DATA STRUCTURES AND ALGORITHMS PROBLEM SET

Albert Atserias Amalia Duch Jordi Petit
Enric Rodríguez-Carbonell
(Editors)

February 6, 2017

Departament de Ciències de la Computació
Universitat Politècnica de Catalunya

Contents

1

Analysis of algorithms

1. **Big sums everywhere**. Prove the following identities:
   (a) $\sum_{i=1}^{n} i = n(n+1)/2$.
   (b) $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$.
   (c) $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$.

2. **Exact time**. Let us suppose that assigments, increments by one unit, and comparisons of integers take 15 μs, 21 μs, and 34 μs, respectively, and that the rest of instructions (increments of the program counter, jumps, etc.) take negligible time. Give the formula that expresses, as a function of the integer input n, the execution time $T(n)$ of the following algorithm:

```
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        ++s;
}   }
```

3. **It takes ages**. For each function $f(n)$ and running time t in the table below, determine the largest input size that can be solved in time t for a problem that requires $f(n)$ microseconds (μs) to solve on an input of size n.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\log n$							
$\sqrt{n}$							
n							
n^2							
n^3							
2^n							

4. **Worst and best cases of selection sort**. Let us suppose that assigments, increments by one unit, comparisons, and accesses to vectors of integers take 15 μs, 21 μs, 34 μs,

and 12 μs, respectively, and that the rest of instructions (increments of the program counter, jumps, etc.) take negligible time. For the program below, give the formula that expresses, as a function of the length n of the input integer vector $A[0, \ldots, n-1]$, the execution time $T_W(n)$ in the worst case according to the content of A (i.e., a content that makes the algorithm take maximum time). Repeat for the execution time $T_B(n)$ in the best case according to the content of A (i.e., a content that makes the algorithm take minimum time).

```
for (int i = 0; i < n; ++i) {
    int k = i;
    for (int j = i; j < n; ++j) {
        if (A[j] < A[k]) k = j;
    }
    int aux = A[i]; A[i] = A[k]; A[k] = aux;
}
```

5. **Polynomials vs. Exponentials**. The goal of this problem is to prove that if $a > 0$ and $b > 1$ are two fixed reals, then $n^a = O(b^n)$ but $n^a \neq \Omega(b^n)$. For fixed reals $r > 0$ and $s > 1$, consider the real function $f(n) = n^r/s^n$ in the interval $(0, +\infty)$.
   (a) Compute the derivative $f'(n)$ of $f(n)$.
   (b) Solve the equation $f'(n) = 0$ in n and prove that it has a unique solution (which from now on we will refer to as $n_{r,s}$).
   (c) Analyze the sign of $f'(n)$ to conclude that $f(n_{r,s})$ is the maximum of $f(n)$.
   (d) Conclude that $n^r = O(s^n)$. Which c and n_0 did you choose for the O?
   (e) Using this prove that, in fact, $\lim_{n\to+\infty} \frac{n^a}{b^n} = 0$ for all reals $a > 0$ and $b > 1$.
   (f) Conclude that $n^a = O(b^n)$ but $n^a \neq \Omega(b^n)$ for all reals $a > 0$ and $b > 1$.

6. **Logarithms vs. Roots**. Use the conclusions of the problem "Polynomials vs. Exponentials" to show that for every pair of reals constants $a, b \geq 1$ it holds that $(\ln n)^a = O(n^{1/b})$ but $(\ln n)^a \neq \Omega(n^{1/b})$.

7. **Polynomials and logarithms**. Prove the following asymptotic identities:
   (a) Every polynomial $p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 n^0$ with $a_k > 0$ is $\Theta(n^k)$.
   (b) For every two bases $a, b > 1$, we have that $\log_a n$ is $\Theta(\log_b n)$.

8. **Growth orders I**. Consider the following functions: $5n \ln n$, $4n\sqrt{n}$, $\log_\pi(n^n)$, $n/\log_2 n$, $n/\ln n$, $3\ln(7^n)$. Sort them, in increasing order of asymptotic growth. If two functions have the same asymptotic growth, point it out.

9. **Growth orders II**. Classify the functions below in such a way that $f(n)$ and $g(n)$ belong to the same class if and only if $f(n) \in \Theta(g(n))$, and enumerate the classes according to their growth order: n, $\sqrt{n}$, $n^{1,5}$, n^2, $n \log n$, $n \log\log n$, $(\log\log n)^2$, $n \log^2 n$, $n^3/(n-1)$, $n^{\log n}$, $n \log(n^2)$, $21n$, 2^n, $2^{\sqrt{n}}$, $2^{\sqrt{\log n}}$, $2^{n/2}$, 37, 2^{2^n}, $2/n$ i $n^2 \log n$.

10. **Match the dozen**. Match each of the following 12 growth orders with its formula: 1) logarithmic, 2) polylogarithmic, 3) radical (or sublinear), 4) linear, 5) quasilinear,

6) quadratic, 7) polynomial, 8) polylogarithmic exponential (or quasipolynomial), 9) radical exponental (or sublinear exponential), 10) linear exponential (or exponential), 11) factorial, 12) polynomial exponential (or quasiexponential).

a) $\Theta(n^2)$,

b) $\Theta(2^{cn})$ for a constant $c > 0$,

c) $\Theta(\log n)$,

d) $\Theta(2^{(\log n)^c})$ for a constant $c \geq 1$,

e) $\Theta(n \log n)$,

f) $\Theta(n^c)$ for a constant $c \geq 1$,

g) $\Theta(2^{n^c})$ for a constant $c \geq 1$,

h) $\Theta(n)$,

i) $\Theta(n!)$,

j) $\Theta(n^c)$ for a constant $0 < c < 1$,

k) $\Theta((\log n)^c)$ for a constant $c \geq 1$,

l) $\Theta(2^{n^c})$ for a constant $0 < c < 1$.

11. **Simplifying expressions**. For each of the functions below, give its order of magnitude in its simplest possible form using asymptotic notation Θ. For example, $3n^2 = \Theta(n^2)$.

(a) $23n^2 + 3n - 4 = \cdots$
(b) $42 = \cdots$
(c) $2n = \cdots$
(d) $3n = \cdots$
(e) $2^n = \cdots$
(f) $3^n = \cdots$
(g) $0.001n \log n + 1000n = \cdots$
(h) $n^2 + n - \sqrt{n} = \cdots$
(i) $n/\log n + 1/n = \cdots$
(j) $n! + 2^n = \cdots$
(k) $n^n + n! = \cdots$

12. **Fors, and fors inside fors**. For each of the following fragments of code, analyze its cost.

```
// Fragment 1
int s = 0;
for (int i = 0; i < n; ++i) {
  ++s;
}

// Fragment 2
int s = 0;
for (int i = 0; i < n; i += 2) {
    ++s;
```

```
}

// Fragment 3
int s = 0;
for (int i = 0; i < n; ++i) {
    ++s;
}
for (int j = 0; j < n; ++j) {
    ++s;
}

// Fragment 4
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        ++s;
}   }

// Fragment 5
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i; ++j) {
        ++s;
}   }

// Fragment 6
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = i; j < n; ++j) {
        ++s;
}   }

// Fragment 7
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int k = 0; k < n; ++k) {
            ++s;
}   }   }

// Fragment 8
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i; ++j) {
        for (int k = 0; k < j; ++k) {
            ++s;
}   }   }

// Fragment 9
```

```
int s = 0;
for (int i = 1; i ≤ n; i *= 2) {
    ++s;
}

// Fragment 10
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i * i; ++j) {
        for (int k = 0; k < n; ++k) {
            ++s;
}   }   }

// Fragment 11
int s = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i * i; ++j) {
        if (j % i == 0) {
            for (int k = 0; k < n; ++k) {
                ++s;
}   }   }   }
```

13. **Parameters**. State the cost of instructions (1), (2) and (3) in the following program.

```
int function1 (vector<int>& v) { return v[0]; }
int function2 (vector<int> v) { return v[0]; }

int f (int n) {
    vector<int> v(n, 33);          // (1)
    int a = function1(v);          // (2)
    int b = function2(v);          // (3)
    return a + b;
}
```

14. **Os, Omegas, or Thetas**. The following sentences refer to the insertion sort algorithm and the X's hide an occurrence of O, Ω, or Θ. For each of the sentences, indicate which options for $X \in \{O, \Omega, \Theta\}$ make the statement true, which options make the statement false, and justify your answers:

    (a) The worst case is $X(n^2)$.

    (b) The worst case is $X(n)$.

    (c) The best case is $X(n^2)$.

    (d) The best case is $X(n)$.

    (e) For every probability distribution, the average case is $X(n^2)$.

    (f) For every probability distribution, the average case is $X(n)$.

    (g) For some probability distribution, the average case is $X(n \log n)$.

15. **Elementary sorting algorithms**. State the running time of the elementary sorting algorithms (selection sort, insertion sort, and bubble sort) when the input is...
    (a) permuted at random,
    (b) sorted,
    (c) sorted in reverse order,
    (d) with all elements the same.

16. **Omer computes medians in his sleep**. Despite feeling sleepy all day long, Omer has written a *correct* program that returns the median of a given table with n elements. You have not seen Omer's algorithm. Nonetheless, you can still assert that only one of the following statements is true. Reason which.
    (a) The cost of Omer's algorithm is necessarily $\Theta(n)$.
    (b) The cost of Omer's algorithm is necessarily $\Omega(n)$.
    (c) The average-case cost of Omer's algorithm is necessarily $O(\log n)$.
    (d) The worst-case cost of Omer's algorithm is necessarily $\Theta(n \log n)$.

17. **Adding up matrices**. Write the straightforward algorithm that adds two matrices $n \times n$. Explain why this algorithm of cost $O(n^2)$ is linear. Argue that any algorithm that adds two matrices $n \times n$ requires $\Omega(n^2)$ steps.

18. **A mystery**. Explain what the following algorithm does and give its cost as a function of n.

```
int mystery(int n) {
    int f = 1;
    for (int i = 2; i ≤ n; ++i) f *= i;
    return f;
}
```

19. **Primality**. The following algorithms are supposed to determine whether the natural number n is a prime number. Say which ones are correct, which ones are not correct, and what their cost is as a function of n.

```
bool is_prime_v1(int n) {
    if (n ≤ 1) return false;
    for (int i = 2; i < n; ++i) if (n%i == 0) return false;
    return true;
}

bool is_prime_v2(int n) {
    if (n ≤ 1) return false;
    for (int i = 2; i*i < n; ++i) if (n%i == 0) return false;
    return true;
}

bool is_prime_v3(int n) {
    if (n ≤ 1) return false;
```

```
    for (int i = 2; i*i ≤ n; ++i) if (n%i == 0) return false;
    return true;
}

bool is_prime_v4(int n) {
    if (n ≤ 1) return false;
    if (n == 2) return true;
    if (n%2 == 0) return false;
    for (int i = 3; i*i ≤ n; i += 2) if (n%i == 0) return false;
    return true;
}
```

20. **Eratostenes' sieve**. The following program implements Eratostenes' sieve to determine, for each number between 0 and n (both included), whether it is a prime number. Analyze its cost.

```
vector<bool> primes(int n) {
    vector<bool> p(n + 1, true);
    p[0] = p[1] = false;
    for (int i = 2; i*i ≤ n; ++i) {
        if (p[i]) {
            for (int j = 2*i; j ≤ n; j += i) p[j] = false;
    }   }
    return p;
}
```

21. **Subtractive**. Solve the following subtractive recurrences:

    (a) $T(n) = T(n-1) + \Theta(1)$,

    (b) $T(n) = T(n-2) + \Theta(1)$,

    (c) $T(n) = T(n-1) + \Theta(n)$,

    (d) $T(n) = 2T(n-1) + \Theta(1)$.

22. **Divisive**. Solve the following divisive recurrences:

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

    (b) $T(n) = 2T(n/2) + \Theta(n)$,

    (c) $T(n) = 2T(n/2) + \Theta(n^2)$,

    (d) $T(n) = 2T(n/2) + O(1)$,

    (e) $T(n) = 2T(n/2) + O(n)$,

    (f) $T(n) = 2T(n/2) + O(n^2)$,

    (g) $T(n) = 4T(n/2) + n$,

    (h) $T(n) = 4T(n/2) + n^2$,

    (i) $T(n) = 4T(n/2) + n^3$,

    (j) $T(n) = 9T(n/3) + 3n + 2$,

(k) $T(n) = T(9n/10) + \Theta(n)$,

(l) $T(n) = 2T(n/4) + \sqrt{n}$,

23. **Fast and slow exponentiation**. Analyze the cost of the following recursive functions that compute x^n for $n \geq 0$.

```
double power_1 (double x, int n) {
    if (n==0) {
        return 1;
    } else {
        return x*power_1(x,n-1);
}   }

double power_2 (double x, int n) {
    if (n==0) {
        return 1;
    } else if (n%2==0) {
        int y = power_2(x,n/2);
        return y*y;
    } else {
        int y = power_2(x,n/2);
        return y*y*x;
}   }

double power_3 (double x, int n) {
    if (n==0) {
        return 1;
    } else if (n%2==0) {
        return power_3(x,n/2) * power_3(x,n/2);
    } else {
        return power_3(x,n/2) * power_3(x,n/2) * x;
}   }
```

24. **Finding x**. The cost of algorithm A is given by the recurrence $T_A(n) = 7T_A(n/2) + n^2$. A competing algorithm B has cost given by $T_B(n) = xT_B(n/4) + n^2$. Which is the largest integer x for which B is asymptotically better than A?

25. **Breaking it into pieces**. Compute the cost of the following algorithm:

```
double F (vector<double>& v, int i, int j) {
    int aux = (j-i+1)/4;
    if (aux>0) {
        int m1 = i-1+aux;
        int m2 = m1+aux;
        int m3 = m2+aux;
        return F(v,i,m1) + F(v,m1+1,m2) + F(v,m2+1,m3) + F(v,m3+1,j);
    } else {
        return i+j-1;
}   }
```

26. **Breaking it into pieces II**. Consider the following two functions:

```
double A (vector<double>& v, int i, int j) {
    if (i < j) {
        int x = f(v, i, j);
        int m = (i + j) / 2;
        return A(v, i, m-1) + A(v, m, j) + A(v, i+1, m) + x;
    } else {
        return v[i];
}   }

double B (vector<double>& v, int i, int j) {
    if (i<j) {
        int x = g(v, i, j);
        int m1 = i + (j - i + 1) / 3;
        int m2 = i + (j - i + 1) * 2 / 3;
        return B(v, i, m1-1) + B(v, m1, m2-1) + B(v, m2, j) + x;
    } else {
        return v[i];
}   }
```

Assuming that the cost of $f()$ is $\Theta(1)$ and that the cost of $g()$ is proportional to the size of the given vector (that is, $\Theta(j - i + 1)$), what are the asymptotic costs of $A()$ and $B()$ as functions of n?

Assuming they compute the same function, which of the two would you choose?

27. **Generic recurrence**. The cost $T(n)$ of a recursive algorithm satisfies $T(n) = xT(n/4) + \Theta(\sqrt{n})$, with $x > 0$. State the cost of $T(n)$ taking into account that there are three possibilities that depend on the interval to which x belongs.

28. **Tricky max and min**. Let T be a table with n elements.

    (a) Write an algorithm that makes exactly $n - 1$ comparisons to find the maximum element in T.

    (b) Write an algorithm that makes exactly $n - 1$ comparisons to find the minimum element in T.

    (c) Write an algorithm that finds both the minimum and the maximum elements in T and makes only $\frac{3}{2}n$ comparisons.

29. **Changing variables**. Solve the recurrence $T(n) = T(\sqrt{n}) + 1$ using a change of variables. Assume that n is of the form 2^{2^i} for an integer $i \geq 0$.

30. **Bisorted bidimensional table**. Consider a bidimensional table of size $n \times n$ where each column has its elements sorted in strict increasing order from top to bottom, and each row has its elements sorted in strict decreasing order from left to right.

    (a) Give an algorithm of cost $\Theta(n)$ that, given an element x, determines if x is in the table or not.

    (b) Give an algorithm of cost $\Theta(n)$ that, given an element x, determines how many elements in the table are strictly smaller than x.

2

Divide and conquer

1. **More rigor, please**. Consider the following problem: given a sorted table T with n elements and a further element x, return -1 if x does not belong to T, or an index i such that $T[i] = x$ if it does.

   Express your opinion about the following three fragments of code (fragment 1 comes from a website of programmers, fragment 2 is an adapted version of the code in the book *Modern software development using Java* by Tymann and Schneider, and fragment 3 is an adapted version of the code from the book *Developing Java software* by Winder and Roberts)

```
// Fragment 1
int lookup1 (vector<int>& T, int x) {
    int l = 0;
    int r = T.size () - 1;
    while (l <= r) {
        int m = (l+r) / 2;
        if (x < T[m]) r = m;
        else if (x > T[m]) l = m + 1;
        else return m;
    }
    return -1;
}

// Fragment 2
int lookup2 (vector<int>& array, int target ) {
    int start = 0;
    int end = array . size ();
    int position = -1;
    while ( start <=end and position==-1) {
        int middle = ( start +end)/2;
        if ( target <array[middle]) end = middle - 1;
```

```
            else if ( target >array[middle]) start = middle + 1;
            else position = middle;
        }
        return position ;
    }

    // Fragment 3
    int lookup2 (vector<int>& v, int o) {
        int hi = v.size ();
        int lo = 0;
        while (true) {
            int centre = (hi+lo)/2;
            if (centre==lo) {
                if (v[centre]==o) return centre ;
                else if (v[centre+1]==o) return centre+1;
                else return -1;
            }
            if (v[centre]<o) lo = centre ;
            else if (o<v[centre]) hi = centre ;
            else return centre ;
    }   }
```

2. **Range**. Write an algorithm of cost $\Theta(\log n)$ that, given a sorted table T with n different elements and two further elements x and y with $x \leq y$, returns the number of elements in T that fall between x and y (both included).

3. **Merge up, merge down**. Sort the table $\langle 3, 8, 15, 7, 12, 6, 5, 4, 3, 7, 1\rangle$ with recursive mergesort and with bottom-up mergesort.

4. **Mergesort**. State the running time of mergesort when the input is...

   (a) sorted at random,

   (b) sorted,

   (c) sorted in reverse order,

   (d) with all elements the same.

5. **Stack depth**. What is the maximum number of recursive calls that need to be stored in the stack when running mergesort on a table with n elements?

6. **Quicksort by hand**. Sort the table $\langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\rangle$ with quicksort and Hoare's partition.

7. **Quicksort**. State the running time of quicksort with Hoare's partition when the input is...

   (a) permuted at random,

   (b) sorted,

   (c) sorted in reverse order,

(d) with all elements the same.

8. **Sorting by two fields**. We are given a vector of vectors $V[0 \,..\, n-1][2]$ containing information of n individuals. Each position $V[i]$ stores the two surnames of the i-th individual (this is the Iberian peninsula): $V[i][0]$ stores the first surname, $V[i][1]$ stores the second surname.

   We want to sort the vector in the usual order (in the Iberian peninsula anyway): The individuals with smallest first surname first. In case of a draw, the individuals with smaller second surname go first. Assume that no two individuals share the same pair of surnames.

   For example, if the initial contents of V were

	0	1	2	3
0	Garcia	Roig	Garcia	Grau
1	Pi	Negre	Cases	Negre

   the final result would have to be

	0	1	2	3
0	Garcia	Garcia	Grau	Roig
1	Cases	Pi	Negre	Negre

   Among the following combinations, one and only one solves the problem in all cases. State which one and what the cost is in terms of running time and memory space:

   (a) First we sort V with quicksort using the first surname, then we sort it with mergesort using the second surname.

   (b) First we sort V with mergesort using the first surname, then we sort it with quicksort using the second surname.

   (c) First we sort V with quicksort using the second surname, then we sort it with mergesort using the first surname.

   (d) First we sort V with mergesort using the second surname, then we sort it with quicksort using the first surname.

9. **Unimodal**. A sequence $A = (a_1, \ldots, a_n)$ of length $n \geq 3$ is called *unimodal* if there exists an index p with $1 \leq p \leq n$ such that $a_1 < a_2 < \cdots < a_p$ and $a_p > a_{p+1} > \cdots > a_n$; in other words, the sequence increases up to a_p and then decreases. The element a_p is called its *top*. Write a divide and conquer algorithm that, given a vector that stores a unimodal sequence, finds its top in cost $O(\log n)$. Argue that the suggested algorithm has cost $O(\log n)$, and prove its correctness.

10. **Quim's wild sorting algorithm**. Let $T[i..j]$ be a table with $n = j - i + 1$ elements. Consider the following sorting algorithm known as *Quim's wild sorting algorithm*:

    (a) If $n \leq 2$, sort the table directly.

    (b) If $n \geq 3$, "divide" the table into three intervals $T[i..k-1]$, $T[k..\ell]$ and $T[\ell+1..j]$, where $k = i + \lfloor n/3 \rfloor$ and $\ell = j - \lfloor n/3 \rfloor$. The algorithm recursively calls itself to first sort $T[i..\ell]$, then sort $T[k..j]$, and finally sort $T[i..\ell]$ again.

    Prove that this algorithm is correct. Give a recurrence for its running time and solve it.

11. **Counting inversions**. Recall that an *inversion* in a table $A[1 \ldots n]$ is a pair of indices that point to elements in inverted order; that is, a pair (i, j) such that $1 \leq i < j \leq n$ and $A[i] > A[j]$. Write an algorithm that counts the number of inversions in a table with n elements. The worst-case cost of the algorithm should be $O(n \log n)$.

12. **One inversion**. Give an answer to the following:

    (a) Prove that if a table has a single inversion, then the two elements of this inversion appear next to each other; that is, if (i, j) is the unique inversion in the table, then $i + 1 = j$.

    (b) Describe an algorithm of cost $\Theta(\log n)$ that, given a table $T[1 \ldots n]$ that has a single inversion, and given an element x, determines whether x is in T of not.

13. **Almost zero subtable**. Suggest an algorithm of cost $O(n \log n)$ that, given a table with n integers, computes a non-empty consecutive subtable whose sum is as close to 0 as possible.

14. **Almost zero subtable II**. Assuming that the table is sorted, find an algorithm of cost $O(n)$ for the problem above.

15. **Searching in two tables**. Suggest an algorithm of cost $\Theta(\log n)$ that, given two tables that are sorted in increasing order, of n elements each, and given an integer k, finds the k-th smallest of all $2n$ elements.

16. **Finding a fixed point**. Let V be a vector of n different integers sorted in increasing order. Suggest an algorithm of cost $O(\log n)$ that returns an index i such that $V[i] = i$, if it exists, and returns -1 otherwise.

17. **Karatsuba's multiplication**. This problem studies a divide and conquer algorithm to multiply two numbers of $n = 2^k$ bits each. Recall that the school algorithm takes $\Theta(n^2)$ steps.

    a) Let x and y be numbers of n bits each and write them as $x = a2^{n/2} + b$ and $y = c2^{n/2} + d$. Check that $xy = ac\ 2^n + (ad + bc)2^{n/2} + bd$.

    Using this idea, suggest a divide and conquer algorithm to reduce the multiplication of two n-bit numbers to four multiplications of $n/2$-bit numbers that are then combined through additions and shifts.

    Analyze the cost of the resulting algorithm.

    b) In 1962, Karatsuba and Ofman discovered a very clever way to decrease the number of multiplications of $n/2$-bit numbers from four to three: Check that $xy = ((a + b)(c + d) - ac - bd)2^{n/2} + ac2^n + bd$.

    Using this idea, suggest a new divide and conquer algorithm and analyze its cost.

    c) How would the above algorithms be generalized when n is not a power of 2?

18. **Strassen's matrix multiplication**. This problem studies a divide and conquer algorithm to multiply two $n \times n$ matrices. Recall that the classical algorithm requires $\Theta(n^3)$.

According to the old saying "row by column", the product of two 2×2 matrices takes 8 real products:

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} a_{00} * b_{00} + a_{01} * b_{10} & a_{00} * b_{01} + a_{01} * b_{11} \\ a_{10} * b_{00} + a_{11} * b_{10} & a_{10} * b_{01} + a_{11} * b_{11} \end{pmatrix}.$$

Around the year 1967, Strassen discovered that this matrix product can be done with 7 real products. To do it, he defined

$$\begin{aligned} m_1 &= (a_{00} + a_{11}) * (b_{00} + b_{11}) \\ m_2 &= (a_{10} + a_{11}) * (b_{00}) \\ m_3 &= (a_{00}) * (b_{01} - b_{11}) \\ m_4 &= (a_{11}) * (b_{10} - b_{00}) \\ m_5 &= (a_{00} + a_{01}) * (b_{11}) \\ m_6 &= (a_{10} - a_{00}) * (b_{00} + b_{01}) \\ m_7 &= (a_{01} - a_{11}) * (b_{10} + b_{11}) \end{aligned}$$

and checked that

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}.$$

The point of this fact is that, using this idea recursively, two big matrices can be multiplied in less than $\Theta(n^3)$ steps: Assume we are given two $n \times n$ matrices A and B, where n is a power of two. Their product can be decomposed into four blocks as follows:

$$\left(\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array}\right) = \left(\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array}\right) * \left(\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array}\right).$$

It is not hard to see that these blocks can be thought of as numbers. For example, the $\frac{n}{2} \times \frac{n}{2}$ matrix C_{00} corresponds to $M_1 + M_4 - M_5 + M_7$, where the M_i come from Strassen's formulas, replacing numbers by matrices.

a) Assuming that n is a power of 2, how many steps does this algorithm take to multiply two $n \times n$ matrices?

b) What simple modification would you apply to the algorithm to multiply two $n \times n$ matrices when n is not a power of 2? What would the cost of the resulting algorithm be?

c) Give a lower bound to the problem of multiplying two $n \times n$ matrices.

19. **Maximal subsequence**. The maximum subsequence problem is the following: given a table A with n integers, to determine the maximum sum that can be found in a consecutive subtable of the input. For example, if the table t has the following 10 elements

$$31, -41, 59, 26, -53, 58, 97, -93, -23, 84$$

then the maximum subsequence sum is 187, corresponding to the sum of the elements $A[2 \,..\, 6]$. Formally, we want to compute

$$\max \left\{ \sum_{k=i}^{j} A[k] \;:\; 0 \le i < n,\ 0 \le j < n \right\}.$$

Note that the empty subtable, which has sum 0, is valid.

(a) Write an algorithm of cost $\Theta(n \log n)$ that solves this problem by divide and conquer.

(b) Write an algorithm of cost $\Theta(n)$ that solves the same problem.

Note: there exist linear algorithms for this problem which are not based on divide and conquer.

20. **Championship timetable**. We need to organize the timetable of a championship between n players, each of whom must confront each adversary exactly once. In addition, each player must play exactly one game each day. Assuming that n is a power of 2, suggest and implement an algorithm that builds the timetable and that completes the championship after $n-1$ days. Analyze the cost of your algorithm.

21. **Quicksort in the average case**. The goal of this problem is to show that the number of comparisons of quicksort, in the average case when the input $A[1,\ldots,n]$ is a permutation of the set $\{1,\ldots,n\}$ chosen uniformly at random, is $O(n \log n)$. Let us consider the version[1] of the algorithm that, on input $A[l,\ldots,r]$ with $|r-l+1| \geq 3$, takes the first element $q := A[l]$ as pivot and makes recursive calls with $A[l,\ldots,q-1]$ and $A[q+1,\ldots,r]$ after having executed the partition algorithm that does the following: compares $A[l]$ with every other element in $A[l,\ldots,r]$ and places the elements of $A[l,\ldots,r]$ that are smaller than q to the left of $A[q]$ (but in the same relative order they were), places the elements of $A[l,\ldots,r]$ that are bigger than q to the right of $A[q]$ (but in the same relative order they were), and places q in $A[q]$.

(a) Argue that, before any recursive call, the probability distribution on the input $A[l,\ldots,r]$ is uniform over the permutations of $\{l,\ldots,r\}$.

(b) Prove that if $T(n)$ is the expected number of comparisons of the algorithm on a subvector of length $n \geq 3$, then $T(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} T(i)$.

(c) Prove that $\sum_{i=1}^{n-1} i \ln(i) \leq \int_1^n x \ln(x) dx = \frac{1}{2}n^2 \ln(n) - \frac{1}{4}n^2 + \frac{1}{4}$ for every $n \geq 1$.

(d) Make use of the previous two points to show that $T(n) \leq 2n \ln(n)$ by induction.

[1] This version uses a partition algorithm which, unlike Hoare's partition seen in class, cannot be implemented in-place; we could have written this problem for Hoare's partition, but then the first part of the problem would have been significantly harder.

3

Dictionaries

1. **Hash table**. Consider a hash table with separate chaining, $M = 10$ positions with integer keys, and hash function $h(x) = x \mod M$.

   (a) Starting from an empty table, show the resulting table after inserting the keys 3441, 3412, 498, 1983, 4893, 3874, 3722, 3313, 4830, 2001, 3202, 365, 128181, 7812, 1299, 999 and 18267.

   (b) Show the resulting table after applying a rehash of the previous table into 20 positions.

2. **Hash table II**. Build the hash table with separate chaining for the keys 30, 20, 56, 75, 31 i 19, with $M = 11$ positions and hash function $h(x) = x \mod 11$.

   (a) Calculate the maximum number of comparisons in a successful search in this table.

   (b) Calculate the expected number of comparisons in a successful search in this table.

3. **BASIC's renum**. A BASIC program consists in a set of instructions numbered in increasing order. The control flow is managed through the instructions *GOTO x* and *GOSUB x*, where x is the instruction's number. For example, the following program calculates the factorial of a number:

```
50 INPUT N
60 LET F = 1
61 LET I = 1
73 IF I=N THEN GOTO 99
76 LET F = F*I
80 LET I = I+1
81 GOTO 73
99 PRINT F
```

   The procedure *RENUM* renumerates the instructions of the program in such a way that the lines go ten by ten. For example, after renumerating the previous program it

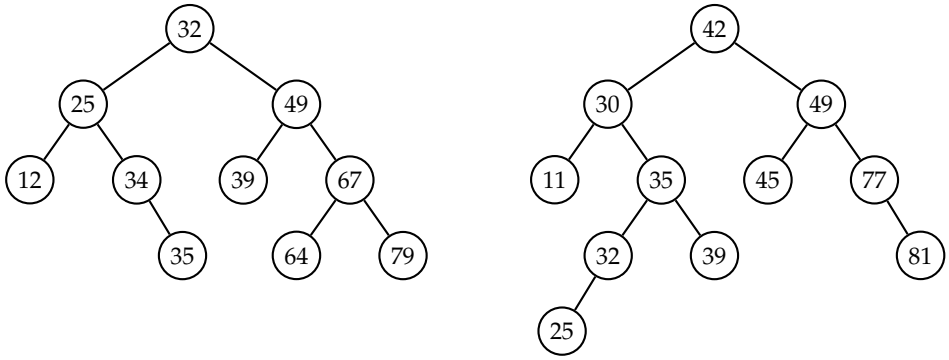
reads as follows:

```
10 INPUT N
20 LET F = 1
30 LET I = 1
40 IF I=N THEN GOTO 80
50 LET F = F*I
60 LET I = I+1
70 GOTO 40
80 PRINT F
```

Describe how to implement the procedure *RENUM* in linear time on average.

4. **All different**. Design and analyze an algorithm that uses a hash table to check that all the elements of a list are different.

5. **The ABC of BSTs**. State whether the following trees are binary search trees or not and why.

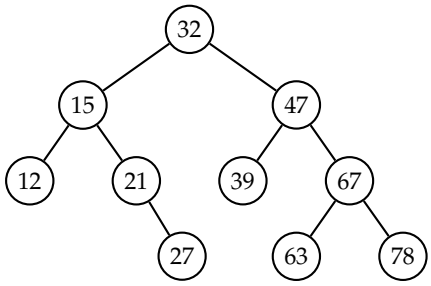


6. **Inserting in a BST**. Starting from an empty binary search tree, insert using the classic algorithm, one after the other, the sequence of keys $32, 15, 47, 67, 78, 39, 63, 21, 12, 27$.

7. **Inserting in a BST II**. Starting from an empty binary search tree, insert using the classic algorithm, one after the other, the sequence of keys $12, 15, 21, 27, 32, 39, 47, 63, 67, 78$.

8. **About the maximum in a BST**. Explain if it is true or false that in a non-empty binary search tree, the maximum element can have a left child node, but not a right one.

9. **Deleting in a BST**. Starting from the following binary search tree, eliminate the keys $63, 21, 15, 32$ one after the other. Explain which algorithm you used to eliminate the keys.



10. **Inorder of a BST**. Prove that the in-order traversal of a binary search tree visits the elements in increasing order.

11. **The BST of a given preorder**. Draw the binary search tree that results in the following pre-order traversal: 8, 5, 2, 1, 4, 3, 6, 7, 11, 9, 13, 12.

12. **BSTs implemented**. For this problem and those that follow, use this type definition for binary search trees:

```
struct node {
    Elem x;              // Information in the node
    node* lc;            // Pointer to the left child node
    node* rc;            // Pointer to the right child node

    node(Elem _x, node* _lc, node* _rc)
      : x(_x), lc(_lc), rc(_rc) {
    }

    ~node () {
        delete lc; delete rc;
    }
};

typedef node* bst;       // A binary search tree is denoted by a pointer
                         // to its root (null if it is empty)
```

    Implement a function *bst create* () that returns an empty binary search tree. What is the cost of the function?

13. **Search in a BST**. Implement a function **bool** *belongs*(*bst a*, *Elem x*) that indicates if the element *x* is inside the binary search tree *a*. What is the cost of the function?

14. **Insert in a BST**. Implement a method **void** *add*(*bst*& *a*, *Elem x*) that adds the element *x* to the binary search tree *a*. What is the cost of the method?

15. **Delete in a BST**. Implement a method **void** *remove*(*bst*& *a*, *Elem x*) that removes the element *x* from the binary search tree *a*. What is the cost of the method?

16. **Minimum in a BST**. Implement a function *Elem minimum*(*bst a*) that returns the minimum element in the non-empty binary search tree *a*. What is the cost of the function?

17. **Maximum in a BST**. Implement a function *Elem maximum*(*bst a*) that returns the biggest element in the non-empty binary tree *a*. What is the cost of the function?

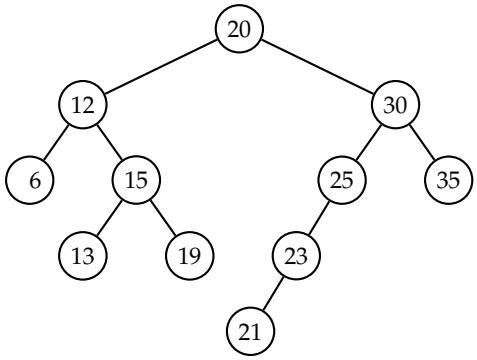
18. **Merge of two BSTs**. The *merge* of two binary search trees *a1* and *a2* is a binary search tree that contains all the elements of *a1* and *a2*. Implement a function

    bst merge(*bst*& *a1*, *bst*& *a2*)

    that returns the merge of the binary search trees *a1* and *a2* (both trees have to be empty after the call). What is the cost of the function?

19. **Segment of a BST**. Design a function *list <Elem> between*(*bst a*, *Elem x1*, *Elem x2*) that, given a binary search tree *a* and two elements *x1* and *x2* with $x1 \leq x2$, returns a list with all the elements of *a* that are between *x1* and *x2*, in decreasing order.

    For example, given the following tree

    and the values 18 and 26, the returning list is $\langle 25, 23, 21, 20, 19 \rangle$.

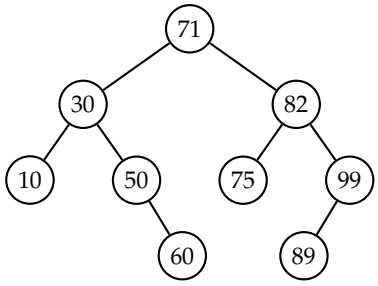
    Calculate the cost of your algorithm in the worst-case scenario.

20. **Smaller in a BST**. Implement a function **int** *smaller* (*bst a*, *Elem x*) that returns the number of elements in the binary search tree *a* that are strictly smaller than *x*. What is the cost of the funcion?

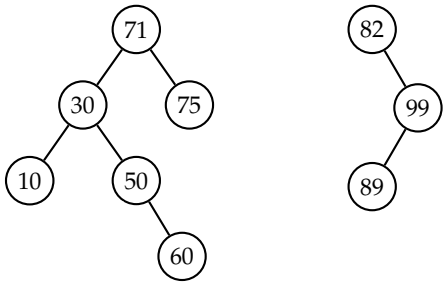
21. **Build the BST of a given preorder**. Implement a function *bst remake*(**vector**<*Elem*> *pre*) that reconstructs a binary search tree from its pre-order traversal stored in a vector *pre*. What is the cost of the function?

22. **Splitting a BST**. Implement a method **void** *split* (*bst*& *a*, *Elem x*, *bst*& *le*, *bst*& *gt*) that given a binary search tree *a*, returns two binary search trees *le* and *gt* where *le* contains all the elements of *a* that are smaller or equal than *x*, and *gt* contains all the elements of *a* that are bigger than *x*. The original tree *a* must be destroyed.

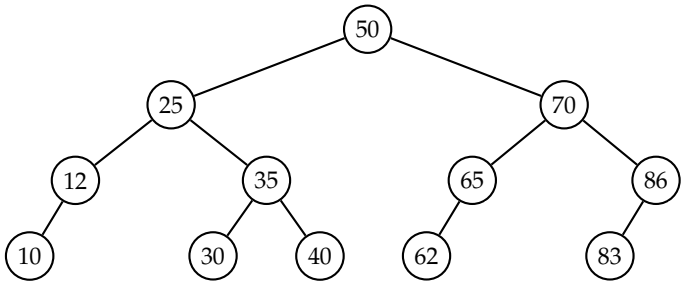
    For example, given the following binary search tree

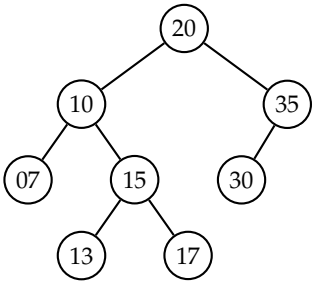
    and the element $x = 75$, the trees *le* and *gt* are the following:



23. **Inserting in an AVL**. Give the three AVL trees resulting from adding the keys 31, 32 and 33 one after the other to the following AVL tree:



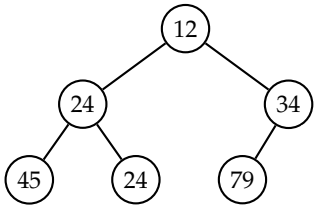
24. **Inserting and deleting in an AVL**. Give the four AVLs resulting from the addition of the keys 18 and 12 and the removal of the keys 7 and 30 to the following AVL tree (apply each operation to the tree obtained from the previous operation):

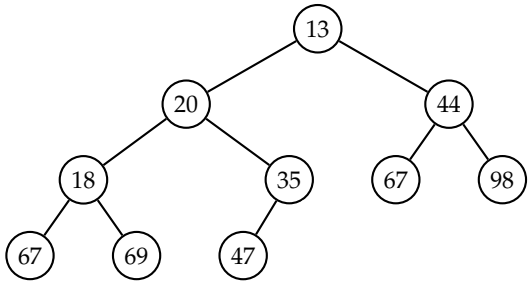


4

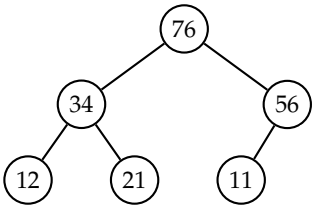
Priority queues

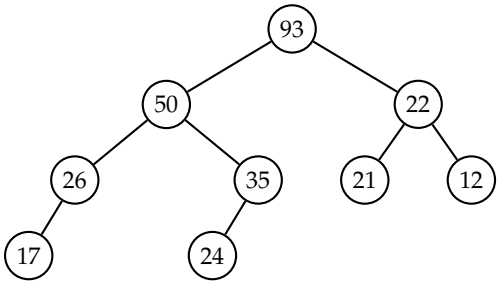
1. **Are they min-heaps?**. State if the following binary trees are min-heaps or not and why.





2. **Are they max-heaps?**. State if the following binary trees are max-heaps or not and why.

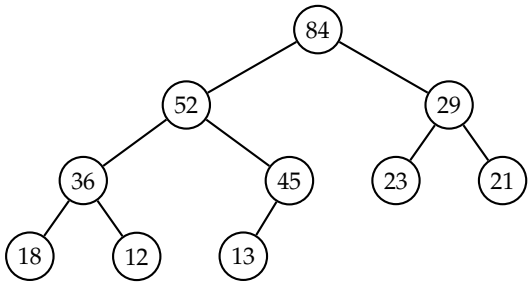




3. **Inserting in a heap**. Starting from an empty min-heap, insert one after the other the following numbers: 45, 67, 23, 46, 89, 65, 12, 34, 98, 76.

4. **Inserting in a heap II**. Starting from an empty max-heap, insert one after the other the following numbers: 45, 67, 23, 46, 89, 65, 12, 34, 98, 76.

5. **Deleting from a heap**. Starting from the following max-heap, eliminate one after the other the maximum element until it becomes an empty max-heap.



6. **Reconstructing the heap**. Given the following min-heap implemented as a table,

7	11	9	23	41	27	12	29

draw the tree represented by the table and draw the evolution of the table and the represented tree, when applying one after the other the operations of inserting the element 3 and eliminating the minimum.

7. **Reconstructing the heap II**. Convert the following table to a min-heap using the top-down heap construction algorithm.

45	53	27	21	11	97	34	78

8. **Reconstructing the heap III**. Convert the following table to a min-heap using the bottom-up heap construction algorithm.

45	53	27	21	11	97	34	78

9. **Test**. Validate or refute the following statements:
   (a) "A table with its elements sorted from lowest to highest is a min-heap."
   (b) "Inserting to a max-heap with n elements has a cost of $\Theta(\log n)$ in the worst case scenario."
   (c) "Finding the maximum element in a min-heap has cost of $O(\sqrt{n})$."
   (d) "Eliminating the maximum element of a max-heap with n different elements has a cost of $\Theta(1)$ in the best case."

10. **Partial sort**. The `partial_sort` algorithm of the STL takes a vector with n elements and a value m, $1 \leq m \leq n$, and re-sorts the vector in such a way that its first m elements are the lowest m elements of the original vector, in increasing order.
    (a) Explain how to implement this function in a way that its cost in the worst case scenario is $\mathcal{O}(n + m \log n)$.
    (b) How could it be implemented for its cost to be $\mathcal{O}(n \log m)$?
    (c) When should we use the first alternative and when the second one?

11. **k-th element out of n sorted tables**. Design an algorithm with cost $\Theta(k \log n + n)$ that, given n tables increasingly sorted with n elements each and a given k, finds the k-th global lowest element.

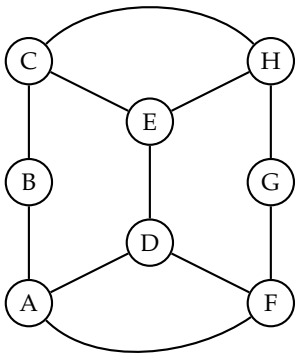
5

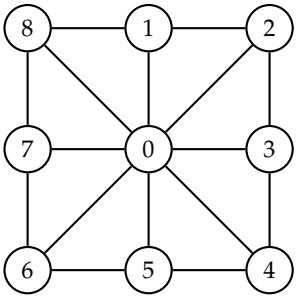
Graphs

1. **Draw the graph**. Draw the graph $G = (V, E)$ given by:
   (a) $V = \{1, 2, 3, 4, 5, 6\}$
   (b) $E = \{\{1, 2\}, \{1, 4\}, \{3, 2\}, \{4, 5\}, \{5, 1\}, \{5, 2\}\}$
   Is it connected? If not, how many connected components does it have?

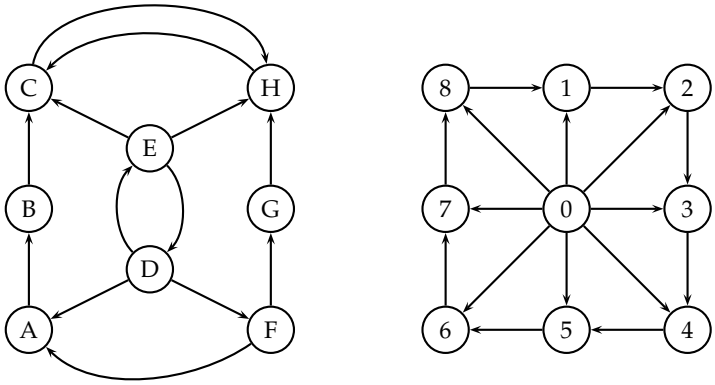
2. **Draw the graph II**. Draw the directed graph $G = (V, E)$ given by:
   (a) $V = \{1, 2, 3, 4, 5\}$
   (b) $E = \{(1, 2), (1, 4), (3, 2), (4, 5), (5, 1), (5, 2)\}$
   Is it strongly connected? Is it weakly connected?

3. **Represent the graphs**. Show how the following pair of graphs are represented using an adjacency matrix and adjacency lists. Count the degree of each vertex. Calculate the diameter of each graph.





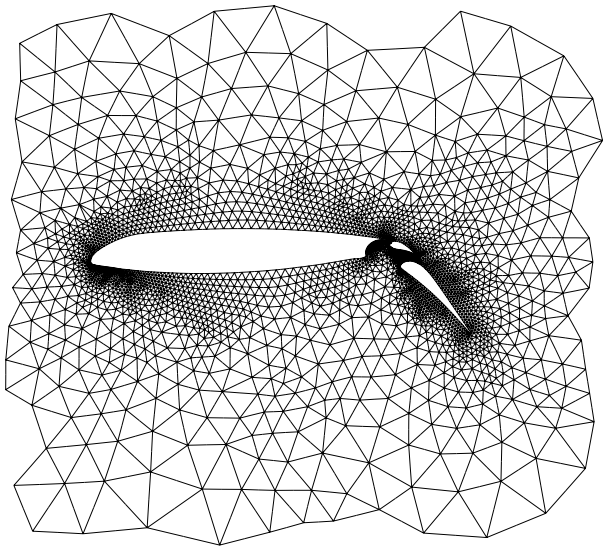
4. **Represent the graphs II**. Show how the following directed graphs are represented using an adjacency matrix and adjacency lists. Count the indegree and the outdegree of each vertex. Say whether the graphs are strongly or weakly connected.



5. **Sum of degrees**. Prove that in an undirected graph $G = (V, E)$ we have $\sum_{u \in V} \text{degree}(u) = 2|E|$.

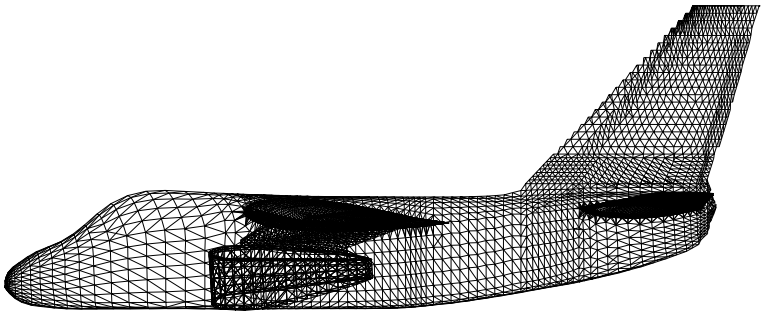
6. **Is there a graph like this?**. Consider an undirected graph with five vertices, all of them with degree 3. If such a graph exists, draw it. If it doesn't, give a reasoned explanation.

7. **The space it takes**. The graph of the following figure has $n = 4253$ vertices and $m = 12289$ edges.

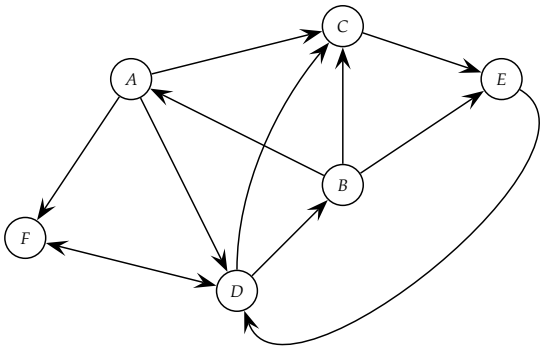


Assume that in your computer each boolean is 1 byte, each integer is 4 bytes and each pointer also 4 bytes. Calculate the amount of memory that is necessary to store this graph using an adjacency matrix representation and an adjacency lists representation.

8. **The space it takes II**. Repeat the previous problem for the following graph, which has $n = 156317$ vertices and $m = 1059331$ edges.



9. **Depth-first traversal**. List all the possible visit sequences of the vertices of this directed graph, applying a depth-first traversal that starts at vertex E.



10. **Breadth first traversal**. Repeat the previous problem applying a breadth first traversal that starts at vertex B.

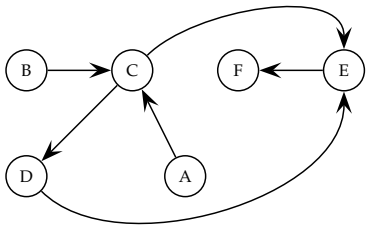
11. **Graphic mysteries**. Consider the following function on an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, implemented with adjacency lists (where vertices are identified with the integers 0, ... $n - 1$):

```
int mystery(const vector<vector<int>>& G) {
    int x = 0;
    for (int u = 0; u < G.size (); ++u)
        for (int w : G[u])
            ++x;
    return x;
}
```

    (a) Briefly explain what the function returns.

    (b) What's the cost of the function?

12. **Topological orderings**. Give, in lexicographical order, all possible topological orderings of the following directed acyclic graph (DAG):



13. **What if we add an isolated vertex?**. If we added an isolated vertex to the previous graph, how many topological orderings would it have? (Do not write them out, just say how many and explain why).

14. **Inverse topological ordering**. An *inverse topological ordering* of a DAG is a sequence formed by all the vertices of the graph in such a way that if (v, w) is an edge of the graph, then w appears before v in the sequence. Starting from the algorithm for depth-first traversal, design and analyse an algorithm that obtains an inverse topological ordering of a DAG.

15. **Roots and leaves in a DAG**. Write an algorithm that, given a DAG $G = \langle V, E \rangle$, calculates how many *roots* and how many *leaves* G has. A vertex v is a *root* of the DAG if its indegree is 0, and it is a *leaf* if its outdegree is 0. The graph is implemented with adjacency lists.

    Calculate the cost of the algorithm as a function of $n = |V|$ and $m = |E|$, and justify its correctness.

16. **Transposing**. The *transpose* of a directed graph $G = (V, E)$ is the directed graph $G^T = (V, E^T)$ where $E^T = \{(u, v \mid (v, u) \in E\}$. Design and analyse an algorithm that transposes a directed graph represented with an adjacency matrix.

    Do the same with a directed graph represented with adjacency lists.

17. **Squaring**. The *square* of a graph $G = (V, E)$ is another graph $G^Q = (V, E^Q)$ where $E^Q = \{\{u, v\} \mid \exists w \in V \mid \{u, w\} \in E \wedge \{w, v\} \in E\}$. Design and analyse an algorithm that, given a graph represented as an adjacency matrix, calculates its square.

    Do the same with a graph represented as adjacency lists.

18. **Squaring II**. Can the square of an n-vertex graph represented with an adjacency matrix be calculated in time lower than $\Theta(n^3)$?

    Hint: Strassen.

19. **Searching for boxes**. In an undirected graph, a *box* is a cycle of length 4. Design an algorithm that, given an undirected graph, determines whether this graph contains a box or not. Analyse its efficiency in different representations.

20. **Searching for boxes II**. Using the solution to problem **Squaring II**, design an algorithm for the previous problem running in time lower than $\Theta(n^3)$, where n is the number of vertices in the graph.

21. **The celebrity**. In a party, a guest is called a *celebrity* if everyone knows him, but he doesn't know anyone (except for himself). The acquaintance relations lead to a directed graph: each guest is a vertex, and an edge exists between u and v is u knows v.

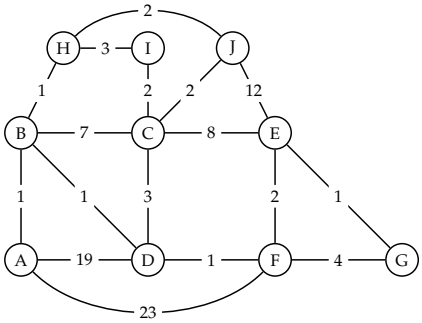
    Give an algorithm that, given a directed graph represented with an adjacency matrix, indicates if there is any celebrity. In a positive case, you have to say who the celebrity is. Your algorithm has to work with time $O(n)$, where n is the number of vertices.

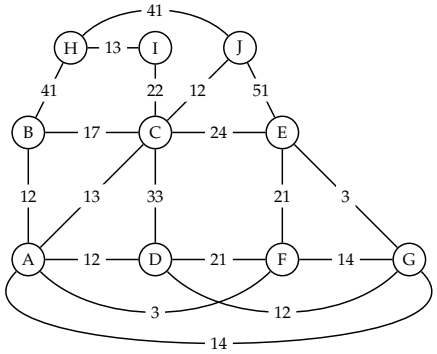
22. **Number of connected components**. Design and analyse an algorithm that calculates the number of connected components in an undirected graph.

23. **Is it acyclic?**. Design and analyse an algorithm that is able to determine if an undirected graph contains cycles or not in time $O(|V|)$.

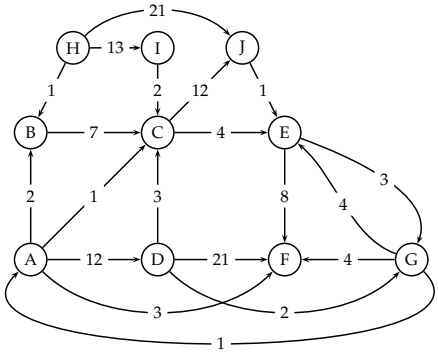
24. **Is it acylic? II**. Design and analyse an algorithm that determines if a directed graph contains cycles or not.

25. **Minimum cost path**. Find minimum cost paths to go from vertex A to vertex G in the following two graphs:





26. **One about Dijkstra**. Show how Dijkstra's algorithm calculates the minimum cost paths to go from vertex C to all the other vertices of the following directed graph:



27. **Negative cycles?**. Does it make sense to calculate minimum cost paths if there are negative cost cycles in a graph?

    Show a graph with negative weights for which Dijkstra's algorithm doesn't work, even though there are no negative cost cycles.

28. **Number of minimum cost paths**. Modify Dijkstra's algorithm to count the number of minimum cost paths between two given vertices. What if the graph had zero cost cycles? And if it had zero cost edges, but no zero cost cycles?

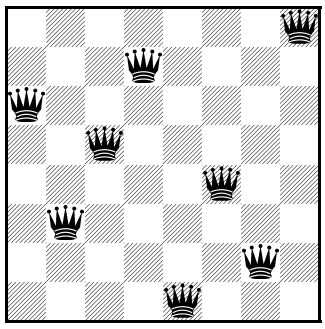
29. **Shortest minimum cost paths**. Modify Dijkstra's algorithm so that if there are more than one minimum cost path between two vertices, returns one with the minimum number of edges.

6

Generation and exhaustive search

1. **Eight queens**. Design an algorithm that writes all possible ways of placing n queens on an $n \times n$ chessboard so that no two queens attack each other.

   For example, this is an 8 queens configuration:



2. **Eight queens II**. Design an algorithm that counts in how many ways we can place n queens on an $n \times n$ chessboard so that no two queens attack each other.

3. **Eight queens III**. Design an algorithm that finds a possible way of placing n queens on an $n \times n$ chessboard so that no two queens attack each other, or indicates that such a placement is not possible.

4. **Knight jumps**. We place a knight on a given square of an $n \times n$ chessboard. Design an algorithm to find if there exists some way of applying $n^2 - 1$ knight moves to visit all the squares of the board.

   For example, this is a possible solution for a 5×5 chessboard starting from the center:

22	5	16	11	24
15	10	23	6	1
4	21	12	17	12
9	14	19	2	7
20	3	8	13	18

5. **Latin squares**. A latin square of order n is an $n \times n$ table where each square is colored using a palette of n possible colors, and where each color is used exactly once in each row and each column.

   For example, this is a latin square of order 5:

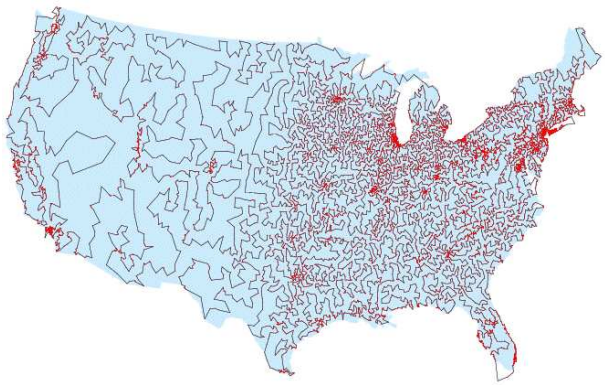
0	2	3	1	4
1	3	4	2	0
2	1	0	4	3
3	4	1	0	2
4	0	2	3	1

   Design an algorithm that writes all latin squares of order n.

6. **Hamiltonian graph**. Design and analyse an algorithm to determine if a connected undirected graph is Hamiltonian or not. Modify it to obtain a Hamiltonian cycle if it exists.

7. **Traveller Salesman Problem**. A traveller must visit n clients that live in n different cities. The distance from city i to city j is $D[i][j]$. Starting at one of the cities, the traveller wants to visit each city exactly once and then return to its starting point. Moreover, he wants to minimize the total travelled distance.

   For example, this is the travel plan of a trade traveller through the USA:



   Design and analyse an algorithm to solve the traveller's problem. Do it with a *backtracking* scheme and with a *branch and bound* scheme.

8. **Clique**. Given an undirected graph $G = (V, E)$, a *clique* is a subset of vertices $S \subseteq V$ such that there is an edge in the graph between u and v for any pair of vertices u and v in S.

   Design and analyse an algorithm that, given a graph G and a natural k, determines if G contains some clique of order k.

9. **Independent set**. Design and analyse an algorithm that, given a graph $G = (V, E)$ and a natural number k, determines if G has an independent set of k vertices (meaning that there exists a subset $A \subseteq V$ of k vertices such that $\{u, v\} \notin E$ for all $u, v \in A$).

10. **Vertex cover**. Design and analyse an algorithm that, given a graph $G = (V, E)$ and a natural number k, determines if G has a covering of the edges with k vertices. Remember that this means that there exists a subset $A \subseteq V$ of k vertices such that, for each $\{u, v\} \in E$, at least one of the endpoints u or v belongs to A.

11. **Subset sum**. Design an algorithm that, given a collection C of n positive integer numbers (possibly with repetitions) and a positive integer number S, determines if there is a subset of C that adds exactly S.

12. **Graph coloring**. We have $c \geq 3$ different colours and a map of n countries, and a Boolean table $v[i][j]$ that indicates if two countries i and j are neighbours. Write an exhaustive search algorithm that finds all possible colourings of the map with the restriction that any two neighbouring countries must be painted with different colours.

    How would you solve the problem for $c = 2$ (bi-colouring)?

13. **Packing**. We are given a collection of n objects that we need to pack in containers of volume capacity C. Object i has volume $v[i]$. Design an algorithm that calculates an optimal packing, namely, the packing that minimizes the number of containers. The objects cannot be fractioned but, in order to simplify the problem, you may assume that an object fits in a partially filled container if the volume of the object is smaller or equal than the free space that is left in the container (independently of the shape that the object may have).

7

Intractability

1. **A Tragicomedy in Three Acts**. Consider the following **NP**-complete problems:

   (a) BIN-PACKING: Given n items with sizes $d_1, \ldots, d_n$ and k bins with capacity m, determine whether all the items can be packed into the bins so that the capacity is not exceeded.

   (b) TRIPARTITE-MATCHING: Given three disjoint sets C_1, C_2, C_3 having n elements each and given a subset $S \subseteq C_1 \times C_2 \times C_3$, determine whether it is possible to select n elements from S so that every element from C_1, C_2, and C_3 appears only once in these n triples.

   (c) 3-COLORABILITY: Given a graph, determine whether it is possible to assign colors to vertices, choosing among three possible colors, so that no edge has its endpoints of the same color.

   (d) HAMILTONIAN-GRAPH: Given a graph, determine whether it has a cycle visiting each vertex exactly once.

   (e) DOMINATING-SET: Given a graph and a positive integer k, determine whether it has a subset with k vertices such that every vertex outside the subset is adjacent to a vertex in the subset.

   Each of the following scenes presents a problem. Show that the problem is hard after identifying it with one of the problems in the previous list.

   Scene 1. Johnny asked Roy to organize a dinner for the usual group of friends. Then, Roy has booked a table at a restaurant; it's a very big round table and everybody fits. However, when they get at the place, the first difficulties arise.

   JOHNNY: Listen Roy, it seems that some people in the group can't stand each other. It would be better if two guys who are angry with each other don't have to sit side by side.

   ROY: Come on... Look, let's get straight to the point. I want you to collect all the

relevant information; that is, for each pair of people, whether they are angry with each other or not. I'll figure out how to seat them so that nobody has a neighbor at the table with whom he or she is angry.

Scene 2. With all the data in his hands, after a long while, Roy hasn't yet finished finding a solution. Moreover, Johnny is back with a worried face.

JOHNNY: Hey, it seems that the guys who are angry with each other don't even want to sit at the same table. But don't worry, I've asked at the restaurant and they said they can prepare three large tables for us.

ROY: Let's see, I'll try to distribute people into three groups so that there's no couple of angry fellows inside any of the groups.

Scene 3. While Roy thinks on how to distribute people around the three tables, Johnny appears again saying that the restaurant is about to close and that they'd better look for a different place for dinner.

ROY: Look, before looking for another place, I'd like to make some things clear to the group: When they say "hey, hey, I don't want to sit near this or that person", they're being childish! I want to talk to them seriously. The problem is that I'm a little aphonic and if I speak to everybody, I'll ran out of voice. I want you to choose k representatives so that, for any person in the group, there is one of the representatives with whom that person is not angry. This way, the k representatives will spread the message to the whole group.

Scene 4. Since Johnny doesn't succeed in choosing the k representatives, he finally takes a megaphone and tells everybody that if they keep acting like that, they'll have to go home very soon.

JOHNNY: Listen, I have a city map where I've located the squares having good restaurants and the streets joining them. We could walk through all of them and check whether they have free tables.

ROY: OK, give me the map for a while so that I can choose the tour; it's cold outside and I'm not feeling like visiting the same place twice.

Scene 5. After a long while, Roy still can't find the solution in spite of having all the available information. Johnny wants to help him:

JOHNNY: Listen Roy, I have a handful of colleagues in town and for each of these squares, a friend of mine lives there. Give me some bucks and I'll phone them and ask them if we can have dinner at any restaurant in their squares.

ROY: Man, I'm skint and can't afford so many calls. I'll give you enough for m phone calls. You can ask your friends to look for restaurants not only in their squares but also in the squares one street away from theirs. Choose the right friends so that they can cover all the squares.

Scene 6. Since he doesn't succeed, Johnny calls to the first person in the list and it turns out that there is a restaurant with free tables near him. When they get there, there is only one table with k chairs, and they are closing soon.

JOHNNY: I got it, here's a solution: We'll have dinner by turns; when somebody finishes, he or she will go out and somebody else will come in. Look, some time

ago I collected information about how long everyone takes for dinner. You know I have some weird hobbies...

ROY: My God...

JOHHNY: With all this data, we can plan the distribution of people on the chairs, so that everybody can have dinner before they close.

ROY: OK, give me that list with the time everybody takes to have dinner and I'll see what can I do. Someday you'll tell me about what information you collect from people...

Scene 7. In the meanwhile, Johnny gets a phone call from a friend saying that near that place there's a very cool restaurant. They all go there at once. When they get there, it seems there are only a few small tables and, on top of that, they all have different colors.

JOHNNY: Listen, Roy. It seems that people are a little bored and they want to take advantage of the situation to sit properly by couples. We are as many boys as girls, and everybody has their own preferences. But the colored tables play a role, too. For example, that person accepts to sit with that other person only if the table is pink, and with some other person if the table is blue. We'll have to find a distribution that satisfies everyone's restrictions. It seems that after all this time without eating, the troop is feeling a bit strange.

ROY: I'm not in the mood with all this nonsense, but OK, I'll try.

Scene 8. They haven't succeeded and it is already very late. Roy and Johnny can only find a small bar having just one table with one chair. However, it is open until very late.

JOHNNY: It seems that the band is a bit bored. Some guys want to go out and come back later. Here I have, for every person, the time they'll come back and the time they'll have to go home. And I still have the list of the time everybody needs to have dinner. I think that, with this information, it should be easy to see if everybody can be assigned a time so that they can sit on the chair and eat according to their schedules.

ROY: Listen Johnny, I'm fed up. I want you to know that this is going to be the last thing I do for this gang. Come on, give me this data and I'll see what can I do.

[In this scene, it is not enough to identify a problem from the list; it is necessary to find a reduction.]

Scene 9. Finally, Roy and Johnny decide to have dinner together at that place, they have only had to ask for another chair to the waitress. They have given an address to the rest of the group where they will just find a wasteland with the highest criminality index in town.

JOHNNY: Come on, man, don't take it like that. Today we just had bad luck.

ROY: Look, it's not bad luck at all. It's not the first time this happens to me. I'm foolish and I'll never learn. Now, I want you to listen carefully. I know I told you before, but this time I'm serious: I'll never ever organize anything else for this gang. And when I say never, it's never. In fact, I don't want even to meet them again.

JOHNNY: OK, right! But it's a pity. Next week, we were thinking of going together to a holiday camp. You know, nature, pure air, quiet, fun, roast meat, naked baths in the river...

ROY: Well, that sounds good! But I don't know, these things always go wrong in the end.

JOHNNY: Besides, this time Steffy is coming. Do you remember?

ROY: STEFFY!?!?!?

JOHNNY: Yes, and she said she's longing very much to see you. I think she's going to have a big disappointment if you don't show up.

ROY: Hummm... well... maybe I've gone a bit too far. In the end, they are good guys... Sometimes they're unbearable, but they're good people, and that's what matters. Friendship, in fact, is full of bad moments, but it's mainly full of marvelous moments of bliss. Come on! What the hell! I'm coming. And if you want, I can help you organize things.

JOHNNY: Perfect! You're the best! A winner, for sure! Come and let's organize it right now. Let's see, the only problem is that there are k cars and...

2. **Understanding the Definitions**. We have a problem X and we have proved that SAT reduces to X (formally, SAT $\leq_m X$). Which one of the following statements can we derive?

   (a) X belongs to **NP**, but we do not know whether it is **NP**-complete.

   (b) X is an **NP**-complete problem.

   (c) None of the above.

   Now, a colleague has additionally found a reduction in the reverse direction, $X \leq_m$ SAT. What can we conclude now?

   (i) X belongs to **NP**, but we do not know whether it is **NP**-complete.

   (ii) X is an **NP**-complete problem.

   (iii) None of the above.

3. **Reductions in What Direction?**. The problems of satisfiability of Boolean formulas (SAT) and of determining whether a given graph is Hamiltonian (HAM) are two well-known **NP**-complete problems. The problem of determining whether a given graph contains an Eulerian circuit (EUL) has a solution with cost $\Theta(n + m)$, where n is the number of vertices in the graph and m is its number of edges.

   Which one of the following sentences can we say it is true?

   (a) SAT reduces to HAM and HAM reduces to EUL.

   (b) EUL reduces to SAT and SAT reduces to HAM.

   (c) SAT and HAM reduce to each other; EUL only reduces to HAM.

   (d) SAT reduces to EUL and EUL reduces to HAM.

4. **PARTITION**. Reduce problem SUBSET-SUM to problem PARTITION: Given a set of integers S (possibly with repetitions), determine whether S can be *partitioned* into two subsets S_1 i S_2 (i.e., each element of S belongs either to S_1 or to S_2, but not the two at the same time) such that $\sum_{x \in S_1} x = \sum_{x \in S_2} x$. In addition, prove that PARTITION belongs to **NP**.

5. **DELETED-SUBGRAPH**. Reduce problem HAMILTONIAN-GRAPH to DELETED-SUBGRAPH: Given two graphs G_1 and G_2, determine whether it is possible to delete some edges of G_2 in such a way that the resulting graph is isomorphic to G_1. Additionally, prove that DELETED-SUBGRAPH belongs to **NP**.

6. **CLIQUE**. Reduce problem INDEPENDENT-SET to CLIQUE: Given a graph and a positive integer k, determine whether there is a subset of k vertices such that there is an edge between every pair of vertices in the subset. Additionally, prove that CLIQUE belongs to **NP**.

7. **INDUCED-SUBGRAPH**. Reduce problem CLIQUE to problem INDUCED-SUBGRAPH: Given two graphs G_1 and G_2, determine whether G_1 is an induced subgraph of G_2. Additionally, prove that INDUCED-SUBGRAPH belongs to **NP**.

8. **One About Steffy**. Probably you will remember that Johnny and Roy organized an excursion with their big group of friends (and Steffy!) to spend a weekend in a holiday camp. Today they have decided to go rafting, but there is only one boat. Additionally, the boat is only safe if people aboard weigh exactly a total of T kilograms (because if they weigh more, the boat sinks and, if they weigh less, the stream is not strong enough to drag them to destination).

   Read the following dialog assuming that all the characters say the truth.

   JOHNNY: Roy, ask everybody's weight so that we can see if it's worth to rent a boat.

   ROY: Here you have, you know I like to make lists of people.

   STEFFY: Hey, guys... It's better you don't try. The problem you want to solve (given an integer T and given n strictly positive integers $p_1, \ldots, p_n$, to determine whether there are a few whose sum is T) is **NP**-complete.

   JOHNNY: Damn, we've got a problem!

   ROY: Wait! By coincidence, I have with me a magic lantern I bought at Istanbul market, in a shop where they speak English. They told me that if you rub, a genie will come out who can efficiently solve any **NP** problem.

   JOHNNY: OK, come on, give it to me!

   [Johnny rubs the lantern and a genie comes out in a smoke cloud.]

   GENIE: Greetings, boy. If you give me m integers $b_1, \ldots, b_m$, I'll determine at the moment whether there's some nonempty subset with zero sum.

   JOHNNY: Damn it, Roy! This is not our problem!

   ROY: Hummm... Maybe that shopper in Istanbul cheated me.

   STEFFY: Don't worry, guys. The shopper didn't lie... I have the solution!

(a) Explain which is the solution Steffy has found to know whether some people in the group can go rafting (that is, give a reduction from Johnny's problem to the genie's problem).

(b) Finish the proof that the genie's problem is **NP**-complete.