the color (between 0 and n-1) of its region
    bool has_onion; // indicates if there is an onion in it or not
}
```

```
typedef vector<vector<Cell> > OnionBoard;
```

fill in the gaps in the C++ code below for the corresponding *backtracking* algorithm to find one solution, or determining that no solution exists. If there is more than one solution, any solution is good.

```
class Onions {
    int n;
    bool solution_found;
    OnionBoard board; // current partial solution
    vector<int> col; // col[i] = column of the onion at row i
    vector<bool> onion_in_col; // onion_in_col[i] = there is an onion in column i
    vector<bool> onion_in_region; // onion_in_region[i] = there is an onion in region i
```

```
public:
    Onions(const OnionBoard& initial_board) {
        board = initial_board;
        n = board.size();
        col = vector<int>(n);
        onion_in_col = vector<bool>(n, false);
        onion_in_region = vector<bool>(n, false);
        solution_found = false;
        backtrack(0);
    }
```

```

    }

void backtrack(int k) {
    if (k == n) { solution_found = true; return; }
    for (int j = 0; j < n and not solution_found; ++j) {
        int cell_col = 
        if (not onion_in_col[j] and not onion_in_region[cell_col]
            and not onion_in_neighborhood(k, j)) {
            col[k] = j;
            
            backtrack(k+1);
            
        }
    }
}

// returns true iff the partial solution board has an onion in a cell adjacent to (i,j);
// you are not asked to implement it
bool onion_in_neighborhood(int i, int j);

```

Tweets

(4 points)

A social network of the Twitter type has different users. These maintain a ‘follow’ relationship with other users (for instance, Salvador follows Anna, Ivet and Leo Messi, but Leo Messi does not follow anyone). Users can add or remove other users from their list of followed users. Moreover, at any moment, a user can tweet a message. Also, at any time, a user can obtain the k most recent tweets of the users she follows. (That is, if she follows n users, she gets at most k tweets, not nk).

The dimensions of such a system are what one would imagine:

- The number of users is huge; the number of tweets is even bigger.
- The number of users followed by some users may be big, but not huge.
- Some users are followed by many other users.
- The number of tweets per user has a high variability.
- The value of k is relatively small; something between 10 and 1000, say.

Users are represented by a *User* struct that contains its personal data (including a *string* with its name that identifies it in a unique way) and that tweets are represented with a struct *Tweet* that contains the message and its posting time:

```

struct User {
    ...                // personal data
    string name;        // different users have different names

```

```
};

struct Tweet {
    string message;           // body of the message
    int post_time;           // number of milliseconds since some epoch
};
```

Explain which data structures and algorithms you would use to store the described information and to implement the following operations in the most efficient way: adding and removing follow relationships, tweeting a message, and obtaining the k most recent tweets from the users that are followed by a given user. Do not explain how users are created/deleted. Analyse the cost of your operations.

The expected answer is open but be specific defining the *SocialNetwork* data structure and describe (without code) how to implement the following operations:

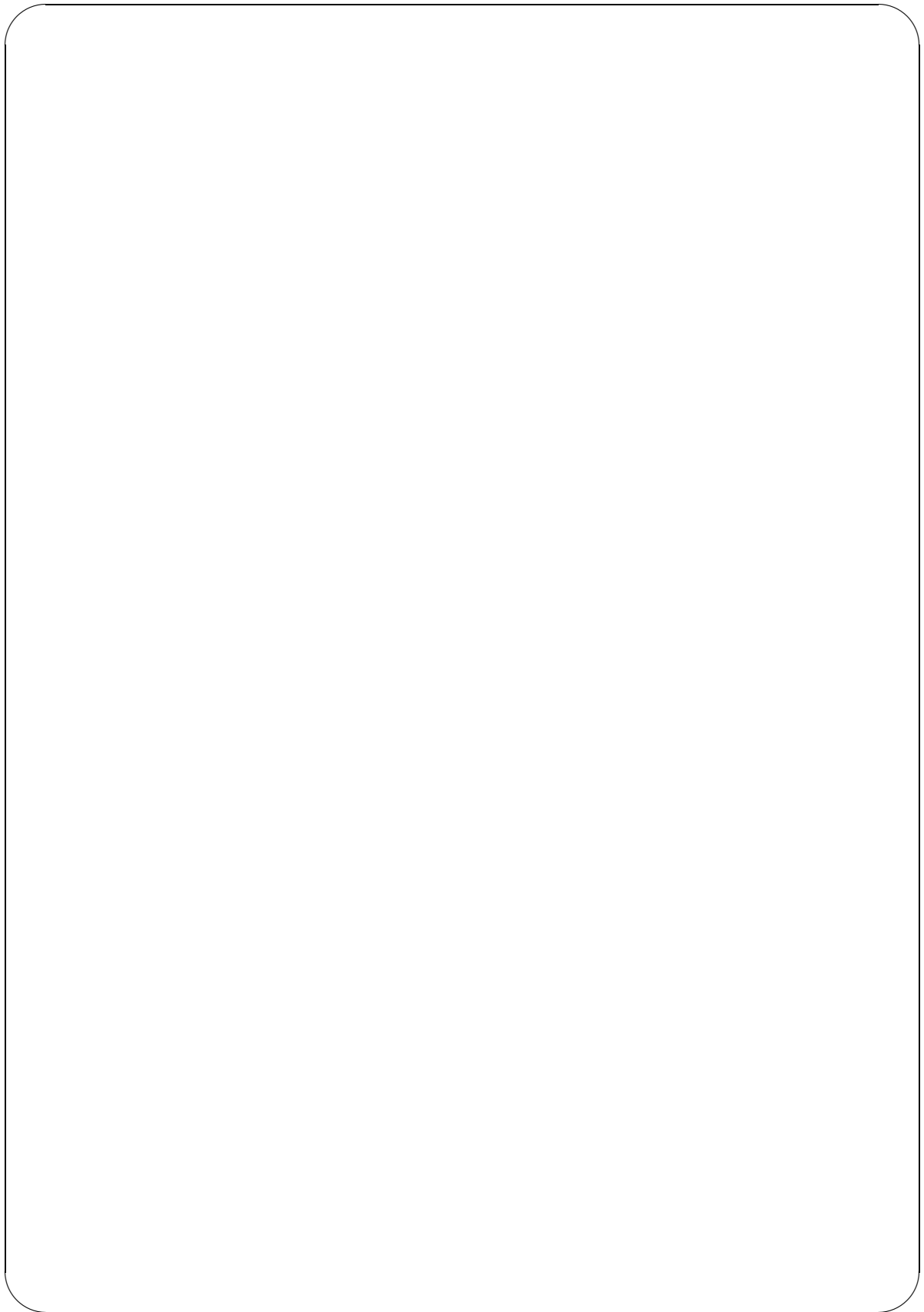
```
struct SocialNetwork { ... };

// u1 follows u2 (if she didn't already); users exist.
void add_follower (SocialNetwork& sn, string u1, string u2);

// u1 unfollows u2 (if she did); users exist.
void remove_follower (SocialNetwork& sn, string u1, string u2);

// u tweets the message m; user exists.
void tweet_message (SocialNetwork& sn, string u, Tweet m);

// Returns a list with the k most recent tweets of the users followed by u; user exists.
list <Tweet> k_most_recent_tweets (const SocialNetwork& sn, string u, int k);
```



Final EDA Exam Length: 3 hours**07/01/2016****Problem 1****(3 pts.)**

Answer the following questions:

(a) (1 pt.) Consider the following decision problems:

- SAT: given a propositional formula, to determine whether it is satisfiable.
- COL: given a graph $G = (V, E)$ and a natural number k , to determine whether G can be painted with k colors so that the extremes of edges are painted with different colors.
- SOR: given an array of n different integers, to determine whether it is sorted in increasing order.

For each of the statements that follow, mark with an 'X' the corresponding column, depending on whether the statement is true, false, or is an open problem and it is still unknown if it is true or false. No justification is needed.

	Cert	Fals	Obert
SOR is in class P			
SOR is in class NP			
SOR is NP-hard			
SAT is in class P			
SAT is in class NP			
SAT is NP-hard			
SOR can be reduced polynomially to COL			
COL can be reduced polynomially to SOR			
SAT cannot be reduced polynomially to SOR			
COL can be reduced polynomially to SOR, and SAT cannot be reduced polynomially to SOR			

(b) (1 pt.) Given a vector of integers v and an integer x , the function

```

int position (const vector<int>& v, int x) {
    int n = v.size ();
    for (int i = 0; i < n; ++i)
        if (v[i] == x) return i;
    return -1;
}

```

returns the first position of v that contains x , or -1 if there is none.

Consider a probability distribution over the input parameters such that the probability that x is element $v[i]$ is $\frac{1}{n}$ for $0 \leq i < n$.

Answer: the average-case cost in time of *position* as a function of n is $\Theta(\text{ } \boxed{\text{ }} \text{ })$.

Justification:

- (c) (1 pt.) What does Floyd-Warshall's algorithm compute? What is its cost in time and space in the worst case? (no justification is needed)

Problem 2

(2 pts.)

Let V be a `vector<int>` with n different integers. Consider this code snippet:

```
set<int> S;  
for (int i = 0; i < n; ++i) {  
    S.insert (V[i]);  
}
```

```
    cout << *S.begin() << ' ';  
}
```

- (a) (0.4 pts.) What does the code snippet write if $n = 6$ and the elements of V are 42 100 23 12 20 24 (in this order)?

- (b) (0.4 pts.) Give a high-level explanation on which values are written by the code snippet with an arbitrary V with n different integers.

- (c) (0.4 pts.) Justify what is the asymptotic cost in time of the code snippet in the worst case as a function of n .

Hint: You can use Stirling's formula: $\log(n!) = \Theta(n \log n)$.

- (d) (0.4 pts.) If the elements of V were repeated, could the code snippet compute something different from what was answered in exercise (b)? If the answer is yes, show an example. If the answer is no, explain why.

- (e) (0.4 pts.) Write with full detail an alternative program to the given code snippet that computes the same for an arbitrary vector V with n different integers ($n \geq 1$), but with asymptotic cost in time strictly less than that computed in exercise (c). Justify what is this asymptotic cost.

Problem 3**(2 pts.)**

Consider the problem of, given a directed acyclic graph $G = (V, E)$, generating all its topological orderings. The next program solves it (assume $V = \{0, 1, \dots, n - 1\}$):

```

#include <iostream>
#include <vector>
using namespace std;

typedef vector<int> AdjList;
typedef vector<AdjList> Graph;

// implementation not given here
Graph read_graph ();
void print_solution (const vector<int>& sol);

bool ok(const Graph& G, const vector<int>& sol) {
    int n = sol.size ();
    vector<bool> mar(n, false);
    for (int i = 0; i < n; ++i) {
        int x = sol[i];
        if (mar[x]) return false;
        mar[x] = true;
        for (int y : G[x])
            for (int j = 0; j < i; ++j)
                if (sol[j] == y) return false;
    }
    return true;
}

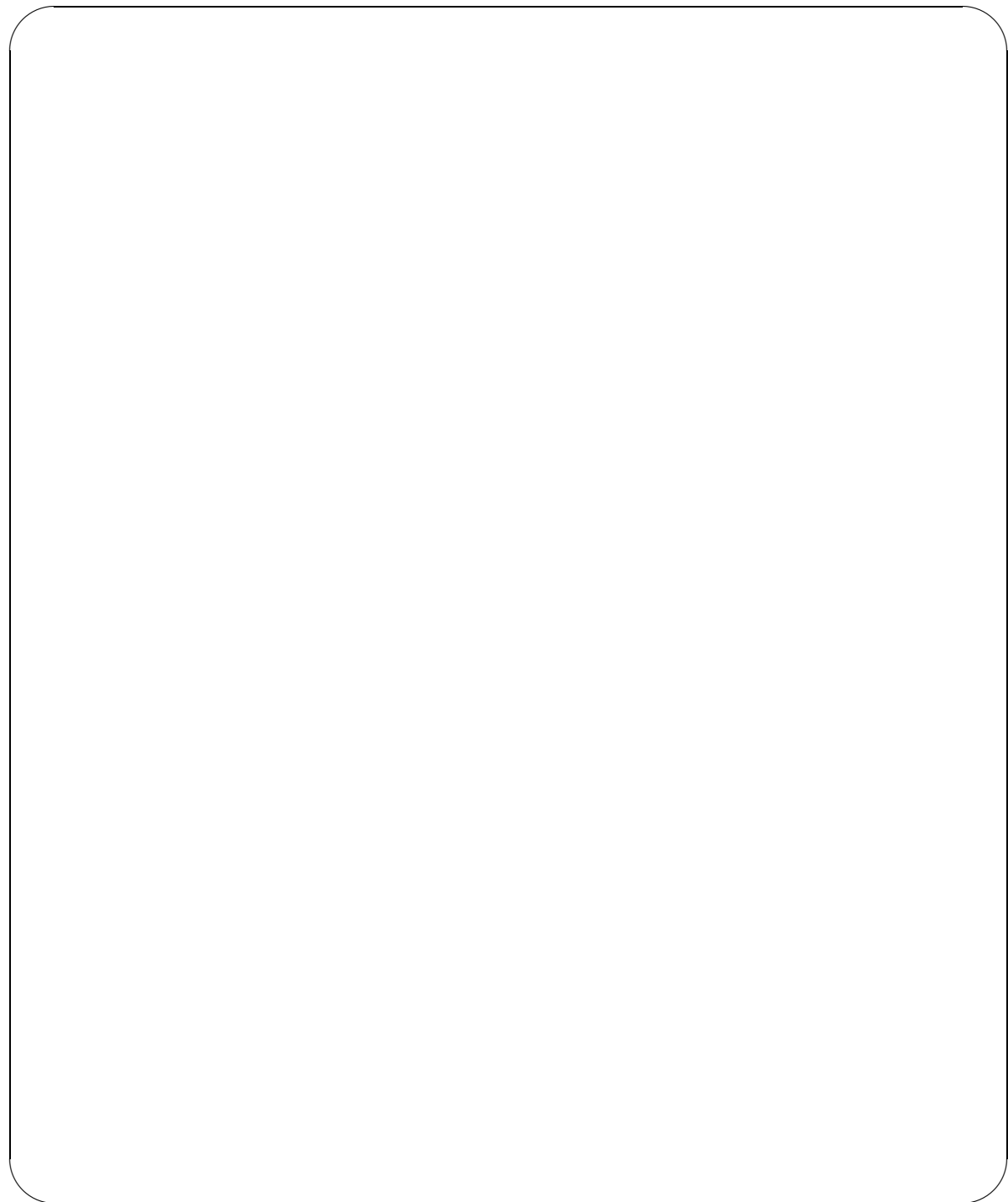
void top_sorts_rec (int k, const Graph& G, vector<int>& sol) {
    int n = sol.size ();
    if (k == n) {
        if (ok(G, sol))
            print_solution (sol);
    }
    else
        for (int x = 0; x < n; ++x) {
            sol[k] = x;
            top_sorts_rec (k+1, G, sol);
        }
}

void top_sorts (const Graph& G, vector<int>& sol) {
    top_sorts_rec (0, G, sol);
}

int main() {
    Graph G = read_graph ();
    vector<int> sol(G.size ());
    top_sorts (G, sol);
}

```

However, it is too slow. Rewrite function `top_sorts` to solve the problem more efficiently, without changing `main`. Implement also the auxiliary functions that you use, except for `print_solution` and the functions of the standard C++ library.

**Problem 4****(3 pts.)**

Let G be the adjacency matrix of a directed graph with n vertices. The *closure matrix* of (the graph represented by) G , which we will represent with G^* , is an $n \times n$ matrix defined as follows: given u, v such that $0 \leq u, v < n$, the coefficient in row u and column v is

$$G_{uv}^* = \begin{cases} 1 & \text{if there is a (possibly empty) path from } u \text{ to } v \text{ in the graph} \\ 0 & \text{otherwise} \end{cases}$$

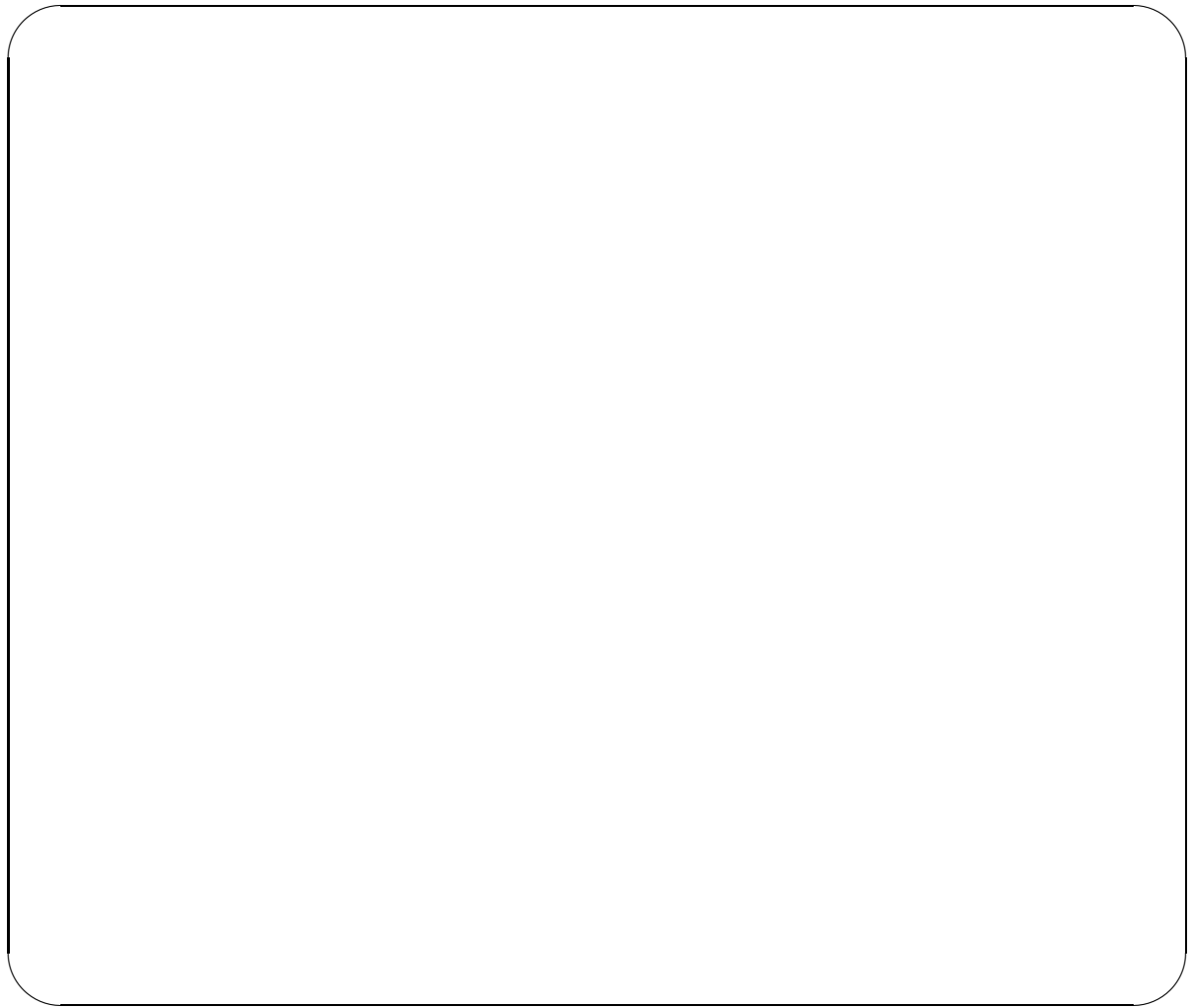
where the vertices of the graph are identified with the integers from 0 to $n - 1$ as usual.

- (a) (1 pt.) Give a high-level description of how to implement a function

```
vector<vector<int>> closure(const vector<vector<int>>& G);
```

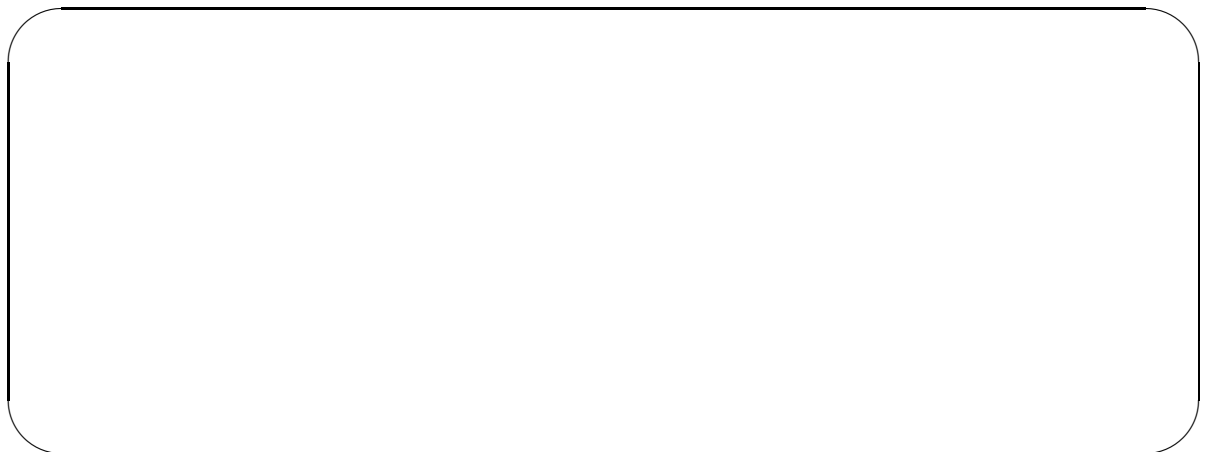
which, given the adjacency matrix G of a directed graph, returns its closure. Analyze the asymptotic cost in time in the worst case.

- (b) (1 pt.) Let us denote by I the $n \times n$ identity matrix, that is, the matrix where all coefficients are 0 except for those in the diagonal, which are 1. Show that, for any $k \geq 0$, there is a path in the graph from u to v with at most k edges if and only if the coefficient in row u column v of the matrix $(I + G)^k$ (the matrix resulting from multiplying k matrices $I + G$) is different from 0, i.e., $(I + G)_{uv}^k \neq 0$.



- (c) (1 pt.) Give a high-level description of an algorithm for, given the adjacency matrix of a directed graph G , computing its closure G^* in time strictly better than $\Theta(n^3)$ in the worst case. Give the cost in the worst case and justify it. Assume that the cost of arithmetic operations with integers is constant.

Hint: if there is a path from u to v , there is at least one with at most $n - 1$ edges.



Final EDA Exam Length: 3 hours

07/01/2016

Problem 1

(1.5 pts.)

Answer the following questions:

- (a) (0.9 pt.) The master theorem for dividing recurrences claims that if we have a recurrence of the form $T(n) = aT(n/b) + \Theta(n^k)$ with $a > 0$, $b > 1$ and $k \geq 0$, then, letting $\alpha = \log_b a$,

$$T(n) = \begin{cases} \boxed{} & \text{if } \alpha < k, \\ \boxed{} & \text{if } \alpha = k, \\ \boxed{} & \text{if } \alpha > k. \end{cases}$$

- (b) (0.3 pt.) The smallest k such that the following statement is true:

“For all directed graph $G = (V, E)$, it holds that $|E| = O(|V|^k)$ ”

is .

- (c) (0.3 pt.) Class `stack<T>` of STL has two member functions `top`:

```
T& top();
const T& top() const;
```

On the other hand, class `priority_queue<T>` has one function `top` only:

```
const T& top() const;
```

What is the problem with class `priority_queue<T>` having a function `T& top()`?

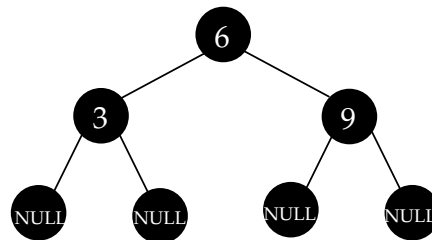
Problem 2

(1.5 pts.)

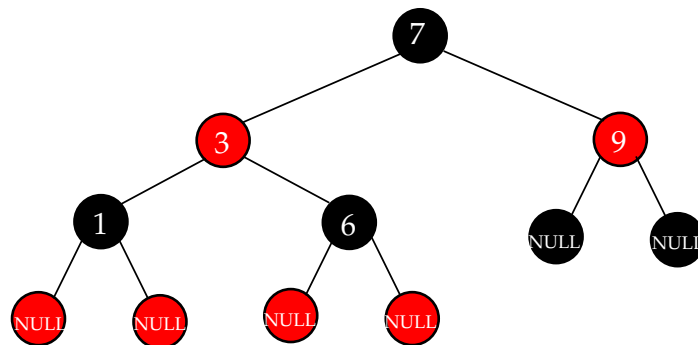
In this exercise, shaded nodes represent red nodes.

Wrong answers penalize with the same grade as right ones reward.

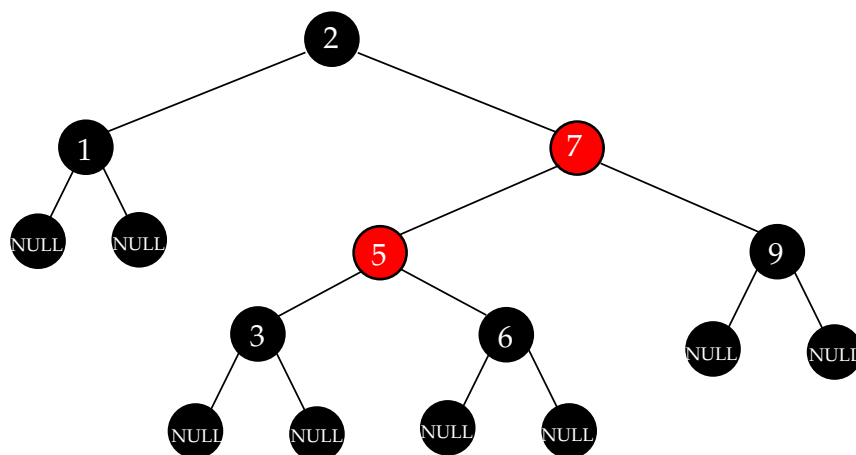
(a) The following binary tree is a red-black tree (answer **Yes** or **No**):



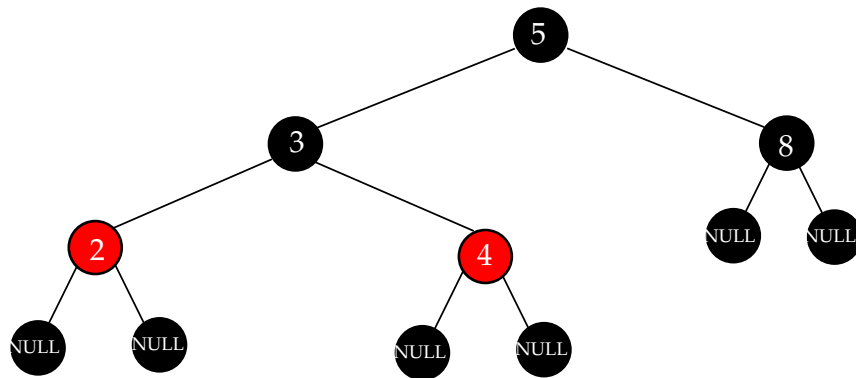
(b) The following binary tree is a red-black tree (answer **Yes** or **No**):



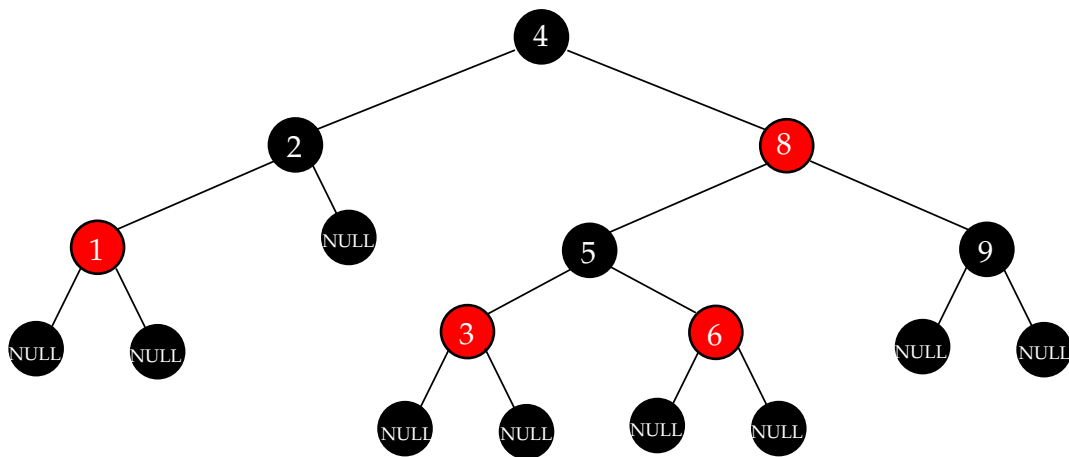
(c) The following binary tree is a red-black tree (answer **Yes** or **No**):



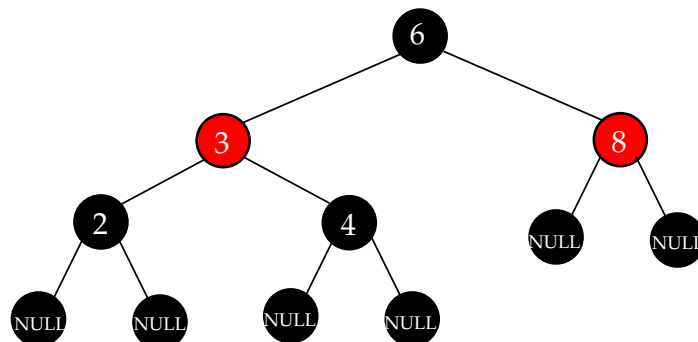
(d) The following binary tree is a red-black tree (answer **Yes** or **No**):



(e) The following binary tree is a red-black tree (answer **Yes** or **No**):



(f) The following binary tree is a red-black tree (answer **Yes** or **No**):



Problem 3

(2 pts.)

Let $s \in \mathbb{R}$ be such that $0 < s < 1$.

Consider the following variant of binary search which, given:

- an element x ,

- a vector v that is sorted in non-decreasing order, and
- two positions l and r ,

returns a position between l and r , both included, in which x appears in v (or -1 if there is none):


double s ; // $0 < s < 1$

```
int position (double  $x$ , const vector<double>&  $v$ , int  $l$ , int  $r$ ) {  
    if ( $l > r$ ) return  $-1$ ;  
    int  $p = l + \text{int}(s*(r-l))$ ;  
    if ( $x < v[p]$ ) return position ( $x$ ,  $v$ ,  $l$ ,  $p-1$ );  
    if ( $x > v[p]$ ) return position ( $x$ ,  $v$ ,  $p+1$ ,  $r$  );  
    return  $p$ ;  
}
```

- (a) (0.75 pt.) According to the value of the parameter s , give a recurrence that describes the cost in time in the worst case of *position* as a function of $n = r-l+1$.



- (b) (0.5 pt.) According to the value of s , when does that worst case take place?



- (c) (0.75 pt.) According to the value of s , solve the recurrence of the answer of exercise (a) and give the asymptotic cost in the worst case.

Problem 4**(2 pts.)**

A *deque* $\langle T \rangle$ (acronym of *double-ended queue*, pronounced: “dèk”) is a sequential container of objects of type T that allows inserting and deleting elements both at the beginning as well as at the end, by means of the following functions:

```
void push_front (const  $T\&$   $x$ );  
void pop_front ();
```

```
void push_back (const  $T\&$   $x$ );  
void pop_back ();
```

respectively. Also, one can recover the element at the beginning with the functions:

```
 $T\&$  front ();  
const  $T\&$  front () const;
```

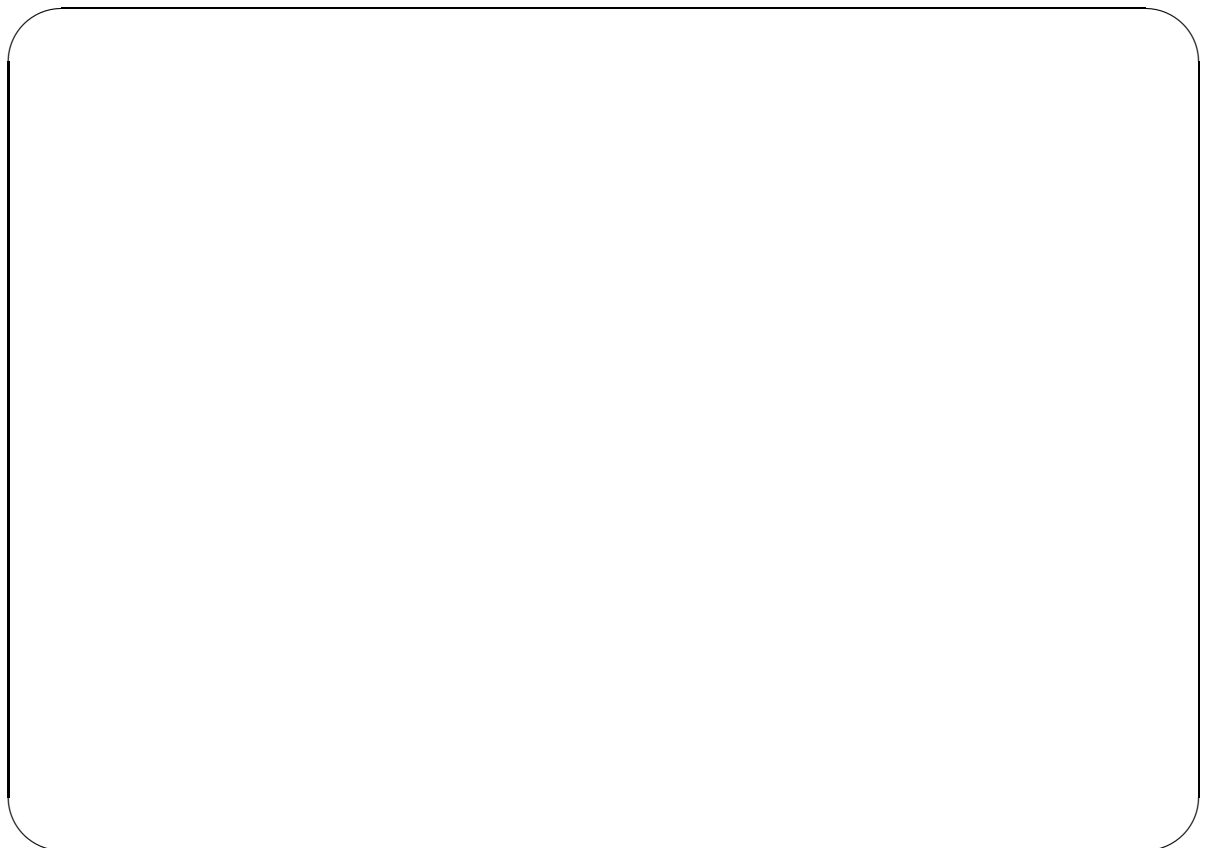
All these functions have cost in time $\Theta(1)$.

- (a) (1 pt.) A weighted directed graph $G = (V, E)$ is *binary* if for any edge $e \in E$, its weight is 0 or 1. By using a *deque*, implement in C++ a function

```
int minimum_cost(const vector<vector<pair<int,int>>>& G, int x, int y);
```

that, given a binary graph G and two vertices x and y of G , returns the minimum cost of going from x to y (or -1 if it is not possible to go from x to y).

Assume vertices are represented with integers between 0 and $n - 1$, where n is the number of vertices; and that the graph is represented with adjacency lists with a `vector<vector<pair<int,int>>>` where the first component of the *pairs* represents the successor vertex and the second one, the weight of the edge.



- (b) (0.5 pt.) Analyze the cost in time of your function `minimum_cost` in the worst case as a function of the number of vertices $|V|$ and edges $|E|$.

- (c) (0.5 pt.) Which cost in time in the worst case would *minimum_cost* have if we used the version of Dijkstra's algorithm that implements the priority queue with a binary heap (like the ones seen at class)?

Problem 5**(3 pts.)**

The problem of DISEQUALITIES consists in, given an interval of integers and a set of disequalities (\neq) between variables, to determine whether there is a solution to all disequalities with values in the interval. More formally, given:

- a (finite) interval $[l, u] \subset \mathbb{Z}$,
- a set of variables V , and
- a set D of disequalities of the form $x \neq y$, where $x, y \in V$,

we have to determine where there is $s : V \rightarrow \mathbb{Z}$ such that:

- for any $x \in V$ we have $s(x) \in [l, u]$, and
- for any $x \neq y \in D$ it holds $s(x) \neq s(y)$.

We will consider that the set of variables is of the form $V = \{x_0, x_1, \dots, x_{n-1}\}$ for a certain $n > 0$, and will identify the variables with integers between 0 and $n - 1$. Besides, we will represent the inputs for the problem of DISEQUALITIES by means of the type *inp_DISEQUALITIES* defined as follows:

```
struct inp_DISEQUALITIES {
    int l, u, n;
    set<pair<int,int>> D; // each pair (i,j) is a disequality  $x_i \neq x_j$ 
};
```

(a) (1 pt.) Complete the following implementation of the function

```
bool has_solution (const inp_DISEQUALITIES& e);
```

that solves the problem of DISEQUALITIES:

```
bool ok(const vector<int>& s, const set<pair<int,int>>& D) {
    for (auto d : D)
        if (  )
            return false;
    return true;
}

bool has_solution (int k, vector<int>& s, const inp_DISEQUALITIES& e) {
    if (k == e.n) return ok(s, e.D);
    for (int v =  ; v ≤  ; ++v) {
        
        if ( has_solution (k+1, s, e) ) return true;
    }
    return false;
}

bool has_solution (const inp_DISEQUALITIES& e) {
    vector<int> s(e.n);
    return has_solution (  , s, e);
}
```

(b) (0.75 pt.) Prove that the cost in time in the worst case of the previous implementation of *has_solution* is $\Omega((u - l + 1)^n)$.

(c) (0.75 pt.) Consider the problem of COLORING: given an undirected graph G and a natural number $c > 0$, to determine whether one can paint the vertices of G with c colors in such a way that any edge has the endpoints painted with different colors.

Suppose that we represent the inputs for the problem of COLORING with the type *inp_COLORING* defined as follows:

```
struct inp_COLORING {  
    vector<vector<int>> G; // graph represented with adjacency lists  
    int c;  
};
```

Implement a polynomial reduction from COLORING to DISEQUALITIES:

```
inp_DISEQUALITIES reduction(const inp_COLORING& ec);
```

(d) (0.5 pt.) Do you see feasible to find a polynomial algorithm for DISEQUALITIES? Why?

Final EDA Exam Length: 3 hours**12/01/2017****Problem 1****(2 pts.)**

Fill the following gaps as precisely as possible:

- (a) (0.25 pts.) A graph with n vertices has $O(\text{ } \boxed{\text{ }} \text{ })$ edges.
- (b) (0.25 pts.) A connected graph with n vertices has $\Omega(\text{ } \boxed{\text{ }} \text{ })$ edges.
- (c) (0.25 pts.) A complete graph with n vertices has $\Omega(\text{ } \boxed{\text{ }} \text{ })$ edges.
- (d) (0.25 pts.) A min-heap with n vertices has $\Theta(\text{ } \boxed{\text{ }} \text{ })$ leaves.
- (e) (0.25 pts.) A binary search tree with n vertices has height $\Omega(\text{ } \boxed{\text{ }} \text{ })$.
- (f) (0.25 pts.) A binary search tree with n vertices has height $O(\text{ } \boxed{\text{ }} \text{ })$.
- (g) (0.25 pts.) An AVL tree with n vertices has height $\Omega(\text{ } \boxed{\text{ }} \text{ })$.
- (h) (0.25 pts.) An AVL tree with n vertices has height $O(\text{ } \boxed{\text{ }} \text{ })$.

Problem 2**(3 pts.)**

Let $G = (V, E)$ be a directed graph with weights $\omega : E \rightarrow \mathbb{R}$. We want to solve the problem of, given a vertex $s \in V$, to compute the distance from s to all vertices of the graph.

Recall that:

- a *path* is a sequence of vertices that are connected consecutively by arcs; that is to say, (u_0, u_1, \dots, u_k) such that for all $1 \leq i \leq k$, we have $(u_{i-1}, u_i) \in E$.
- the *weight* of a path is the sum of the weights of its arcs:

$$\omega(u_0, u_1, \dots, u_k) = \sum_{i=1}^k \omega(u_{i-1}, u_i).$$

- the *distance* of a vertex u to a vertex v is the weight of the path (called *minimum path*) with minimum weight among those leaving from u and arriving at v , if it exists.
- (a) (0.2 pts.) Let us assume that for any edge $e \in E$, we have $\omega(e) = 1$; that is to say, all weights are 1. Which algorithm can we use to solve efficiently the problem of the distances?

- (b) (0.2 pts.) Let us assume that for any edge $e \in E$, we have $\omega(e) \geq 0$; that is to say, all weights are non-negative. Which algorithm can we use to solve efficiently the problem of the distances?

- (c) (0.2 pts.) Which algorithm can we use to solve efficiently the problem of the distances, if some weights can be negative?

- (d) (0.4 pts.) Which condition over the cycles of the graph must be satisfied so that, for all pairs of vertices $u, v \in V$, there exists a minimum path from u to v ?

- (e) (1 pt.) A function $\pi : V \rightarrow \mathbb{R}$ is a *potential* of the graph if it satisfies that, for any edge $(u, v) \in E$, we have $\pi(u) - \pi(v) \leq \omega(u, v)$. Moreover, the *reduced weights* ω_π are defined as $\omega_\pi(u, v) = \omega(u, v) - \pi(u) + \pi(v)$ for any $(u, v) \in E$.

Prove that if c is a path from u to v then $\omega_\pi(c) = \omega(c) - \pi(u) + \pi(v)$.

- (f) (1 pt.) Suppose that the graph has a potential π . Then, it can be proved that for all pairs of vertices $u, v \in V$ there is a minimum path with weights ω from u to v . Assuming this fact, explain how to use the potential π to compute the distance from a given vertex s to all vertices of the graph with an alternative algorithm to that of part (c) when the weights can be negative.

Problem 3**(2 pts.)**

We define a type *matrix* to represent square matrices of real numbers. Consider the following program:

```
typedef vector<vector<double>> matrix;

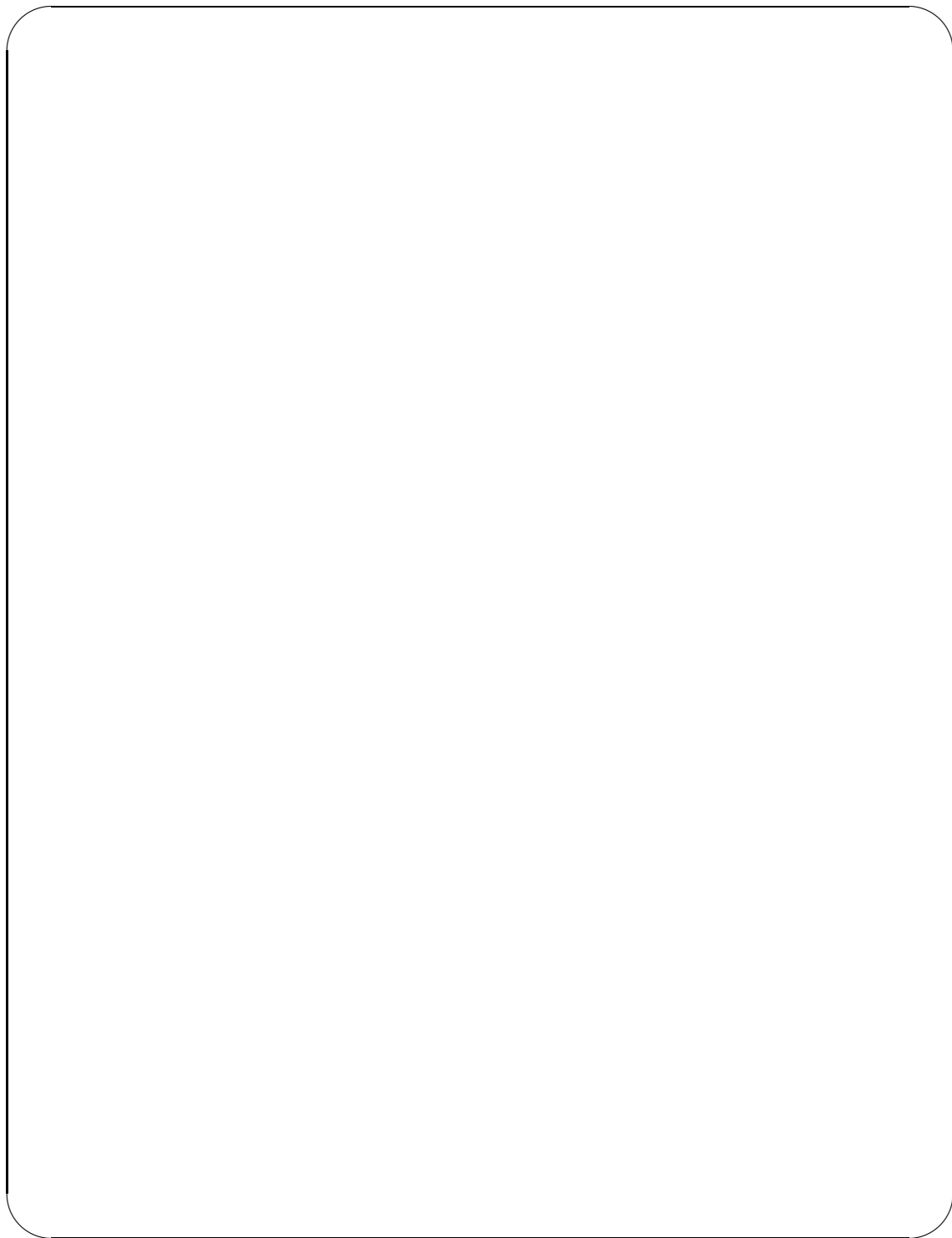
matrix aux(const matrix& A, const matrix& B) {
    int n = A.size ();
    matrix C(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                C[i][j] += A[i][k] * B[k][j];
    return C;
}

matrix mystery(const matrix& M) {
    int n = M.size ();
    matrix P(M), Q(M);
    for (int i = 1; i < n; ++i) {
        P = aux(P, M);
        Q = aux(Q, P);
    }
    return Q;
}
```

- (a) (0.5 pts.) What does the function `matrix_mystery(const matrix& M)` compute in terms of the matrix M ?

- (b) (0.5 pts.) If M is an $n \times n$ matrix, what is the cost in the worst case of the function `matrix_mystery(const matrix& M)` as a function of n ? Justify your answer.

- (c) (1 pt.) Implement in C++ a function that computes the same as the function `matrix_mystery(const matrix& M)` and which takes $\Theta(n^3 \log n)$ time in the worst case. Also implement the auxiliary functions you use, except for `aux` and the functions of the standard library of C++. Justify that the cost of your function is as required.

**Problem 4****(3 pts.)**

The problem of HAMILTONIAN GRAPH consists in, given an (undirected) graph, to decide whether it is Hamiltonian, that is to say, whether there exists a cycle that visits all its vertices exactly once. It is well-known that HAMILTONIAN GRAPH is an NP-complete problem.

It is also known that, from the proof that a problem belongs to class NP, one can derive a brute force algorithm that solves it. The following function **bool ham(const vector<vector<int>>& G)** which, given a graph *G* represented with adjacency lists, returns if *G* is Hamiltonian, implements this algorithm in the case of HAMILTONIAN GRAPH.

```

1  bool ham_rec(const vector<vector<int>>& G, int k, vector<int>& p) {
2      int n = G.size ();
3      if (k == n) {
4          vector<bool> mkd(n, false);
5          for (int u : p) {
6              if (mkd[u]) return false;
7              mkd[u] = true;
8          }
9          for (int k = 0; k < n; ++k) {
10             int u = p[k];
11             int v;
12             if (k < n-1) v = p[k+1];
13             else v = p[0];
14             if (find(G[u].begin(), G[u].end(), v) == G[u].end()) return false;
15         }
16         return true;
17     }
18     for (int v = 0; v < n; ++v) {
19         p[k] = v;
20         if (ham_rec(G, k+1, p)) return true;
21     }
22     return false;
23 }
24
25 bool ham(const vector<vector<int>>& G) {
26     int n = G.size ();
27     vector<int> p(n);
28     return ham_rec(G, 0, p);
29 }
```

Note: The function of the STL library

Iterator find (Iterator first , Iterator last , int val);

returns an iterator to the first element in the range *[first,last)* that compares equal to *val*. If no such element exists, the function returns last.

- (a) (0.5 pts.) Identify the variable in the previous program which represents the witness when the function *ham* returns **true**.

- (b) (0.5 pts.) Identify the code corresponding to the verifier in the previous program. To that end, use the line numbers on the left margin.

- (c) (1 pt.) Fill the following gaps so that the function *ham2* computes the same as the function *ham*, but more efficiently.

```

bool ham2_rec(const vector<vector<int>>& G, int k, int u, vector<int>& next) {
    int n = G.size ();
    if (  == n)
        return find(G[u].begin (), G[u].end (),  ) != G[u].end();

    for (int v : G[u])
        if (next[v] ==  ) {
            next[u] =  ;
            if (ham2_rec(G, k+1, v, next)) return true;
            next[u] = -1;
        }
    return false;
}

bool ham2(const vector<vector<int>>& G) {
    int n = G.size ();
    vector<int> next(n, -1);
    return ham2_rec(G,  , 0, next);
}

```

- (d) (0.5 pts.) Suppose that the adjacency lists of the representation of *G* are sorted (for example, increasingly). Explain how to use this to make the function of part (c) more efficient.

- (e) (0.5 pts.) Suppose that *G* is a **disconnected** graph. Explain how to use this to make the function of part (c) more efficient.

Solutions to Mid Term Exams

Solution Midterm Exam EDA

31/03/2016

Proposed solution to problem 1

- (a) $\Theta(n^{\log 7})$
- (b) The master theorem for subtractive recurrences states that given a recurrence $T(n) = aT(n - c) + \Theta(n^k)$ with $a, c > 0$ and $k \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } a < 1, \\ \Theta(n^{k+1}) & \text{if } a = 1, \\ \Theta(a^{\frac{n}{c}}) & \text{if } a > 1. \end{cases}$$

- (c) $\Theta(n)$

Proposed solution to problem 2

- (a) The cost of the program is determined by the sum of the costs of two groups of instructions:

- A.** The instructions that are executed at each iteration of the loop: the evaluation of the condition $i \leq n$, the call $f(i)$, the evaluation of the condition $i == p$ and the increment $i++$.
- B.** The instructions that are executed when the condition of the **if** is true: the call $g(n)$ and the increment $p *= 2$.

We count separately the contribution to the total cost of the two groups of instructions:

- A.** If the cost of $f(m)$ is $\Theta(m)$, then at the i -th iteration the cost of **A** is $\Theta(i)$. So in total the contribution to the cost is $\sum_{i=1}^n \Theta(i) = \Theta(\sum_{i=1}^n i) = \Theta(n^2)$.
- B.** If the cost of $g(m)$ is $\Theta(m)$, then every time that the condition of the **if** is true the cost of **B** is $\Theta(n)$. Since this happens $\Theta(\log n)$ times, the contribution to the total cost is $\Theta(n \log n)$.

Hence in total the cost of $h(n)$ is $\Theta(n^2) + \Theta(n \log n) = \Theta(n^2)$.

- (b) Let us consider the same two groups of instructions of the previous exercise, and again we count separately their contribution to the total cost:

- A.** If the cost of $f(m)$ is $\Theta(1)$, then the cost of **A** at each iteration is $\Theta(1)$. Since $\Theta(n)$ are performed, the cost in total is $\Theta(n)$.
- B.** If the cost of $g(m)$ is $\Theta(m^2)$, then every time that the condition of the **if** is true the cost of **B** is $\Theta(n^2)$. Since this happens $\Theta(\log n)$ times, the cost is $\Theta(n^2 \log n)$.

Hence in total the cost of $h(n)$ is $\Theta(n) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$.

Proposed solution to problem 3

- (a) The i -th row uses $i + 1$ cells ($0 \leq i < n$). In total, the consumed space is

$$\sum_{i=0}^{n-1} (i + 1) = \sum_{j=1}^n j = \Theta(n^2).$$

- (b) A possible solution:

```

int p(int x) { return x*(x+1)/2;}

int mystery(int k, int l, int r) {
    if (l+1 == r) return l;
    int m = (l+r)/2;
    if (p(m) ≤ k) return mystery(k, m, r);
    else          return mystery(k, l, m);
}

pair<int,int> row_column(int n, int k) {
    if (k < 0 or k ≥ p(n)) return {-1,-1};
    int i = mystery(k, 0, n);
    return {i, k - p(i)};
}

```

- (c) Let $C(N)$ be the cost in the worst case of a call to the function $mystery(k, l, r)$, where $N = r - l + 1$. The recurrence that describes $C(N)$ is

$$C(N) = C(N/2) + \Theta(1),$$

since one makes one recursive call over an interval of half the size, and operations that take constant time. By using the master theorem of divisive recurrences, the solution to the recurrence is $C(N) = \Theta(\log N)$.

Therefore, the cost in the worst case of a call to function $row_column(n, k)$ is $\Theta(\log n)$.

- (d) The index i of the searched row is the natural number $0 \leq i < n$ such that $p(i) \leq k < p(i + 1)$. We can compute it by solving the second degree equation $p(x) = k$ and taking $i = \lfloor x \rfloor$:

```

int p(int x) { return x*(x+1)/2;}

pair<int,int> row_column(int n, int k) {
    if (k < 0 or k ≥ p(n)) return {-1,-1};
    int i( floor (( sqrt (1.+8*k) - 1)/2));
    return {i, k - p(i)};
}

```

Proposed solution to problem 4

- (a) Let us show it by contradiction. Let us assume that i is such that $0 \leq i < n-1$ i $f_A(i+1) < f_A(i)$. Let us take $k = i+1$, $j = f_A(i+1)$ and $l = f_A(i)$, so that $0 \leq i < k < n$ and $0 \leq j < l < n$. Since $j = f_A(k)$, we have that $A_{k,j} \leq A_{k,l}$. And as $l = f_A(i)$, we have that $A_{i,l} \leq A_{i,j}$; in fact, since $j < l$, it must be $A_{i,l} < A_{i,j}$. Thus, by adding we have that $A_{i,l} + A_{k,j} < A_{i,j} + A_{k,l}$. This contradicts that A is a Monge matrix.
- (b) For each even i , one can compute the column where the leftmost minimum of the i -th row of A appears as follows. From the recursive call over B_1 , we have the column j_1 where the leftmost minimum of the i -th row of A appears between columns 0 and $\frac{n}{2} - 1$. Similarly, from the recursive call over B_2 , we have the column j_2 where the leftmost minimum of the i -th row of A appears between columns $\frac{n}{2}$ and $n-1$. Hence, $f_A(i) = j_1$ if $A_{i,j_1} \leq A_{i,j_2}$, and $f_A(i) = j_2$ otherwise.

For each odd i , one determines $f_A(i)$ by examining the columns between $f_A(i-1)$ and $f_A(i+1)$ (defining $f_A(n) = n-1$ for notational convenience), and choosing the index where the leftmost minimum appears. This has cost $\Theta(f_A(i+1) - f_A(i-1) + 1)$. In total:

$$\sum_{i=1, i \text{ odd}}^{n-1} \Theta(f_A(i+1) - f_A(i-1) + 1) = \Theta((n-1) - f_A(0) + \frac{n}{2}) = \Theta(n)$$

since $0 \leq f_A(0) < n$.

- (c) Let $C(n)$ be the cost of the proposed algorithm for computing the function f_A for all rows of a Monge matrix A of size $n \times n$.

Apart from the two recursive calls in step (2) over matrices of size $\frac{n}{2} \times \frac{n}{2}$, steps (1) and (3) require time $\Theta(n)$ in total. So the recurrence that describes the cost is

$$C(n) = 2C(n/2) + \Theta(n).$$

By using the master theorem of divisive recurrences, the solution to the recurrence is $C(n) = \Theta(n \log n)$.

Solution Midterm Exam EDA

07/11/2016

Proposed solution to problem 1

	<i>Best case</i>	<i>Average case</i>	<i>Worst case</i>
(a) Quicksort (with Hoare's partition)	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Insertion	$\Theta(n)$	Do not fill	$\Theta(n^2)$

(b) $\Theta(\sqrt{n})$ (c) $\Theta(\sqrt{n} \log n)$ (d) $\Theta(n)$ (e) Karatsuba's algorithm computes the product of two natural numbers of n bits in time $\Theta(n^{\log_2 3})$.(f) Strassen's algorithm computes the product of two matrices of size $n \times n$ in time $\Theta(n^{\log_2 7})$.

Proposed solution to problem 2

(a) A possible solution:

#include <vector>**using namespace std;**

```
bool dic_search (const vector<int>& a, int l, int r, int x) {
    if (l > r) return false;
    int m = (l+r)/2;
    if (a[m] < x) return dic_search (a, m+1, r, x);
    if (a[m] > x) return dic_search (a, l, m-1, x);
    return true;
}
```

```
bool search (const vector<int>& a, int l, int r, int x) {
    if (l+1 == r) return a[l] == x or a[r] == x;
    int m = (l+r)/2;
    if (a[m] ≥ a[l]) {
        if (a[l] ≤ x and x ≤ a[m]) return dic_search (a, l, m, x);
        else return search (a, m, r, x);
    }
    else {
        if (a[m] ≤ x and x ≤ a[r]) return dic_search (a, m, r, x);
        else return search (a, l, m, x);
    }
}
```

```
bool search (const vector<int>& a, int x) {
```

```

    return search(a, 0, a.size()-1, x);
}

```

- (b) Let $C(n)$ be the cost of dealing with a vector of size n (be it with function *search* or with function *dic_search*) in the worst case (for example, when the element x does not appear in the sequence). Apart from operations of constant cost (arithmetic computations, comparisons and assignments between integers, vector accesses), exactly one call is made over a vector of size half of that of the input. Therefore, the cost is determined by the recurrence:

$$C(n) = C(n/2) + \Theta(1),$$

which, by the master theorem of divisive recurrences, has solution $C(n) = \Theta(\log(n))$.

Proposed solution to problem 3

- (a) It computes m^n .
- (b) The cost of the function is determined by the cost of the loop. Each iteration requires time $\Theta(1)$, since only arithmetic operations and integer assignments are performed. Therefore, the cost is proportional to the number of iterations. We observe that if y is even, then y is reduced to half its value. And if y is odd with $y > 1$, then at the next iteration the value $y - 1$ is considered, which is even, and then it is reduced to half its value. Hence if $n \geq 1$ the number of iterations is between $1 + \lfloor \log(n) \rfloor$ and $1 + 2\lfloor \log(n) \rfloor$. Altogether, the cost is $\Theta(\log(n))$.

Proposed solution to problem 4

- (a) We have that $\phi - 1 = \frac{\sqrt{5}+1}{2} - 1 = \frac{\sqrt{5}-1}{2}$, and then

$$\phi \cdot (\phi - 1) = \frac{\sqrt{5}+1}{2} \cdot \frac{\sqrt{5}-1}{2} = \frac{(\sqrt{5}+1)(\sqrt{5}-1)}{4} = \frac{5-1}{4} = 1$$

Hence $\phi^{-1} = \phi - 1$.

- (b) By induction over n :

- **Base case $n = 0$:** we have $F(0) = 0$, and

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} \Big|_{n=0} = \frac{1-1}{\sqrt{5}} = 0.$$

- **Base case $n = 1$:** we have $F(1) = 1$, and

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} \Big|_{n=1} = \frac{\phi + \phi^{-1}}{\sqrt{5}} = \frac{\sqrt{5}}{\sqrt{5}} = 1.$$

- **Inductive case $n > 1$:** by induction hypothesis,

$$F(n) = F(n-2) + F(n-1) = \frac{\phi^{n-2} - (-\phi)^{-(n-2)}}{\sqrt{5}} + \frac{\phi^{n-1} - (-\phi)^{-(n-1)}}{\sqrt{5}} =$$

$$\begin{aligned}
&= \frac{\phi^{n-1}(\phi^{-1} + 1) - (-\phi)^{-(n-1)}(-\phi + 1)}{\sqrt{5}} = \frac{\phi^{n-1} \cdot \phi - (-\phi)^{-(n-1)}(-\phi^{-1})}{\sqrt{5}} = \\
&= \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}
\end{aligned}$$

(c) We have that

$$\lim_{n \rightarrow +\infty} \frac{F(n)}{\phi^n} = \lim_{n \rightarrow +\infty} \frac{\frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}}{\phi^n} = \lim_{n \rightarrow +\infty} \frac{1 - (\frac{-1}{\phi^2})^n}{\sqrt{5}} = \frac{1}{\sqrt{5}}$$

since $\phi > 1$, $\phi^2 > 1$ and $\lim_{n \rightarrow +\infty} (\frac{-1}{\phi^2})^n = 0$. So $F(n) = \Theta(\phi^n)$.

5

Solutions to Lab Exams

Solution to Lab Exam EDA - shift 1

13/12/2010

Shift 1

Solution to problem 1

```

#include <iostream>
#include <vector>
#include <queue>
#include <cassert>

using namespace std;

const int N_DIRS = 4;
const int di[N_DIRS] = { 1, 0, -1, 0};
const int dj[N_DIRS] = { 0, 1, 0, -1};

struct Pos {
    int i, j;
    Pos(int ii = -1, int jj = -1) : i(ii), j(jj) { }
};

bool ok(int n, int m, int i, int j) {
    return
        0 ≤ i and i < n and
        0 ≤ j and j < m;
}

int search(const vector< vector<char> >& map, int n, int m, int i0, int j0) {
    const int MINFTY = -1;
    vector< vector<int> > dist(n, vector<int> (m, MINFTY));
    queue<Pos> q;
    int max_dist = MINFTY;
    q.push(Pos(i0, j0));
    dist[i0][j0] = 0;
    while (not q.empty()) {
        Pos p = q.front();
        q.pop();
        int i = p.i;
        int j = p.j;
        for(int k = 0; k < N_DIRS; ++k) {
            int ii = i + di[k];
            int jj = j + dj[k];
            if (ok(n, m, ii, jj) and map[ii][jj] ≠ 'X' and dist[ii][jj] == MINFTY) {
                q.push(Pos(ii, jj));
                dist[ii][jj] = 1 + dist[i][j];
                if (map[ii][jj] == 't') max_dist = dist[ii][jj];
            }
        }
    }
}

```



```

    }
    return max_dist;
}

int main(void) {
    int n, m;
    cin >> n >> m;
    vector < vector<char> > map(n, vector<char>(m));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> map[i][j];
    int f, c;
    cin >> f >> c;
    --f;
    --c;
    int dist = search(map, n, m, f, c);
    if (dist ≥ 0) cout << "distancia maxima: " << dist << endl;
    else      cout << "no es pot arribar a cap tresor" << endl;
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<bool> Vec;
typedef vector<Vec > Mat;

void next(int i, int j, int n, int& ni, int& nj) {
    if (j < n-1) {
        ni = i;
        nj = j+1;
    }
    else {
        ni = i+1;
        nj = 0;
    }
}

const int N_DIRS = 8;
const int DI[N_DIRS] = { -1, -1, 0, 1, 1, 1, 0, -1};
const int DJ[N_DIRS] = { 0, -1, -1, -1, 0, 1, 1, 1};

bool ok(int i, int j, int n) {
    return
        0 ≤ i and i < n and

```

```

    0 ≤ j  and  j < n;
}

bool safe(int i, int j, int n, Mat& m) {
    for (int k = 0; k < 8; ++k) {
        int ii = i + DI[k];
        int jj = j + DJ[k];
        if (ok(ii, jj, n) and m[ii][jj]) return false;
    }
    return true;
}

void escriu(int n, const Mat& m) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            if (m[i][j]) cout << 'K';
            else      cout << '.';
        cout << endl;
    }
    cout << "-----" << endl;
}

void search(int i, int j, int n, int r, int s, Mat& m) {
    if (s == r) escriu(n, m);
    else if (i ≠ n) {
        int ni, nj;
        next(i, j, n, ni, nj);
        if (safe(i, j, n, m)) {
            m[i][j] = true;
            search(ni, nj, n, r, s+1, m);
        }
        m[i][j] = false;
        search(ni, nj, n, r, s, m);
    }
}

int main(void) {
    int n, r;
    cin >> n >> r;
    Mat m(n, Vec(n, false));
    search(0, 0, n, r, 0, m);
}

```

Shift 2**Solution to problem 1**

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

typedef map<string, int>::iterator Iterator ;

int main(void) {
    map<string, int> bagmul;
    string command;
    while (cin >> command) {
        if (command == "store") {
            string word;
            cin >> word;
            pair<Iterator, bool> res = bagmul.insert (pair<string,int>(word, 1));
            if (not res.second) ++bagmul[word];
        }
        else if (command == "delete") {
            string word;
            cin >> word;
            Iterator i = bagmul.find(word);
            if (i != bagmul.end()) {
                if (i->second == 1) bagmul.erase(i);
                else --bagmul[word];
            }
        }
        else if (command == "minimum?") {
            if (bagmul.empty()) cout << "indefinite minimum" << endl;
            else {
                Iterator i = bagmul.begin();
                cout << "minimum: "
                     << i->first << ", "
                     << i->second << " time(s)" << endl;
            }
        }
        else {
            if (bagmul.empty()) cout << "indefinite maximum" << endl;
            else {
                Iterator i = bagmul.end();
                --i;
                cout << "maximum: "
                     << i->first << ", "
                     << i->second << " time(s)" << endl;
            }
        }
    }
}
```

```

    }
  }
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> CVec;
typedef vector<CVec> CMat;
typedef vector<bool> BVec;
typedef vector<BVec> BMat;

const int N_DIRS = 4;
const int DR[N_DIRS] = { 0, -1, 0, 1 };
const int DC[N_DIRS] = { 1, 0, -1, 0 };

struct Point {
    int r, c;
    Point(int rr, int cc) : r(rr), c(cc) { }
};

bool ok(int r, int c, int n, int m) {
    return
        0 ≤ r and r < n and
        0 ≤ c and c < m;
}

void search(int re, int ce, int n, int m, const CMat& t, vector<Point>& v, BMat& seen) {
    Point p = v.back();
    if (p.r == re and p.c == ce) {
        for (int i = 0; i < v.size(); ++i)
            cout << t[v[i].r][v[i].c];
        cout << endl;
    }
    else {
        for (int k = 0; k < N_DIRS; ++k) {
            int rr = p.r + DR[k];
            int cc = p.c + DC[k];
            if (ok(rr, cc, n, m) and not seen[rr][cc]) {
                seen[rr][cc] = true;
                v.push_back(Point(rr, cc));
                search(re, ce, n, m, t, v, seen);
                v.pop_back();
                seen[rr][cc] = false;
            }
        }
    }
}

```

```
    }  
  }  
}
```

```
int main(void) {  
  int n, m;  
  cin >> n >> m;  
  CMat t(n, CVec(m));  
  BMat seen(n, BVec(m, false ));  
  for (int i = 0; i < n; ++i)  
    for (int j = 0; j < m; ++j)  
      cin >> t[i][j];  
  int ri, ci, re, ce;  
  cin >> ri >> ci >> re >> ce;  
  seen[ri][ci] = true;  
  vector<Point> v(1, Point(ri, ci ));  
  search(re, ce, n, m, t, v, seen);  
}
```

Solution to Lab Exam EDA

19/5/2011

Solution to problem 1

```

#include <iostream>
#include <vector>

using namespace std;

struct Point {
    int r, c;
    Point(int rr, int cc) : r(rr), c(cc) {}
};

bool ok(int n, int m, const Point& p) {
    return
        0 ≤ p.r and p.r < n and
        0 ≤ p.c and p.c < m;
}

const int N_DIRS_KNIGHT = 8;
const int DR_KNIGHT[8] = {-2, -1, 2, 1, -2, -1, 2, 1};
const int DC_KNIGHT[8] = {-1, -2, -1, -2, 1, 2, 1, 2};

const int N_DIRS_BISHOP = 4;
const int DR_BISHOP[4] = {1, -1, 1, -1};
const int DC_BISHOP[4] = {1, 1, -1, -1};

int dfs(int n, int m, const Point& p,
        vector< vector<char> >& map, vector< vector<bool> >& marked,
        const int N_DIRS, const int DR[], const int DC[]) {
    int s = 0;
    marked[p.r][p.c] = true;
    for (int i = 0; i < N_DIRS; ++i) {
        Point q(p.r + DR[i], p.c + DC[i]);
        if (ok(n,m,q) and map[q.r][q.c] ≠ 'T' and not marked[q.r][q.c])
            s += dfs(n, m, q, map, marked, N_DIRS, DR, DC);
    }
    if ('0' ≤ map[p.r][p.c] and map[p.r][p.c] ≤ '9') {
        s += map[p.r][p.c] - '0';
        map[p.r][p.c] = '.';
    }
    return s;
}

int main(void) {
    int n, m;
    while (cin >> n >> m) {

```

```

vector<Point> knights, bishops ;
vector< vector<char> > map(n, vector<char>(m));
for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j) {
        cin >> map[i][j];
        switch(map[i][j]) {
            case 'K': knights.push_back(Point(i,j)); break;
            case 'B': bishops.push_back(Point(i,j)); break;
        };
    }
int s = 0;
vector< vector<bool> > marked_knight(n, vector<bool>(m, false));
for (int k = 0; k < knights.size (); ++k) {
    Point p = knights[k];
    if (not marked_knight[p.r][p.c])
        s += dfs(n, m, p, map, marked_knight, N_DIRS_KNIGHT, DR_KNIGHT, DC_KNIGHT);
}
vector< vector<bool> > marked_bishop(n, vector<bool>(m, false));
for (int k = 0; k < bishops.size (); ++k) {
    Point p = bishops[k];
    if (not marked_bishop[p.r][p.c])
        s += dfs(n, m, p, map, marked_bishop, N_DIRS_BISHOP, DR_BISHOP, DC_BISHOP);
}
cout << s << endl;
}
}

```

Solution to problem 2

```

#include <iostream>
#include <set>

using namespace std;

typedef long long int lint ;

int main(void) {

    lint suma = 0;
    set<lint> selec ;
    set<lint> resta ;

    int n;
    cin >> n;
    string op;
    lint val;
    while (cin >> op >> val) {
        if (op == "deixar") {

```

```

    if ( selec . size () < n ) {
        selec . insert ( val );
        suma += val;
    }
    else {
        lint min = *( selec . begin () );
        if ( min < val ) {
            selec . insert ( val );
            selec . erase ( selec . begin () );
            suma = suma + val - min;
            resta . insert ( min );
        }
        else resta . insert ( val );
    }
}
else {
    if ( *( selec . begin () ) ≤ val ) {
        selec . erase ( val );
        suma -= val;
        if ( resta . size () > 0 ) {
            set<lint>:: iterator it = resta . end ();
            --it;
            selec . insert ( * it );
            suma += *it;
            resta . erase ( it );
        }
    }
    else resta . erase ( val );
}
cout << suma << endl;
}
}

```


Solution to Lab Exam EDA

13/12/2011

Solution to problem 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> VC;
typedef vector<VC> MC;
typedef vector<bool> VB;
typedef vector<VB> MB;

const int N_DIRS = 4;
const int DI[N_DIRS] = { 1, 0, -1, 0};
const int DJ[N_DIRS] = { 0, 1, 0, -1};

bool ok(int n, int m, int i, int j) {
    return
        0 ≤ i and i < n and
        0 ≤ j and j < m;
}

int search(const MC& map, int n, int m, int i, int j, MB& marked) {
    int t = 0;
    if (map[i][j] == 't') ++t;
    marked[i][j] = true;
    for (int k = 0; k < N_DIRS; ++k) {
        int ii = i + DI[k];
        int jj = j + DJ[k];
        if (ok(n, m, ii, jj) and map[ii][jj] ≠ 'X' and not marked[ii][jj])
            t += search(map, n, m, ii, jj, marked);
    }
    return t;
}

int main(void) {
    int n, m;
    cin >> n >> m;
    MC map(n, VC(m));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> map[i][j];
    int f, c;
    cin >> f >> c;

    MB marked(n, VB(m, false));

```

```
    cout << search(map, n, m, f-1, c-1, marked) << endl;
}
```

Solution to problem 2

```
#include <iostream>
#include <vector>

using namespace std;

void b(int k, int n, string& s) {
    if (k == n) cout << s << endl;
    else {
        s[k] = 'A'; b(k+1, n, s);
        s[k] = 'C'; b(k+1, n, s);
        s[k] = 'G'; b(k+1, n, s);
        s[k] = 'T'; b(k+1, n, s);
    }
}

int main() {
    int n;
    cin >> n;
    string s(n, 'X');
    b(0, n, s);
}
```

Solution to Lab Exam EDA

14/05/2012

Solution to problem 1

```
#include<iostream>
#include<string>
#include<map>

using namespace std;

typedef map<string,string>::iterator ite ;

int main() {
    string s;
    map<string,string> liats;
    while(cin >> s) {
        if (s == "info") {
            cout << "PARELLES:" << endl;
            for (ite it = liats.begin(); it != liats.end(); ++it)
                if (it->second != "" and it->first < it->second)
                    cout << it->first << " " << it->second << endl;
            cout << "SOLS:" << endl;
            for (ite it = liats.begin(); it != liats.end(); ++it)
                if (it->second == "")
                    cout << it->first << endl;
            cout << "-----" << endl;
        }
        else {
            string x, y;
            cin >> x >> y;
            if (liats[x] != "") liats[liats[x]] = "";
            if (liats[y] != "") liats[liats[y]] = "";
            liats[x] = y;
            liats[y] = x;
        }
    }
}
```

Solution to problem 2

```
#include<iostream>
#include<vector>
#include<queue>
#include<algorithm>

using namespace std;
```

```

const int xf[8] = {-1,-1,-1,0,0,1,1,1};
const int yf[8] = {-1,0,1,-1,1,-1,0,1};
const int x[4] = {0,0,1,-1};
const int y[4] = {1,-1,0,0};

bool cerca_bolet (vector<vector<char> >& M, pair<int,int> p) {
    queue<pair<int, int> > Q;
    if (M[p.first][p.second] == 'X') return false;
    M[p.first][p.second] = 'X'; Q.push(p);
    while (not Q.empty()) {
        pair<int,int> q = Q.front (); Q.pop();
        for (int i = 0; i < 4; ++i) {
            int u = q.first + x[i];
            int v = q.second + y[i];
            if (M[u][v] == 'B') return true;
            if (M[u][v] != 'X') {
                M[u][v] = 'X';
                Q.push(make_pair(u,v));
            }
        }
    }
    return false;
}

int main() {
    int f, c;
    while (cin >> f >> c) {
        vector<vector<char> > M(f, vector<char> (c));
        pair<int, int> p;
        queue<pair<int,int> > F;
        for (int i = 0; i < f; ++i) {
            for (int j = 0; j < c; ++j) {
                cin >> M[i][j];
                if (M[i][j] == 'P') {p.first = i; p.second = j;}
                if (M[i][j] == 'F') F.push(make_pair(i, j));
            }
        }
        while (not F.empty()) {
            int i = (F.front()).first;
            int j = (F.front()).second;
            F.pop();
            for (int k = 0; k < 8; ++k) M[i+xf[k]][j+yf[k]] = 'X';
        }
        if (cerca_bolet (M,p)) cout << "si" << endl;
        else cout << "no" << endl;
    }
}

```

Solution to Lab Exam EDA

29/11/2012

Solution to problem 1

```
#include <iostream>
#include <sstream>
#include <vector>
#include <queue>

using namespace std;

typedef vector< queue<string> > VQS;

void read_queues(int n, VQS& v) {
    v = VQS(n);
    for (int k = 0; k < n; ++k) {
        string line;
        getline (cin, line);
        istringstream in (line);
        string name;
        while (in >> name) v[k].push(name);
    }
}

void process_exits (VQS& v) {
    cout << "SORTIDES" << endl;
    cout << "-----" << endl;

    string op;
    while (cin >> op) {
        if (op == "SURT") {
            int idx;
            cin >> idx;
            --idx;
            if (0 ≤ idx and idx < v.size () and not v[idx].empty()) {
                cout << v[idx].front () << endl;
                v[idx].pop ();
            }
        }
        else {
            string name;
            int idx;
            cin >> name >> idx;
            --idx;
            if (0 ≤ idx and idx < v.size ())
                v[idx].push(name);
        }
    }
}
```

```

    cout << endl;
}

void write_final_contents (VQS& v) {
    cout << "CONTINGUTS FINALS" << endl;
    cout << "-----" << endl;
    for (int k = 0; k < v.size (); ++k) {
        cout << "cua " << k+1 << ":";
        while (not v[k].empty()) {
            cout << ' ' << v[k].front ();
            v[k].pop ();
        }
        cout << endl;
    }
}

int main() {
    int n;
    cin >> n;
    string line;
    getline (cin, line); // Read empty line.

    VQS v;
    read_queues(n, v);
    process_exits (v);
    write_final_contents (v);
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

void compute_topological_ordering (const VVI& g, VI& indeg, VI& ord) {
    int n = g.size ();
    priority_queue <int, vector<int>, greater<int> > pq;
    for (int u = 0; u < n; ++u)
        if (indeg[u] == 0) pq.push(u);

    ord = VI(n);
    int cnt = 0;
    while (not pq.empty()) {

```

```

    int u = pq.top ();
    pq.pop ();
    ord[cnt] = u;
    ++cnt;
    for (int k = 0; k < g[u].size (); ++k) {
        int v = g[u][k];
        --indeg[v];
        if (indeg[v] == 0) pq.push(v);
    }
}

void write(const VI& v) {
    cout << v[0];
    for (int k = 1; k < v.size (); ++k)
        cout << ' ' << v[k];
    cout << endl;
}

int main() {
    int n, m;
    while (cin >> n >> m) {
        VVI g(n);
        VI indeg(n, 0);
        for (int k = 0; k < m; ++k) {
            int u, v;
            cin >> u >> v;
            g[u].push_back(v);
            ++indeg[v];
        }
        VI ord;
        compute_topological_ordering (g, indeg, ord);
        write(ord);
    }
}

```

Solution to Lab Exam EDA

22/5/2013

Solution to problem 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> VI;

int n, m;
VI div;

bool ok(int y) {
    for (int i = 0; i < m; ++i)
        if (y % div[i] == 0)
            return false;
    return true;
}

void backtracking (int k, int x) {
    if (k == n) cout << x << endl;
    else {
        for (int v = 0; v ≤ 9; ++v) {
            int y = 10*x + v;
            if (ok(y)) backtracking (k+1, y);
        }
    }
}

int main() {
    while (cin >> n >> m) {
        div = VI(m);
        for (int i = 0; i < m; ++i)
            cin >> div[i];
        backtracking (0, 0);
        cout << "-----" << endl;
    }
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>
#include <queue>

```



```
using namespace std;
```

```
typedef vector<char> VC; typedef vector<int> VI;
typedef vector< VC > VVC; typedef vector< VI > VVI;
```

```
typedef pair<int,int> P;
```

```
const int DF[] = {1, 1,-1,-1, 2, 2,-2,-2};
const int DC[] = {2,-2, 2,-2, 1,-1, 1,-1};
```

```
const int oo = 200 * 200;
```

```
bool ok(int i, int j, int n, int m, const VVC& t) {
    return i ≥ 0 and i < n and j ≥ 0 and j < m and t[i][j] ≠ 'X';
}
```

```
int distance(int f0, int c0, const VVC& t) {
    int n = t.size();
    int m = t[0].size();
    VVI d(n, VI(m, +oo));
    queue<P> q;
    q.push(P(f0, c0));
    d[f0][c0] = 0;
    while (not q.empty()) {
        P p = q.front();
        q.pop();
        int f = p.first;
        int c = p.second;
        if (t[f][c] == 'p') return d[f][c];
        for (int k = 0; k < 8; ++k) {
            int i = f + DF[k];
            int j = c + DC[k];
            if (ok(i, j, n, m, t) and d[i][j] == +oo) {
                q.push(P(i, j));
                d[i][j] = d[f][c] + 1;
            }
        }
    }
    return +oo;
}
```

```
int main() {
    int n, m;
    while (cin >> n >> m) {
        VVC t(n, VC(m));

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < m; ++j)
                cin >> t[i][j];
    }
}
```

```
    int f0, c0;  
    cin >> f0 >> c0;  
    int dist = distance(f0-1, c0-1, t);  
    if (dist == +oo) cout << "no" << endl;  
    else             cout << dist << endl;  
  }  
}
```

Solution to Lab Exam EDA

4/12/2013

Solution to problem 1

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const int UNDEF = -1;
const int N_DIRS = 4;
const int DI[N_DIRS] = { 1, 0, -1, 0};
const int DJ[N_DIRS] = { 0, 1, 0, -1};

struct Pos {
    int i, j;
    Pos(int ii, int jj) : i(ii), j(jj) { }
};

bool ok(int n, int m, int i, int j) {
    return
        0 ≤ i and i < n and
        0 ≤ j and j < m;
}

int bfs(const vector < vector<char> >& map, int i0, int j0) {
    int n = map.size();
    int m = map[0].size();
    queue<Pos> q;
    q.push( Pos(i0, j0) );
    vector< vector<int> > dist(n, vector<int>(m, UNDEF));
    dist[i0][j0] = 0;
    while (not q.empty()) {
        Pos p = q.front();
        q.pop();
        int i = p.i;
        int j = p.j;
        if (map[i][j] == 't') return dist[i][j];
        else {
            for(int k = 0; k < N_DIRS; ++k) {
                int ii = i + DI[k];
                int jj = j + DJ[k];
                if (ok(n, m, ii, jj) and map[ii][jj] ≠ 'X' and dist[ii][jj] == UNDEF) {
                    q.push( Pos(ii, jj) );
                    dist[ii][jj] = 1 + dist[i][j];
                }
            }
        }
    }
}

```

```

    }
}
return UNDEF;
}

int main(void) {

    int n, m;
    cin >> n >> m;

    vector < vector<char> > map(n, vector<char>(m));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> map[i][j];

    int f, c;
    cin >> f >> c;

    int dist = bfs(map, f-1, c-1);
    if (dist > 0) cout << "distancia minima: " << dist << endl;
    else        cout << "no es pot arribar a cap tresor" << endl;
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>
#include <map>
#include <assert.h>

using namespace std;

typedef map<string,int> MSI;
typedef vector<string> VS;
typedef vector<bool> VB;
typedef vector<int> VI;
typedef vector<VI> VVI;

int n_outputs, n_inputs;
VVI gdir, ginv;
MSI s2v;
VS v2s;

int string2vertex (const string& s) {
    auto i = s2v.find(s);
    if (i != s2v.end())
        return i->second;
}

```

```
    else {
        int v = v2s.size ();
        v2s.push_back(s);
        s2v.insert (make_pair(s, v));
        return v;
    }
}

int main() {

    n_outputs = n_inputs = 0;

    string token;

    cin >> token;
    assert (token == "OUTPUT");
    while (cin >> token and token != "END") {
        ++n_outputs;
        string2vertex (token );
    }

    cin >> token;
    assert (token == "INPUT");
    while (cin >> token and token != "END") {
        ++n_inputs;
        string2vertex (token );
    }

    while (cin >> token and token != "END") {

        string s;
        cin >> s;
        int ov = string2vertex (s);
        if (ov+1 > gdir.size ()) gdir.resize (ov+1);

        cin >> s;
        int iv1 = string2vertex (s);
        if (iv1 + 1 > ginv.size ()) ginv.resize (iv1 + 1);

        if (token == "NOT") {
            gdir[ov ].push_back(iv1);
            ginv[iv1 ].push_back(ov );
        }
        else {
            cin >> s;
            int iv2 = string2vertex (s);
            if (iv2 + 1 > ginv.size ()) ginv.resize (iv2 + 1);
            if (token == "AND") {
                gdir[ov ].push_back( min(iv1,iv2) );
            }
        }
    }
}
```

```

        gdir[ov].push_back( max(iv1,iv2) );
    }
    else {
        gdir[ov].push_back( max(iv1,iv2) );
        gdir[ov].push_back( min(iv1,iv2) );
    }
    ginv[iv1].push_back(ov);
    ginv[iv2].push_back(ov);
}
}

int n = gdir.size ();
VI ddir(n, 0);
for (int v = 0; v < n; ++v)
    ddir[v] = gdir[v].size ();

VI bag;
for (int v = n_outputs; v < n_inputs + n_outputs; ++v)
    bag.push_back(v);

VI ord;
while (not bag.empty()) {
    int v = bag.back ();
    ord.push_back(v);
    bag.pop_back ();
    for (auto w : ginv[v]) {
        --ddir[w];
        if (ddir[w] == 0)
            bag.push_back(w);
    }
}

VB val(n);
while (cin >> token) {
    val[ n_outputs ] = (token == "T");
    for (int v = n_outputs+1; v < n_inputs + n_outputs; ++v) {
        cin >> token;
        val[v] = (token == "T");
    }

    for (auto v : ord) {
        if (gdir[v].size () == 1) {
            val[v] = not val[gdir[v][0]];
        }
        else if (gdir[v].size () == 2) {
            int iv1 = gdir[v][0];
            int iv2 = gdir[v][1];
            if (iv1 < iv2) val[v] = val[iv1] and val[iv2];
            else val[v] = val[iv1] or val[iv2];
        }
    }
}

```

```
    }  
  }  
  cout << (val[0] ? 'T' : 'F');  
  for (int v = 1; v < n_outputs; ++v)  
    cout << ' ' << (val[v] ? 'T' : 'F');  
  cout << endl;  
}
```

Solution to Lab Exam EDA

19/5/2014

Solution to problem 1

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<bool> VB;
typedef vector<int> VI;
typedef vector<VI> VVI;

void path(int xi, int xf, const VI& p) {
    if (xf == xi) cout << xi;
    else {
        path(xi, p[xf], p);
        cout << " " << xf;
    }
}

void bfs(const VVI& g, int xi, int xf) {
    int n = g.size ();
    VI p(n, -1);
    VB mkd(n, false);
    VI cur, pos;

    cur.push_back(xi);
    mkd[xi] = true;
    while (not cur.empty()) {
        for (int x : cur) {
            if (x == xf) {
                path(xi, xf, p);
                cout << endl;
                return;
            }
            for (int y : g[x])
                if (not mkd[y]) {
                    pos.push_back(y);
                    mkd[y] = true;
                    p[y] = x;
                }
        }
        swap(pos, cur);
    }
}

```



```

int main() {
    int n, m;
    while (cin >> n >> m) {
        VVI g(n);
        for (int k = 0; k < m; ++k) {
            int x, y;
            cin >> x >> y;
            g[x].push_back(y);
        }
        for (int x = 0; x < n; ++x)
            sort(g[x].begin(), g[x].end());

        bfs(g, 0, n-1);
    }
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<bool> VB;

void bt(int k, int n, string& w, VB& mkd) {
    if (k == n) cout << w << endl;
    else {
        for (int i = 0; i < n; ++i)
            if (not mkd[i] and (k == 0 or 'a' + i != w[k-1] + 1)) {
                mkd[i] = true;
                w[k] = 'a' + i;
                bt(k+1, n, w, mkd);
                mkd[i] = false;
            }
    }
}

int main() {
    int n;
    cin >> n;
    string w(n, 'a');
    VB mkd(n, false);
    bt(0, n, w, mkd);
}

```

Solution to Lab Exam EDA

22/12/2014

Solution to problem 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<bool> VB;
typedef vector<int> VI;
typedef vector<VI> VVI;

const int UNDEF = -1;

bool cyclic (int x, const VVI& g, VB& mkd, VI& par) {
    if (mkd[x]) return true;
    mkd[x] = true;
    for (int y : g[x])
        if (par[x] != y) {
            par[y] = x;
            if (cyclic (y, g, mkd, par)) return true;
        }
    return false;
}

int nombre_arbres(const VVI& g) {
    int n = g.size ();
    VB mkd(n, false);
    VI par(n, UNDEF);
    int n_arb = 0;
    for (int x = 0; x < n; ++x) {
        if (not mkd[x]) {
            if (cyclic (x, g, mkd, par)) return UNDEF;
            else ++n_arb;
        }
    }
    return n_arb;
}

int main() {
    int n, m;
    while (cin >> n >> m) {
        VVI g(n);
        for (int k = 0; k < m; ++k) {
            int x, y;
            cin >> x >> y;
            g[x].push_back(y);
        }
    }
}

```

```

        g[y].push_back(x);
    }
    int n_arb = nombre_arbres(g);
    if (n_arb == UNDEF) cout << "no" << endl;
    else                cout << n_arb << endl;
}
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> VI;

int bt(int k, const VI& m, int x, int sum_par, int max_und) {
    if (sum_par > x or sum_par + max_und < x) return 0;
    if (k == m.size()) return 1;
    int cnt = 0;
    for (int v = 0; v ≤ 2; ++v)
        cnt += bt(k+1, m, x, sum_par + v*m[k], max_und - 2*m[k]);
    return cnt;
}

int main() {
    int x, n;
    while (cin >> x >> n) {
        VI m(n);
        int s = 0;
        for (int k = 0; k < n; ++k) {
            cin >> m[k];
            s += m[k];
        }
        cout << bt(0, m, x, 0, 2*s) << endl;
    }
}

```

Solution to Lab Exam EDA

25/05/2015

Solution to problem 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> VC;
typedef vector<VC> VVC;

int bt(int i, int j, VVC& sol, int curr) {
    int n = sol.size();
    int m = sol[0].size();
    if (i == n) return curr;
    int next_i, next_j;
    if (j == m-1) {
        next_i = i+1;
        next_j = 0;
    }
    else {
        next_i = i;
        next_j = j+1;
    }

    sol[i][j] = 'L';
    int new_sols = 0;
    if (i >= 2 and sol[i-1][j] == 'O' and sol[i-2][j] == 'L') ++new_sols;
    if (j >= 2 and sol[i][j-1] == 'O' and sol[i][j-2] == 'L') ++new_sols;
    if (i >= 2 and j >= 2 and sol[i-1][j-1] == 'O' and sol[i-2][j-2] == 'L') ++new_sols;
    if (i >= 2 and j+2 < m and sol[i-1][j+1] == 'O' and sol[i-2][j+2] == 'L') ++new_sols;
    int nl = bt(next_i, next_j, sol, curr + new_sols);

    sol[i][j] = 'O';
    int no = bt(next_i, next_j, sol, curr);

    return max(nl, no);
}

int main() {
    int n, m;
    while (cin >> n >> m) {
        VVC sol(n, VC(m));
        cout << bt(0, 0, sol, 0) << endl;
    }
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> VC;
typedef vector<VC> VVC;
typedef vector<bool> VB;
typedef vector<VB> VVB;

bool ok(int i, int j, const VVC& t) {
    int n = t.size();
    int m = t[0].size();
    return i ≥ 0 and i < n and j ≥ 0 and j < m and t[i][j] ≠ 'X';
}

const int di[] = {0, 1, 0, -1};
const int dj[] = {1, 0, -1, 0};

bool possible(int i_ini, int j_ini, int i_fin, int j_fin, const VVC& t, VVB& mkd) {
    if (mkd[i_ini][j_ini]) return false;
    mkd[i_ini][j_ini] = true;
    if (i_ini == i_fin and j_ini == j_fin) return true;
    for (int k = 0; k < 4; ++k) {
        int i = i_ini + di[k];
        int j = j_ini + dj[k];
        if (ok(i, j, t) and possible(i, j, i_fin, j_fin, t, mkd)) return true;
    }
    return false;
}

int main() {
    int n, m;
    while (cin >> n >> m) {
        VVC t(n, VC(m));
        int i_ini, j_ini, i_fin, j_fin;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < m; ++j)
                cin >> t[i][j];

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < m; ++j)
                if (t[i][j] == 'I') {
                    i_ini = i;
                    j_ini = j;
                }
                else if (t[i][j] == 'F') {

```

```

        i_fin = i;
        j_fin = j;
    }
    else if (t[i][j] == 'M') {
        t[i][j] = 'X';
        if (i-1 ≥ 0 and t[i-1][j] ≠ 'M') t[i-1][j] = 'X';
        if (i+1 < n and t[i+1][j] ≠ 'M') t[i+1][j] = 'X';
        if (j-1 ≥ 0 and t[i][j-1] ≠ 'M') t[i][j-1] = 'X';
        if (j+1 < m and t[i][j+1] ≠ 'M') t[i][j+1] = 'X';
    }

    VVB mkd(n, VB(m, false));
    if (possible(i_ini, j_ini, i_fin, j_fin, t, mkd)) cout << "SI" << endl;
    else cout << "NO" << endl;
}
}

```

Solution to Lab Exam EDA

22/12/2015

Solution to problem 1

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const vector<pair<int, int>> DIRS = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };
const int UNDEF = -1;

int dist_2on_tresor_mes_Ilunya (int i0, int j0, vector<vector<char>>& mapa) {
    int max_dist = UNDEF;
    int max_dist2 = UNDEF;
    queue<pair<pair<int, int>, int>> q;
    q.push({{i0, j0}, 0});
    mapa[i0][j0] = 'X';
    while (not q.empty()) {
        int i = q.front (). first . first ;
        int j = q.front (). first .second;
        int d = q.front (). second;
        q.pop ();
        for (auto dir : DIRS) {
            int ii = i + dir . first ;
            int jj = j + dir .second;
            if (mapa[ii][jj] != 'X') {
                if (mapa[ii][jj] == 't') {
                    max_dist2 = max_dist;
                    max_dist = d+1;
                }
                q.push({{ii, jj}, d+1});
                mapa[ii][jj] = 'X';
            }
        }
    }
    return max_dist2;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<char>> mapa(n+2, vector<char>(m+2, 'X'));
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            cin >> mapa[i][j];
}

```

```

int f, c;
cin >> f >> c;

int d2 = dist_2on_tresor_mes_llunya (f, c, mapa);
if (d2 == UNDEF) cout << "no es pot arribar a dos o mes tresors" << endl;
else          cout << "segona distancia maxima: " << d2 << endl;
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>

using namespace std;

void escriu (int f, int c, const vector<int>& col) {
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j)
            if (col[i] == j) cout << "R";
            else          cout << ".";
        cout << endl;
    }
    cout << endl;
}

void torres (int f, int c, int i, vector<int>& col, vector<bool>& marked) {
    if (i == f) escriu (f, c, col);
    else
        for (int j = 0; j < c; ++j)
            if (not marked[j]) {
                col[i] = j;
                marked[j] = true;
                torres (f, c, i+1, col, marked);
                marked[j] = false;
            }
}

int main() {
    int f, c;
    cin >> f >> c;
    vector<int> col(f);
    vector<bool> marked(c, false);
    torres (f, c, 0, col, marked);
}

```


Solution to Lab Exam EDA

19/05/2016

Solution to problem 1

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

typedef pair<int,int> P;

int main(void) {
    int n, m;
    while (cin >> n >> m) {
        vector<vector<P>> > g(n);
        for (int k = 0; k < m; ++k) {
            int x, y, c;
            cin >> x >> y >> c;
            --x;
            --y;
            g[x].push_back(P(c, y));
            g[y].push_back(P(c, x));
        }
        vector<bool> mkd(n, false);
        mkd[0] = true;
        priority_queue<P, vector<P>, greater<P>> > pq;
        for (P x : g[0]) pq.push(x);
        int sz = 1;
        int sum = 0;
        while (sz < n) {
            int c = pq.top().first;
            int x = pq.top().second;
            pq.pop();
            if (not mkd[x]) {
                mkd[x] = true;
                for (P y : g[x]) pq.push(y);
                sum += c;
                ++sz;
            }
        }
        cout << sum << endl;
    }
}

```

Solution to problem 2

```
#include <vector>

using namespace std;

// Pre:  $l \leq r$ ,  $x < v[r]$ .
// Return the smallest  $i$  s.t.  $l \leq i \leq r$  and  $x < v[i]$ .
int rightmost(double x, const vector<double>& v, int l, int r) {
    if (l == r) return l;
    int m = (l+r)/2;
    if (x < v[m]) return rightmost(x, v, l, m);
    else return rightmost(x, v, m+1, r);
}

int rightmost(double x, const vector<double>& v) {
    return rightmost(x, v, 0, v.size ());
}
```

Solution to Lab Exam EDA

15/12/2016

Shift 1

Solution to problem 1

```

#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>>> su;
vector<vector<bool>>> r_mkd, c_mkd;
vector<vector<vector<bool>>>> s_mkd;

void write() {
    cout << endl;
    for (int i = 0; i < 9; ++i) {
        cout << su[i][0] + 1;
        for (int j = 1; j < 9; ++j)
            cout << ' ' << su[i][j] + 1;
        cout << endl;
    }
}

bool fill (int i, int j) {
    if (i == 9) return true;
    if (j == 9) return fill (i+1, 0);
    if (su[i][j] != -1) return fill (i, j+1);
    for (int v = 0; v < 9; ++v)
        if (not r_mkd[i][v] and not c_mkd[j][v] and not s_mkd[i/3][j/3][v]) {
            r_mkd[i][v] = c_mkd[j][v] = s_mkd[i/3][j/3][v] = true;
            su[i][j] = v;
            if (fill (i, j+1)) return true;
            r_mkd[i][v] = c_mkd[j][v] = s_mkd[i/3][j/3][v] = false;
        }
    su[i][j] = -1;
    return false;
}

int main() {
    su = vector<vector<int>>>(9, vector<int>(9));
    int n;
    cin >> n;
    cout << n << endl;
    while (n-- > 0) {
        r_mkd = c_mkd = vector<vector<bool>>>(9, vector<bool>(9, false));
        s_mkd = vector<vector<vector<bool>>>>(3, vector<vector<bool>>>(3, vector<bool>(9, false)));
        for (int i = 0; i < 9; ++i)

```

```

    for (int j = 0; j < 9; ++j) {
        char c;
        cin >> c;
        if (c == '.') su[i][j] = -1;
        else {
            int v = c - '1';
            r_mkd[i][v] = c_mkd[j][v] = s_mkd[i/3][j/3][v] = true;
            su[i][j] = v;
        }
    }
    if (fill(0, 0)) write();
}
}

```

Solution to problem 2

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool search(const vector<int>& a, int l, int r, int x) {
    if (l+1 == r) return a[l] == x or a[r] == x;
    int m = (l+r)/2;
    if (a[m] ≥ a[l]) {
        if (a[l] ≤ x and x ≤ a[m]) {
            return binary_search(a.begin()+l, a.begin()+m+1, x);
        }
        else return search(a, m, r, x);
    }
    else {
        if (a[m] ≤ x and x ≤ a[r]) {
            return binary_search(a.begin()+m, a.begin()+r+1, x);
        }
        else return search(a, l, m, x);
    }
}

bool search(int x, const vector<int>& a) {
    return search(a, 0, a.size()-1, x);
}

```

Shift 2**Solution to problem 1**

```
#include <iostream>
#include <vector>

using namespace std;

int m, n;
vector<int> l;

void gen(int i, int pos) {
    if (pos ≥ -m/2 and pos ≤ m/2) {
        if (i == n) cout << pos << endl;
        else {
            gen(i+1, pos+l[i]);
            gen(i+1, pos-l[i]);
        }
    }
}

int main() {
    cin >> m >> n;
    l = vector<int>(n);
    for (int& x : l) cin >> x;
    gen(0, 0);
}
```

Solution to problem 2

```
#include <iostream>

using namespace std;

typedef int Matrix [2][2];

void product(int m, const Matrix& a, const Matrix& b, Matrix& c) {
    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 2; ++j) {
            c[i][j] = 0;
            for (int k = 0; k < 2; ++k)
                c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % m;
        }
}

void identity (Matrix& b) {
```

```
b [0][0] = b [1][1] = 1;
b [0][1] = b [1][0] = 0;
}
```

```
void power(int n, int m, const Matrix& a, Matrix& b) {
```

```
    if (n == 0)
        identity (b);
    else {
        if (n%2 == 0) {
            Matrix c;
            power(n/2, m, a, c);
            product(m, c, c, b);
        }
        else {
            Matrix c, d;
            power(n/2, m, a, c);
            product(m, c, c, d);
            product(m, a, d, b);
        }
    }
}
```

```
int main() {
    Matrix a;
    a [0][0] = 1;      a [0][1] = 1;
    a [1][0] = 1;      a [1][1] = 0;
    int n, m;
    while (cin >> n >> m) {
        Matrix b;
        power(n, m, a, b);
        cout << b[1][0] << endl;
    }
}
```

6

Solutions to Final Exams

Solution of the Final EDA Exam**07/01/2016****Proposed solution to problem 1**

- (a) The master theorem for dividing recurrences claims that if we have a recurrence of the form $T(n) = aT(n/b) + \Theta(n^k)$ with $a > 0$, $b > 1$ and $k \geq 0$, then, letting $\alpha = \log_b a$,

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } \alpha < k, \\ \Theta(n^k \log n) & \text{if } \alpha = k, \\ \Theta(n^\alpha) & \text{if } \alpha > k. \end{cases}$$

- (b) 2

- (c) *priority_queue* <T>'s are implemented with max-heaps. If there existed a function $T\& \text{top}()$;

then a user of the class could modify the root of the heap and break the invariant that each node is greater than or equal to its children.

Proposed solution to problem 2

- (a) Yes
 (b) No (leaves must be black)
 (c) No (the children of a red node must be black)
 (d) Yes
 (e) No (it must be a binary search tree)
 (f) No (the number of black nodes in a path from the root to a leaf must be constant)

Proposed solution to problem 3

- (a) Let $C(n)$ be the cost of the algorithm in the worst case as a function of n . Then:

- If $s \geq \frac{1}{2}$: $C(n) = C(sn) + \Theta(1)$
- If $s \leq \frac{1}{2}$: $C(n) = C((1-s)n) + \Theta(1)$

- (b) The worst case takes place, for instance:

- if $s \geq \frac{1}{2}$, when $x < v[l]$.
- if $s \leq \frac{1}{2}$, when $x > v[r]$.

- (c) We can solve the recurrences separately, or notice that

$$C(n) = C(\max(s, 1-s)n) + \Theta(1) = C\left(\frac{n}{\frac{1}{\max(s, 1-s)}}\right) + \Theta(1)$$

As $0 < s < 1$, we have $0 < \max(s, 1-s) < 1$, and so $\frac{1}{\max(s, 1-s)} > 1$. By applying the master theorem for dividing recurrences we have that $\alpha = k = 0$, and therefore the cost is $\Theta(\log n)$ (independently of s).

Proposed solution to problem 4

(a) A possible solution:

```

#include <limits>

const int oo = numeric_limits<int>::max();

int minimum_cost(const vector<vector<pair<int,int>>>& G, int x, int y) {
    vector<int> cost(G.size(), +oo);
    cost[x] = 0;
    deque<int> dq;
    dq.push_back(x);
    while (not dq.empty()) {
        int u = dq.front();
        dq.pop_front();
        if (u == y) return cost[y];
        for (auto p : G[u]) {
            int v = p.first;
            int w = p.second;
            if (cost[v] > cost[u] + w) {
                cost[v] = cost[u] + w;
                if (w == 0) dq.push_front(v);
                else      dq.push_back(v);
            }
        }
    }
    return -1;
}

```

(b) There is a cost that is proportional to the number of vertices (initialization of the vector of costs). But the cost of the loop is proportional to the number of edges that are visited (being thus the worst case when all edges are visited). In total the cost is $\Theta(|V| + |E|)$.

(c) $\Theta((|V| + |E|) \log |V|)$

Proposed solution to problem 4

(a) A possible solution:

```

bool ok(const vector<int>& s, const set<pair<int,int>>& D) {
    for (auto d : D)
        if (s[d.first] == s[d.second])
            return false;
    return true;
}

bool has_solution(int k, vector<int>& s, const inp_DISEQUALITIES& e) {
    if (k == e.n) return ok(s, e.D);
    for (int v = e.l; v ≤ e.u; ++v) {
        s[k] = v;
        if (has_solution(k+1, s, e)) return true;
    }
    return false;
}

```

```
}

```

```
bool has_solution (const inp_DISEQUALITIES& e) {
    vector<int> s(e.n);
    return has_solution (0, s, e);
}
```

- (b) When there is no solution, the algorithm considers each of the $(u - l + 1)^n$ possible assignments of the n variables to the values in $[l, u]$. For each of these assignments, work is performed with cost $\Omega(1)$. Hence, the cost in the worst case is $\Omega((u - l + 1)^n)$.
- (c) A possible solution:

```
inp_DISEQUALITIES reduction(const inp_COLORING& ec) {
    inp_DISEQUALITIES ed;
    ed.l = 1;
    ed.u = ec.c;
    ed.n = ec.G.size ();
    for (int u = 0; u < ec.G.size (); ++u)
        for (int v : ec.G[u])
            if (u < v)
                ed.D.insert ({u, v});
    return ed;
}
```

- (d) No. By the previous exercise we have that, as COLORING is NP-hard, then so is DISEQUALITIES. Thus, if there were a polynomial algorithm for solving DISEQUALITIES, we would have $P = NP$, and we would have solved a long-standing open problem in theoretical computer science.

Solution of the Final EDA Exam

12/01/2017

Proposed solution to problem 1

- (a) (0.25 pts.) A graph with n vertices has $O(n^2)$ edges.
- (b) (0.25 pts.) A connected graph with n vertices has $\Omega(n)$ edges.
- (c) (0.25 pts.) A complete graph with n vertices has $\Omega(n^2)$ edges.
- (d) (0.25 pts.) A min-heap with n vertices has $\Theta(n)$ leaves.
- (e) (0.25 pts.) A binary search tree with n vertices has height $\Omega(\log n)$.
- (f) (0.25 pts.) A binary search tree with n vertices has height $O(n)$.
- (g) (0.25 pts.) An AVL tree with n vertices has height $\Omega(\log n)$.
- (h) (0.25 pts.) An AVL tree with n vertices has height $O(\log n)$.

Proposed solution to problem 2

- (a) Breadth-first search.
- (b) Dijkstra's algorithm.
- (c) Bellman-Ford's algorithm.
- (d) There cannot be any cycle with negative weight in the graph.
- (e) By induction over the number of arcs of the path.
 - **Base case:** If the path has not arc, the source vertex u is the same as the target vertex v . So in this case we have that $\omega_\pi(c) = \omega(c) = 0$. Since $\omega(c) - \pi(u) + \pi(v) = 0$, what we wanted to prove holds.
 - **Inductive case:** Assume that the path has k arcs, that is, is of the form (u_0, u_1, \dots, u_k) , where $u_0 = u$ and $u_k = v$. As u_1, \dots, u_k is a path from u_1 to u_k with $k - 1$ arcs, we can apply the induction hypothesis. So $\omega_\pi(u_1, \dots, u_k) = \omega(u_1, \dots, u_k) - \pi(u_1) + \pi(u_k)$. But

$$\begin{aligned}
 \omega_\pi(u_0, \dots, u_k) &= \omega_\pi(u_0, u_1) + \omega_\pi(u_1, \dots, u_k) \\
 &= \omega_\pi(u_0, u_1) + \omega(u_1, \dots, u_k) - \pi(u_1) + \pi(u_k) \\
 &= \omega(u_0, u_1) - \pi(u_0) + \pi(u_1) + \omega(u_1, \dots, u_k) - \pi(u_1) + \pi(u_k) \\
 &= \omega(u_0, u_1) - \pi(u_0) + \omega(u_1, \dots, u_k) + \pi(u_k) \\
 &= \omega(u_0, \dots, u_k) - \pi(u_0) + \pi(u_k)
 \end{aligned}$$

- (f) If π is a potential, then the reduced weights ω_π are non-negative. So we can apply Dijkstra's algorithm to compute the distances with weights ω_π from s to all vertices. Then we can compute the distances with weights ω using the following observation: if $u, v \in V$ i c is the minimum path with weights ω from u to v (which exists by hypothesis), then c is the minimum path with weights ω_π from u to v and $\omega(c) = \omega_\pi(c) + \pi(u) - \pi(v)$.

Proposed solution to problem 3

- (a) If M is an $n \times n$ matrix, function *matrix_mystery*(**const matrix&** M) computes $M^{\sum_{i=1}^n i}$, or equivalently, $M^{\frac{n(n+1)}{2}}$.
- (b) The product of two matrices $n \times n$, which is computed by function *aux*, takes $\Theta(n^3)$ time. Since $\Theta(n)$ iterations are performed, and in each of them two matrix products are computed with cost $\Theta(n^3)$, in total the cost is $\Theta(n^4)$.
- (c) A possible solution:

```
matrix exp(const matrix& M, int k) {
    if (k == 1) return M;
    matrix P = exp(M, k/2);
    if (k % 2 == 0) return aux(P, P);
    else return aux(aux(P, P), M);
}
```

```
matrix mystery(const matrix& M) {
    int n = M.size ();
    return exp(M, n*(n+1)/2);
}
```

Fast exponentiation makes $\Theta(\log(n(n+1)/2)) = \Theta(\log(n))$ matrix products, each of which takes $\Theta(n^3)$ time. In total, the cost is $\Theta(n^3 \log n)$.

Proposed solution to problem 4

- (a) The witness is p .
- (b) The code of the verifier is between lines 4 and 16.
- (c) A possible solution:

```
bool ham2_rec(const vector<vector<int>>& G, int k, int u, vector<int>& next) {
    int n = G.size ();
    if (k == n)
        return find(G[u].begin (), G[u].end (), 0) != G[u].end();

    for (int v : G[u])
        if (next[v] == -1) {
            next[u] = v;
            if (ham2_rec(G, k+1, v, next)) return true;
            next[u] = -1;
        }
    return false;
}

bool ham2(const vector<vector<int>>& G) {
    int n = G.size ();
    vector<int> next(n, -1);
    return ham2_rec(G, 1, 0, next);
}
```

(d) One can replace the call

```
return find(G[u].begin(), G[u].end(), 0)  $\neq$  G[u].end();
```

by

```
return not G[u].empty() and G[u][0] == 0;
```

(e) If G is not connected then it cannot be Hamiltonian. In this case the function only needs to return **false**.