

## Examen Parcial EDA - torn 1

Duració: 2 hores

18/10/2010

## Problema 1

(1 punt)

- Doneu la definició de  $O(f)$ :

- El teorema mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma  $T(n) = aT(n/b) + \Theta(n^k)$  amb  $b > 1$  i  $k \geq 0$ , llavors, fent  $\alpha = \log_b a$ ,

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

## Problema 2

(2 punts)

Ompliu els blancs de la forma més precisa possible.

- Quicksort ordena  $n$  elements en temps  en el cas pitjor.
- Per multiplicar dues matrius grans eficientment podem fer servir l'algorisme de .
- $2 + \cos(n) = \Theta(\text{ })$ .
- $2\sqrt{n} + 1 + n + n^2 = \Theta(\text{ })$ .
- $\log n + \log \log(n^2) = \Theta(\text{ })$ .
- $\frac{n^2 - 6n}{2} + 5n = \Theta(\text{ })$ .
- $n^2 - 3n - 18 = \text{ } (n)$ .
- Si

$$T(n) = \begin{cases} 4 & \text{si } n = 1, \\ 3T(n/2) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(\text{ })$ .

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 2T(n-1) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(\text{ })$ .

**Problema 3****(1.5 punts)**

Sigui  $c \in \mathbb{R}$  i sigui  $h$  una funció de  $\mathbb{N}$  a  $\mathbb{R}$ . Recordeu que

$$\lim_{n \rightarrow \infty} h(n) = c \iff \forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : |h(n) - c| \leq \epsilon.$$

Sigui  $c \in \mathbb{R}$  i siguin  $f$  i  $g$  dues funcions de  $\mathbb{N}$  a  $\mathbb{R}$ . Demostreu que

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \implies f \in O(g).$$

**Problema 4****(1 punt)**

Suposeu que  $\text{primer}(n)$  és una crida a una funció amb temps  $O(\sqrt{n})$ .

Considereu un procediment amb cos principal:

```
if (primer(n)) A;  
else B;
```

Doneu fites senzilles i ajustades amb notació  $O$  per al temps d'aquest procediment, en funció de  $n$ , suposant que:

1.  $A$  triga  $O(n)$  i  $B$  triga  $O(1)$ .
2.  $A$  i  $B$ , ambdòs, triguen  $O(1)$ .

**Problema 5****(1 punt)**

```
int misteri (int n) {  
    if (n == 1) return 1;  
    return misteri (n-1) + 2*n - 1;  
}
```

- Diguen què calcula la funció *misteri*.
- Doneu el seu cost en funció de  $n$ .

**Problema 6****(2 punts)**

Dissenyau un algorisme de temps  $O(n \log m)$  que trobi la unió  $A \cup B$  de un conjunt  $A$  de  $n$  elements amb un conjunt  $B$  de  $m$  elements, amb  $m \leq n$ . Els conjunts  $A$  i  $B$  són de nombres reals i estan representats per vectors no necessàriament ordenats i sense elements repetits. La sortida és un vector, no necessàriament ordenat, sense elements repetits que representa la unió.

Quins canvis requereix el vostre algorisme si es demana la intersecció en comptes de la unió?

**Problema 7****(1.5 punts)**

```

void misteri_2 (vector<int>& T, int e, int m1, int m2, int d) {
    vector<int> B(d-e+1);
    int i = e, j = m1+1, k = m2+1, l = 0;
    while (i ≤ m1 and j ≤ m2 and k ≤ d) {
        if (T[i] ≤ T[j] and T[i] ≤ T[k]) B[l++] = T[i++];
        else if (T[j] ≤ T[k]) B[l++] = T[j++];
        else B[l++] = T[k++];
    }
    while (i ≤ m1 and j ≤ m2) {
        if (T[i] ≤ T[j]) B[l++] = T[i++];
        else B[l++] = T[j++];
    }
    while (i ≤ m1 and k ≤ d) {
        if (T[i] ≤ T[k]) B[l++] = T[i++];
        else B[l++] = T[k++];
    }
    while (j ≤ m2 and k ≤ d) {
        if (T[j] ≤ T[k]) B[l++] = T[j++];
        else B[l++] = T[k++];
    }
    while (i ≤ m1) B[l++] = T[i++]; while (j ≤ m2) B[l++] = T[j++];
    while (k ≤ d) B[l++] = T[k++];
    for (l=0; l ≤ d-e; ++l) T[e+l] = B[l];
}

void misteri_1 (vector<int>& T, int e, int d) {
    if (e+1 == d) {
        if (T[e] > T[d]) swap(T[e], T[d]);
    }
    if (e+1 < d) {
        int m1 = e + (d-e+1)/3;
        int m2 = e + 2*(d-e+1)/3;
        misteri_1 (T, e, m1); misteri_1 (T, m1+1, m2); misteri_1 (T, m2+1, d);
        misteri_2 (T, e, m1, m2, d);
    }
}

void misteri (vector<int>& T) {
    misteri_1 (T, 0, T.size ()-1);
}

```

De quin algorisme clàssic és variant aquest algorisme? Quin és el seu cost?

## Examen Parcial EDA - torn 2

Duració: 2 hores

18/10/2010

## Problema 1

(1 punt)

- Doneu la definició de  $\Omega(f)$ :

- El teoreme mestre de resolució de recurrències substractores afirma que si tenim una recurrència de la forma  $T(n) = aT(n - c) + \Theta(n^k)$  amb  $c > 0$  i  $k \geq 0$ , llavors,

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

## Problema 2

(2 punts)

Ompliu els blancs de la forma més precisa possible.

- Per multiplicar dos naturals molts llargs eficientment podem fer servir l'algorisme de .
- Per ordenar  $n$  elements en temps  $\Theta(n \log n)$  en el cas pitjor podem servir l'algorisme de .
- Multipliqueu  $\log n + 6 + O(1/n)$  per  $n + O(\sqrt{n})$ . El resultat simplificat tan com possible és  $O(\text{ })$ .
- $\sin(n) + 10 + n = \Theta(\text{ })$ .
- $\frac{1}{3}n^2 + 3n \log n + 5n^8 = \Theta(\text{ })$ .
- $n\sqrt{n} + n^2 = \Theta(\text{ })$ .
- $3n \log n = \text{ } (n^2)$ .
- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 3T(n/2) + n - 2 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(\text{ })$ .

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 4T(n-1) + n & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(\text{ })$ .

**Problema 3****(1.5 punts)**

Sigui  $c \in \mathbb{R}$  i sigui  $h$  una funció de  $\mathbb{N}$  a  $\mathbb{R}$ . Recordeu que

$$\lim_{n \rightarrow \infty} h(n) = c > 0 \iff \forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : |h(n) - c| \leq \epsilon.$$

Sigui  $c \in \mathbb{R}$  tal que  $c > 0$  i siguin  $f$  i  $g$  dues funcions de  $\mathbb{N}$  a  $\mathbb{R}$ . Demostreu que

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \implies f \in \Omega(g).$$

**Problema 4****(1 punt)**

```

int misteri (int m, int n) {
    int result = 0;
    while (m > 0) {
        if (m % 2 != 0) result += n;
        m /= 2; n *= 2;
    }
    return result ;
}

```

- Diguen què calcula la funció *misteri*.

- Doneu el seu cost en funció de  $m$ .

**Problema 5****(1 punt)**

Considereu un procediment amb cos principal:

```

int k = f(n);
int s = 0;
for (int i = 1; i ≤ k; ++i) s += i;

```

amb  $f(n)$  una crida a la funció  $f$ .

Doneu fites senzilles i ajustades amb notació  $O$  per al temps d'aquest procediment, en funció de  $n$ , suposant que:

1. El temps de  $f(n)$  és  $O(n)$  i el valor de  $f(n)$  és  $n!$ .

2. El temps de  $f(n)$  és  $O(n)$  i el valor de  $f(n)$  és  $n$ .

3. El temps de  $f(n)$  és  $O(n^2)$  i el valor de  $f(n)$  és  $n!$ .

4. El temps de  $f(n)$  és  $O(1)$  i el valor de  $f(n)$  és 0.

**Problema 6****(2.5 punts)**

Donada una seqüència  $A$  de  $n$  enters representada per un vector no necessàriament ordenat, es vol un algorisme de temps  $O(n \log n)$  per determinar si  $A$  conté més de  $n/2$  elements iguals.

1. Ordenant el vector amb un algorisme d'ordenació de temps  $O(n \log n)$  és immediat resoldre aquest problema. Digueu com.
2. Supposeu que els elements només es poden comparar per igualtat ( $=$ ,  $\neq$ ) però no es poden comparar per ordre ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ). Proposeu un algorisme de Dividir i Vèncer de temps  $O(n \log n)$  que resolgui el problema.

**Problema 7****(1 punt)**

```
int misteri_2 (vector<int>& T, int e, int d) {
    int x = T[d];
    int i = e-1;
    for (int j = e; j <= d; ++j) {
        if (T[j] <= x) swap(T[++i], T[j]);
    }
    if (i < d) return i;
    return i-1;
}
```

```
void misteri_1 (vector<int>& T, int e, int d) {
    if (e < d) {
        int q = misteri_2 (T, e, d);
        misteri_1 (T, e, q);
        misteri_1 (T, q+1, d);
    }
}
```

```
void misteri (vector<int>& T) {
    misteri_1 (T, 0, T.size ()-1);
}
```

De quin algorisme clàssic es tracta? Quin és el seu cost quan tots els elements són iguals?

## Examen Parcial EDA

Duració: 2 hores

21/03/2011

## Problema 1

(5 punts)

Ompliu els blancs de la forma més precisa possible.

- Insertion sort ordena  $n$  elements en temps  en el cas millor.
- Trobar la mediana de  $n$  elements té una complexitat en temps  $\Omega(\text{  })$ .
- Com de ràpid es poden multiplicar una matriu  $kn \times n$  per una  $n \times kn$  fent servir l'algorisme de Strassen? .
- Quantes línies, en funció de  $n$  i notació  $\Theta$ , imprimeix el següent programa? Escriviu una recurrència i resoleu-la.

```
//pre: n ≥ 1
void f(int n) {
    if (n > 1) {
        cout << "segueix" << endl;
        f(n/2);
        f(n/2);
    }
}
```

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n < 3, \\ 9T(n/3) + n^2 & \text{si } n \geq 3. \end{cases}$$

Aleshores,  $T(n) = \Theta(\text{  })$ .

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n = 1, \\ 3T(n-1) + n^3 & \text{si } n > 1. \end{cases}$$

Aleshores,  $T(n) = \Theta(\text{  })$ .

- Multipliqueu  $101 \times 101$  (en binari) fent servir l'algorisme de Karatsuba. Mostreu els passos que heu seguit.
- Les taules següents representen etapes de l'execució dels algorismes d'ordenació per fusió, ràpida, per inserció i per selecció (mergesort, quicksort, insertion sort i selection sort) aplicats a la taula: 

10	2	5	3	7	13	1	6
----	---	---	---	---	----	---	---

 Escriviu al costat de cada taula a quin algorisme dels anteriors correspon (suposeu que hi ha una taula de cada algorisme).

2	3	5	10	1	6	7	13	
2	3	5	10	7	13	1	6	
1	2	5	3	7	6	13	10	
1	2	3	5	6	13	10	7	

- Indiqueu per a cada parell de funcions  $(A, B)$  a la taula següent si  $A$  és  $O$ ,  $\Omega$  o  $\Theta$  de  $B$ . Supposeu que  $k \geq 1$ ,  $\epsilon > 0$  i  $c > 1$  són constants. La resposta ha de ser “sí” o “no” a cada casella buida.

$A$	$B$	$O$	$\Omega$	$\Theta$
$\log^k(n)$	$n^\epsilon$			
$\log n$	$\log \log(n^2)$			
$n^k$	$c^n$			
$2^{n+1}$	$2^n$			
$2^{2n}$	$2^n$			

**Problema 2****(2.5 punts)**

```

int misteri_2 (vector<int>& T, int e, int d) {
    int x = T[d];
    int i = e-1;
    for (int j = e; j ≤ d; ++j) {
        if (T[j] ≤ x) swap(T[++i], T[j]);
    }
    if (i < d) return i;
    return i-1;
}

int misteri_1 (vector<int>& T, int e, int d, int k) {
    if (e == d) return T[e];
    int q = misteri_2 (T, e, d);
    if (q - e + 1 ≥ k) return misteri_1 (T, e, q, k);
    return misteri_1 (T, q + 1, d, k - q + e - 1);
}

// pre: 1 ≤ k ≤ n = T.size()
int misteri (vector<int>& T, int k) {
    return misteri_1 (T, 0, T.size ()-1, k);
}

int main() {
    int k, x;
    cin >> k;
    vector<int> v;
    while (cin >> x) v.push_back(x);
    cout << misteri(v, k) << endl;
}

```

- Digueu què calcula el programa anterior.
- Doneu una recurrència que descrigui el seu cost en temps en funció de  $n$  en el cas pitjor (justifiqueu) i resoleu-la.
- Doneu el seu cost en temps en funció de  $n$  en el cas millor (justifiqueu).
- Digueu a quin algorisme clàssic us recorda l'algorisme anterior i perquè.



**Problema 3****(2.5 punts)**

Escriuiu un algorisme que, donat un natural  $N \geq 0$ , calculi  $\lfloor \sqrt{N} \rfloor$  (la part entera per defecte de l'arrel quadrada de  $N$ ) en temps  $\Theta(\log N)$  en el cas pitjor. Justifiqueu-ne la complexitat.

Ajuda: penseu en la cerca dicotòmica.

## Examen Parcial EDA

Duració: 2 hores

20/10/2011

## Problema 1

(4,5 punts)

- (0,5 punts) Calculeu  $101 + 102 + \dots + 999 + 1000$ .
- (0,5 punts) Expliqueu per quina raó l'afirmació "El temps d'execució de l'algorisme  $A$  amb entrada  $n$  és com a mínim  $O(n^2)$ " no significa res.
- (1 punt) Ordeneu les funcions següents segons el seu ordre de creixement asimptòtic:  $5\log(n + 100)^{10}$ ,  $2^{2n}$ ,  $0.001n^4 + 3n^3 + 1$ ,  $(\ln(n))^2$ ,  $\sqrt{n}$ ,  $3^n$ .
- (1,5 punts) Considereu les tres alternatives següents per resoldre un mateix problema:
  - L'algorisme  $A$  resol una instància del problema dividint-la en cinc instàncies de la meitat de la talla, resolent recursivament cada instància, i combinant les solucions en temps lineal.
  - L'algorisme  $B$  resol una instància de talla  $n$  resolent recursivament dues instàncies de talla  $n - 1$  i combinant les solucions en temps constant.
  - L'algorisme  $C$  resol una instància de talla  $n$  dividint-la en nou instàncies de talla  $n/3$ , resolent recursivament cada instància, i combinant les solucions en temps  $\Theta(n^2)$ .

Analitzeu el cost de cadascun dels tres algorismes. Quin és el més eficient?

- (1 punt) Sigui  $x$  un natural representat en una taula de  $n$  bits. Doneu, en funció de  $n$ , el cost de calcular  $x + 1$  en els casos pitjor i millor.

## Problema 2

(3.5 punts)

- (1 punt) Analitzeu el cost temporal de *fusio2* en funció de les mides de  $a$  i  $b$ :

```
vector<int> fusio2(const vector<int>& a, const vector<int>& b) {
    // Pre: a i b estan ordenats de forma no-decreixent
    int na = a.size();
    int nb = b.size();
    vector<int> c(na + nb);
    int ia = 0;
    int ib = 0;
    int j = 0;
    while (ia < na and ib < nb) {
        if (a[ia] <= b[ib]) { c[j] = a[ia]; ++ia; }
        else { c[j] = b[ib]; ++ib; }
        ++j;
    }
    while (ia < na) { c[j] = a[ia]; ++ia; ++j; }
    while (ib < nb) { c[j] = b[ib]; ++ib; ++j; }
    return c;
}
```

- (1 punt) Suposem que tenim  $k$  vectors ordenats (de manera no-decreixent), cadascun amb  $n$  elements, guardats en un `vector<vector<int>>` de mida  $k$ . Volem calcular el `vector<int>` de mida  $kn$  resultant de fer la fusió de tots ells.

Un possible algorisme consisteix en fusionar usant *fusio2* el primer i el segon vectors, després fusionar el resultat amb el tercer vector, després amb el quart, etc. Quin és el cost en temps d'aquest algorisme, en funció de  $k$  i  $n$ ?

- (1,5 punts) Implementeu en C++ una funció

`vector<int> fusio2(const vector<int> &t)`

que resolgui més eficientment el problema. Justifiqueu el seu cost temporal.

### Problema 3

(2 punts)

Els nombres de Fibonacci satisfan la recurrència  $f_{n+2} = f_{n+1} + f_n$  per a  $n \geq 0$ , amb  $f_0 = 0$  i  $f_1 = 1$ . Demostreu que, per a  $n \geq 1$ , se satisfà la identitat següent:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}$$

i aprofiteu-la per donar un algorisme de cost  $\Theta(\log n)$  que, donat un natural  $n$ , calculi  $f_n$ . No es valorarà cap algorisme que no tingui cost  $\Theta(\log n)$ . No cal escriure codi però cal incloure l'anàlisi de l'algorisme que proposeu.

(1 punt) Demostració:

(1 punt) Descripció i anàlisi de l'algorisme:

## Examen Parcial EDA

Duració: 2 hores

19/3/2012

## Problema 1

(3,5 punts)

- (1) Per a les funcions  $f$  i  $g$  següents, digueu si és cert o fals que  $f = O(g)$ ,  $f = \Omega(g)$  i  $f = \Theta(g)$ .

$f(n)$	$g(n)$	$f = O(g)$	$f = \Omega(g)$	$f = \Theta(g)$
$\sqrt{n}$	$n^{2/3}$			
$\log(2n)$	$\log(3n)$			
$n^{0.1}$	$(\log n)^{10}$			
$2^n$	$2^{n+1}$			
$100n + \log n$	$n + (\log n)^2$			

- (0,5) Donat un vector de  $n$  elements possiblement repetits, digueu com trobar, en temps  $O(n \log n)$ , tots els elements repetits.
- (1,5) Volem convertir a binari  $10^n$  (un u seguit de  $n$  zeros), on  $n$  és una potència estrictament positiva de 2.

Completeu l'algorisme següent:

```
string pwr2bin(int n) {
    if (n == 1) return "1010"; // 10 en binari (8 + 2)
    string z = 
    return Karatsuba(z,z);
}
```

Doneu una recurrència que descrigui el cost temporal de l'algorisme anterior i resoleu-la.

- (0,5) El professor X us diu a classe que és asimptòticament més lent elevar al quadrat un enter de  $n$  bits que multiplicar dos enters de  $n$  bits. L'heu de creure? Justifiqueu la vostra resposta.

## Problema 2

(1,5 punts)

Considereu el codi següent:

```
void f(vector<int>& v, int i, int j) {
    if ((j-i+1)>1) {
        int k = (i+j)/2;
        f(v, i, k);
        if (k%2 == 0) f(v, k+1, j);
        g(v, i, k, j);
    }
}

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
```

```

    for (int i=0; i<n; ++i) cin >> a[i];
    f(a, 0, n-1);
}

```

La funció  $g$  no crida a la funció  $f$ .

Digueu:

- Com a màxim, quantes crides recursives a la funció  $f$  poden haver a la pila de recursió en un moment donat?

a) 2   b)  $\Theta(n)$    c)  $\Theta(\log n)$    d)  $\Theta(1)$

- En total, en el cas pitjor, quantes crides es fan a la funció  $f$ ? Escolliu l'opció més precisa.

a)  $\Theta(n\sqrt{n})$    b)  $O(n)$    c)  $O(n^2)$    d)  $\Omega(\log n)$

### Problema 3

(1,5 punts)

Considereu el codi següent:

//  $e$  i  $d$  delimiten un interval tancat (potser buit) dins del vector  $V$

```

void misteri (int e, int d, vector<int>& V) {
    if (e < d) {
        barreja (e, d, V); // barreja a l'atzar els elements de V[e..d],
                           // amb cost proporcional a d - e + 1
        misteri (e, d - 1, V);
        int i = e;
        while (V[i] < V[d]) ++i;
        intercanvia (V[i], V[d]); // cost constant
        misteri (i + 1, d, V);
    }
}

```

Digueu, justificadament, què fa l'acció `misteri`, i quin cost té en el cas pitjor i millor.

### Problema 4

(1 punt)

Siguin  $f, g : \mathbb{N} \rightarrow \mathbb{R}^{\geq 1}$  dues funcions tals que  $\lim_{n \rightarrow \infty} g(n) = \infty$ , on  $\mathbb{R}^{\geq 1}$  representa el conjunt dels nombres reals més grans o iguals que 1.

- És cert que  $f \in O(g)$  implica  $\log f \in O(\log g)$ ? Justifiqueu la resposta o doneu un contraexemple.
- És cert que  $f \in O(g)$  implica  $2^f \in O(2^g)$ ? Justifiqueu la resposta o doneu un contraexemple.

**Problema 5****(2,5 punts)**

Siguin  $A = \langle a_1, a_2, \dots, a_n \rangle$  i  $B = \langle b_1, b_2, \dots, b_m \rangle$  dues seqüències amb  $n \geq m$ . Es diu que  $B$  és una subseqüència de  $A$ , i s'escriu  $B \subseteq A$ , si existeixen  $m$  índexs en  $A$ ,  $i_1 < i_2 < \dots < i_m$ , tals que  $\forall j: 1 \leq j \leq m: b_j = a_{i_j}$ . Per exemple, si  $A = abcaddaa$  i  $B = bcda$ , llavors  $B \subseteq A$ , però si  $B = cbda$ , aleshores  $B \not\subseteq A$ .

Suposeu que disposeu d'un algorisme `is_subseq`, de cost  $O(n + m)$ , que determina si  $B$  és o no subseqüència de  $A$ .

Definim  $B^i$ , amb  $i \geq 0$ , com la seqüència que s'obté prenent un per un  $i$  de forma consecutiva els elements de  $B$  i repetint-los  $i$  vegades. Per exemple, si  $B = abbc$ ,  $B^0 = \lambda$  (seqüència buida),  $B^1 = abbc$ ,  $B^2 = aabbbcc$ ,  $B^3 = aaabbbbbbccc$ , etc.

És fàcil mostrar que si  $B^i \subseteq A$ , aleshores  $\forall j: 0 \leq j \leq i: B^j \subseteq A$ .

Proposeu un algorisme de dividir i vèncer per trobar, en temps  $O(n \log n)$ , el màxim valor de  $i$  tal que  $B^i \subseteq A$ . Justifiqueu tant la correctesa com el cost de la vostra solució.

Noteu que si  $B$  no és una subseqüència de  $A$ , llavors  $i$  serà zero, i que el valor més gran possible per a  $i$  és  $n/m$ , perquè per tal que  $B^i$  sigui subseqüència de  $A$ , la longitud de  $B^i$  ha de ser menor o igual a la de  $A$ .

## Examen Parcial EDA

Duració: 2 hores

22/10/2012

Aquestes identitats us poden ser útils per aquest examen:

- $\sum_{i=1}^n i = n(n+1)/2$
- $\sum_{i=1}^n i^2 = n(n+1)(2n+1)/6$
- $\sum_{i=1}^n i^3 = (n(n+1)/2)^2$
- $\sum_{i=1}^n c^i = \Theta(c^n)$  per a  $c > 1$ .

## Problema 1

(2,5 punts)

- Sabent que  $T_1 = O(f)$  i que  $T_2 = O(f)$ , digueu si les afirmacions següents són certes o falses.

– (0,5 punts)  $T_1 + T_2 = O(f)$ .

– (0,5 punts)  $T_1 - T_2 = O(f)$ .

– (0,5 punts)  $T_1 / T_2 = O(f)$ .

– (0,5 punts)  $T_1 = O(T_2)$ .

- (0,5 punts) Resoleu la recurrència:

$$T(n) = \begin{cases} 1 & \text{si } n \leq 4 \\ 2T(\frac{n}{4}) + \sqrt{n} & \text{si } n > 4 \end{cases}$$

## Problema 2

(1 punt)

Demostreu que el nombre de bits de  $n!$  és  $\Theta(n \log n)$ .

## Problema 3

(1,5 punts)

```
int misteri (int n) {
    int r = 0;
    for (int i = 1; i ≤ n; ++i)
        for (int j = i+1; j ≤ n; ++j)
            for (int k = j+1; k ≤ n; ++k)
                ++r;
    return r;
}
```

1. (1 punt) Assumint  $n \geq 3$ , aquesta funció calcula un polinomi  $p(n) = a_0 + a_1n + a_2n^2 + \dots + a_d n^d$ . Quin? Doneu-ne el grau  $d$  i tots els coeficients exactes  $a_0, \dots, a_d$ .
2. (0,5 punts) Doneu una fita ajustada (amb notació  $\Theta$ ) al cost de la funció *misteri*.

**Problema 4****(2 punts)**

Considereu el codi següent:

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

void misteri_b(int n, vector<double>& V) {
    for (int i = 0; i < int(V.size()); ++i) V[i] = misteri_a(n, V[i]);
}
```

Digueu, justificadament, què fan *misteri\_a* i *misteri\_b*, i quin cost tenen, en funció de  $n$  i  $m = V.size()$ , en els casos pitjor, mitjà i millor.

**Problema 5****(1 punt)**

Proposeu (sense implementar) un algorisme eficient per calcular  $m^m$ , on  $m$  és un natural llarg. Recordeu que es poden fer servir algorismes explicats a classe. Doneu el cost del vostre algorisme en funció de la talla de l'entrada  $n$ , és a dir,  $n = \log_2 m$ .

**Problema 6****(2 punts)**

Es diu que una funció  $f: [a, b] \rightarrow \mathbb{R}$  és:

- *estrictament creixent* quan per a tot  $x, y \in [a, b]$ ,  $x < y$  implica  $f(x) < f(y)$ ;
- *estrictament decreixent* quan per a tot  $x, y \in [a, b]$ ,  $x < y$  implica  $f(x) > f(y)$ ;
- *estrictament monòtona* quan és estrictament creixent o decreixent.

Sigui  $f: [a, b] \rightarrow \mathbb{R}$  una funció contínua i estrictament monòtona que satisfà que  $f(a) \cdot f(b) < 0$ . És sabut que aleshores existeix un únic  $z \in (a, b)$ , anomenat el *zero* de la funció, tal que  $f(z) = 0$ .

Assumint que  $f$  és una funció contínua i estrictament monòtona  $f: [a, b] \rightarrow \mathbb{R}$  tal que  $f(a) \cdot f(b) < 0$ , escriviu en C++ una funció

```
double zero (double a, double b, double tol)
```

que, donades  $a, b$  i una tolerància  $tol$  tal que  $0 < tol < b - a$ , retorni un  $\tilde{z}$  tal que el zero de  $f$  es troba en l'interval  $(\tilde{z} - \frac{tol}{2}, \tilde{z} + \frac{tol}{2})$ . Analitzeu el cost del vostre programa en el cas pitjor. Assumiu que podeu fer crides a  $f$ , i que el seu cost és  $O(1)$ . Només es consideraran com a vàlides les solucions amb cost  $O(\log(\frac{b-a}{tol}))$ .



**Examen Parcial EDA****Duració: 2 hores****18/3/2013****Problema 1****(2 punts)**

Disposem dels tres algorismes següents per resoldre un cert problema ( $n$  denota la mida de l'entrada):

- A: resol el problema dividint-lo en 5 subproblemes de mida  $n/2$ , resol recursivament cada subproblema i combina les solucions en temps  $\Theta(n)$ .
- B: resol el problema dividint-lo en 2 subproblemes de mida  $n - 1$ , resol recursivament cada subproblema i combina les solucions en temps  $\Theta(1)$ .
- C: resol el problema dividint-lo en 9 subproblemes de mida  $n/3$ , resol recursivament cada subproblema i combina les solucions en temps  $\Theta(n^2)$ .

Quin és el cost en temps de cada algorisme fent servir la notació  $\Theta$ ?

(0,5 punts) Algorisme A:

(0,5 punts) Algorisme B:

(0,5 punts) Algorisme C:

(0,5 punts) Quin dels tres algorismes escolliries? Per què?

**Problema 2****(2 punts)**

Sigui  $\pi(n)$  el nombre de nombres primers dins l'interval  $[1, \dots, n]$ . És un fet que

$$\pi(n) = \Theta\left(\frac{n}{\log n}\right).$$

Per a les preguntes següents, recordeu que els nombres *compostos* són els que no són primers, i que els nombres *quadrats* són els de la forma  $k^2$  on  $k$  és un natural.

(1 punt) Digueu si és CERT o FALS que, per a  $n$  suficientment gran, hi ha més nombres compostos que primers dins l'interval  $[1, \dots, n]$ , i justifiqueu la resposta:

(1 punt) Digueu si és CERT o FALS que, per a  $n$  suficientment gran, hi ha més nombres quadrats que primers dins l'interval  $[1, \dots, n]$ , i justifiqueu la resposta:

**Problema 3****(2 punts)**

Donat un vector  $V$  amb  $n$  nombres diferents  $V[0], \dots, V[n-1]$ , i un nombre  $s$ , es vol saber si existeixen tres índexs  $i, j$  i  $k$  entre 0 i  $n-1$  tals que  $V[i] + V[j] + V[k] = s$ . Els índexs poden estar repetits. Per exemple, si  $n = 2$ ,  $V[0] = 5$ ,  $V[1] = 6$ , i  $s = 16$ , la resposta és afirmativa. (Una solució seria  $i = j = 0$ ,  $k = 1$ .)

Considereu aquests algorismes:

- a) Provem per a tota  $i$ , tota  $j$  i tota  $k$  entre 0 i  $n-1$ , si  $V[i] + V[j] + V[k] = s$ .
- b) Provem per a tota  $i$  entre 0 i  $n-1$ , tota  $j$  entre  $i$  i  $n-1$  i tota  $k$  entre  $j$  i  $n-1$ , si  $V[i] + V[j] + V[k] = s$ .

- c) Construïm un vector  $Q$  de mida  $n^2$  amb totes les sumes  $V[i] + V[j]$ , amb  $i$  i  $j$  entre 0 i  $n - 1$ . Ordenem  $Q$ . Busquem, per a tota  $k$  entre 0 i  $n - 1$ , si  $s - V[k]$  es troba a  $Q$ .
- d) Construïm un vector  $Q$  de mida  $n^2$  amb totes les sumes  $V[i] + V[j]$ , amb  $i$  i  $j$  entre 0 i  $n - 1$ . Ordenem  $V$ . Busquem, per a tota  $\ell$  entre 0 i  $n^2 - 1$ , si  $s - Q[\ell]$  es troba a  $V$ .

Digueu raonadament el cost en el cas pitjor de cadascun d'aquests algorismes. (0,5 punts cadascun).

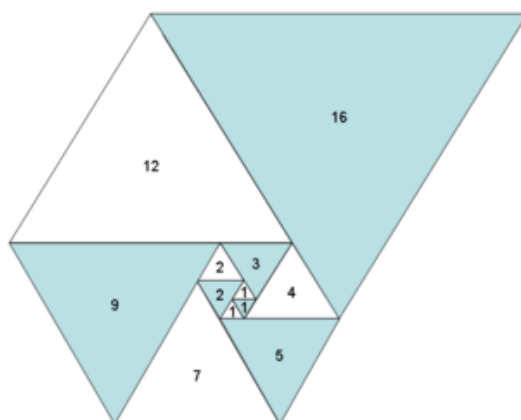
#### Problema 4

(2 punts)

La *successió de Padovan* és una successió de nombres naturals definida pels valors inicials  $P(0) = P(1) = P(2) = 1$  i la relació de recurrència

$$P(n) = P(n-2) + P(n-3)$$

per a  $n \geq 3$ . Entre altres llocs, aquesta successió apareix, gràficament, aquí:



1. Implementa en C++ una funció `int g1(int n)` que calculi el  $n$ -èsim nombre de Padovan. La funció ha de tenir cost en temps  $O(n)$ .
2. Es pot comprovar que

$$\begin{pmatrix} P(n) \\ P(n-1) \\ P(n-2) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

per  $n \geq 2$ . Fent ús d'aquest fet (que no us demanem demostrar), i d'un tipus `matrix<int>` amb constructor `matrix<int>(files, columnes, valor)` i amb operació de multiplicació  $m1 * m2$ , completa la següent implementació de `int g2(int n)` que també calcula  $P(n)$ , aquest cop en temps  $O(\log n)$ .

```
void misteri (const matrix<int>& m, int k, matrix<int>& p) {
    if (  )
        for (int i = 0; i < 3; ++i) p[i][i] = 1;
    else {
        misteri (  );
        p =  *  ;
    }
}
```

```

        if (k % 2 == 1)  ;
    } }

int g2(int n) {
    if (n ≤ 2) return 1;
    else {
        matrix<int> m(3, 3, 0);
        m[0][1] = m[0][2] = m[1][0] = m[2][1] = 1;
        matrix<int> p(3, 3, 0);
        mister_i(m, n-2, p);
        return p[0][0] + p[0][1] + p[0][2];
    } }

```

L'enunciat d'aquest problema està inspirat en la pàgina: [http://en.wikipedia.org/wiki/Padovan\\_sequence](http://en.wikipedia.org/wiki/Padovan_sequence)

### Problema 5

(2 punts)

Suposeu donat un tipus `vector_inf<int>` que implementa vectors de mida  $\infty$  en què les  $n$  primeres posicions (de la 0 a la  $n - 1$ ) contenen valors enters (i per tant finits) no necessàriament ordenats, i que les posicions restants (a partir de la  $n$ ) contenen el valor  $\infty$ . Doneu, en C++, un algorisme

```
int busca( vector_inf<int>& V)
```

que, donat un vector  $V$  com aquest amb  $n \geq 1$ , determini  $n$  en temps  $O(\log n)$ .

Suposeu que el tipus `vector_inf<int>` està implementat de tal manera que podeu accedir a les seves posicions de la mateixa manera que amb els vectors ordinaris en temps constant. També podeu fer comparacions  $v[i] == \infty$  en temps constant.

Pista: comenceu per trobar un  $\infty$  ràpidament.

## Examen Parcial EDA

Duració: 2 hores

21/10/2013

## Problema 1

(2 punts)

- Sigui  $n$  parell. El cost de l'algorisme d'ordenació per inserció amb l'entrada  $[2, 1, 4, 3, 6, 5, \dots, 2i, 2i - 1, \dots, n, n - 1]$  és  $T(n) = \Theta(\quad)$ .
- Indiqueu **totes** les afirmacions que siguin correctes: El cost de mergesort en el cas pitjor és: ☐  $O(n^2)$ ; ☐  $\Theta(n \log n)$ ; ☐  $\Omega(n \log n)$ .
- Doneu una funció  $f(n)$  que sigui alhora  $\Omega(n)$  i  $O(n \log n)$ , però que no sigui  $\Theta$  de cap de les dues. Resposta:  $f(n) = \quad$
- Resoleu  $T(n) = 3T(n/3) + \Theta(n^2)$ . Resposta:  $T(n) = \Theta(\quad)$ .
- La **recurrència** que expressa el cost de l'algorisme de Karatsuba per multiplicar dos nombres de  $n$  dígitos és  $T(n) = \quad$ .
- Sigui  $x$  un vector de  $n$  bits qualssevol. Interpretant  $x$  com un nombre natural escrit en binari, el cost en cas pitjor de l'algorisme obvi per sumar 1 a  $x$  és  $T(n) = \Theta(\quad)$ .
- Sigui  $x$  un vector de  $n$  bits escollits uniformement i independentment a l'atzar. Interpretant  $x$  com un nombre natural escrit en binari, el cost esperat de l'algorisme obvi per sumar 1 a  $x$  és  $T(n) = \Theta(\quad)$ .
- Considereu l'algorisme següent de cost  $\Theta(\log n)$  on  $n$  és la mida del vector:
 

```

int dicotomica(vector<int>& v, int e, int d, int x) {
    if (e > d) return -1;
    int m = (d + e)/2;
    if (v[m] < x) return dicotomica(v, m + 1, d, x);
    if (v[m] > x) return dicotomica(v, e, m - 1, x);
    return m;
}

```

Quin cost tindria aquest mateix algorisme si tinguéssim la mala pata d'oblidar-nos l'& en el pas de paràmetres? Resposta:  $T(n) = \Theta(\quad)$ .

## Problema 2

(2 punts)

Donat un vector  $A$  amb  $n > 0$  nombres, considerem dos algorismes per calcular els elements mínim ( $\min$ ) i màxim ( $\max$ ) simultàniament:

1. **Seqüencial.** Les variables  $\min$  i  $\max$  prenen inicialment el valor  $A[0]$  i es van actualitzant en un recorregut seqüencial del vector.
2. **Dividir i vèncer.** Si el vector només té un o dos elements, es calculen  $\min$  i  $\max$  de la manera òbvia fent una sola comparació. En cas que  $n > 2$ , es divideix el vector en dues meitats i, per a cadascuna, es fa una crida recursiva i s'assigna a  $\min$  el més petit dels mínims fent una comparació i a  $\max$  el més gran dels màxims fent una altra comparació.

Justifiqueu les afirmacions següents:

(1 punt) El cost asimptòtic de tots dos algorismes és el mateix.

(1 punt) Si comptem el nombre **exacte** de comparacions entre elements del vector que fa cada algorisme, el del segon és millor que el del primer. Per simplificar, suposeu que  $n$  és una potència de 2. (Pista: per analitzar el segon algorisme, dibuixeu l'arbre de recursió per  $n = 8$  i recordeu que  $\sum_{i=0}^{k-1} 2^i = 2^k - 1$ .)

### Problema 3

(2 punts)

(1 punt) Proposeu un algorisme lineal que, donat un vector  $V[1, \dots, n]$  que conté una permutació qualsevol dels nombres de 1 a  $n$ , retorni un vector  $D[1, \dots, n-1]$  on  $D[i]$  és la distància a la que es troben els nombres  $i$  i  $i+1$  en el vector  $V$ . Per exemple, si  $V = [5, 2, 1, 4, 6, 3]$ , llavors  $D = [1, 4, 2, 3, 4]$ . No es valorarà cap solució que no sigui de cost  $\Theta(n)$ .

(1 punt) Ara suposeu que el vector  $V[1, \dots, n]$  conté nombres diferents qualssevol (no necessàriament entre 1 i  $n$ ) i voleu retornar un vector  $D[1, \dots, n-1]$  on  $D[i]$  és la distància a la que es troben l' $i$ -èssim nombre de  $V$  i l' $(i+1)$ -èssim nombre de  $V$ , enumerats de petit a gran. Per exemple, si  $V = [1231, -2, 453456, -23434]$ , llavors  $D = [2, 1, 2]$ . Descriviu un algorisme de cost  $\Theta(n \log n)$  per fer això. (Pista: penseu en una modificació del mergesort.)

### Problema 4

(2 punts)

Considereu el codi següent:

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

void misteri_b(int n, vector<double>& V) {
    for (int i = 0; i < int(V.size()); ++i) V[i] = misteri_a(n, V[i]);
}
```

Digueu, justificadament, què fan `misteri_a` i `misteri_b`, i quin cost tenen, en funció de  $n$  i  $m = V.size()$ , en els casos pitjor, mitjà i millor.

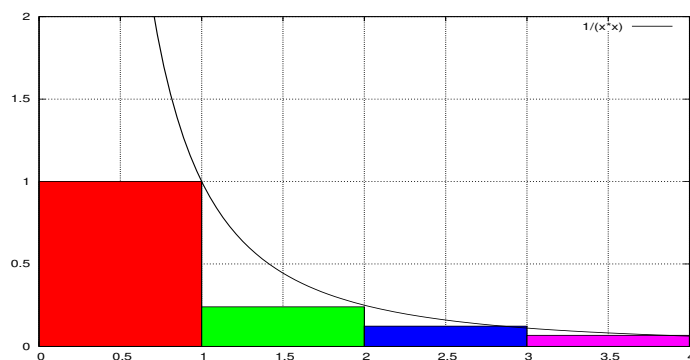
### Problema 5

(1 punt)

Sigui

$$S(n) = \sum_{i=1}^n \frac{1}{i^2}.$$

Inspirats en el gràfic adjunt, demostreu que  $S(n) = \Theta(1)$ . (Pista: àrea = integral.)

**Problema 6****(1 punt)**

Demostreu o doneu un contraexemple a l'enunciat següent: Per qualsevol funció  $f(n)$  i qualsevol constant  $c > 0$  tenim que  $f(cn) = \Theta(f(n))$ .

## Examen Parcial EDA

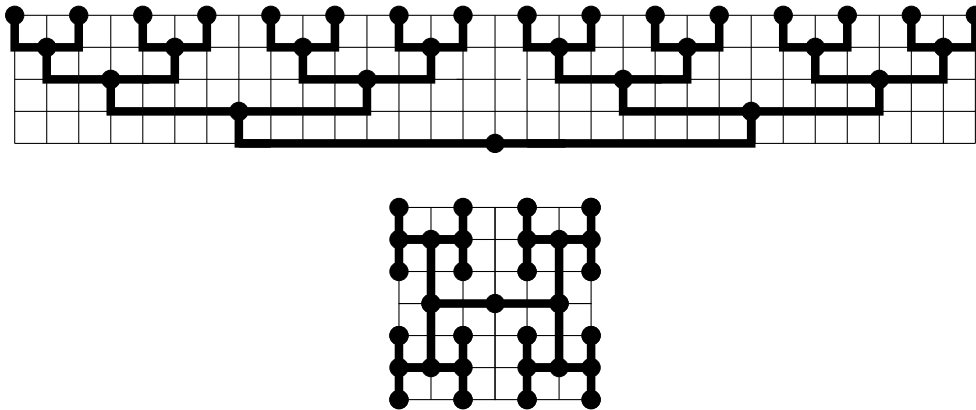
Duració: 2 hores

24/3/2014

## Problema 1

(3 punts)

Com a grafs, tots els arbres són planars. En particular això vol dir que tots es poden representar en una graella rectangular prou gran de manera que les arestes no s'intersequin les unes amb les altres (excepte, òbviament, en els vèrtexs). Aquí teniu dues representacions planars d'un arbre binari complet de 16 fulles:



(1 punt) Seguint el patró recursiu de la **primera** representació, doneu les recurrències que expressen l'alçada  $H(n)$  i l'amplada  $W(n)$ , respectivament, de la graella que es necessita per representar un arbre binari complet de  $2^n$  fulles en aquesta representació.

(1 punt) Seguint el patró recursiu de la **segona** representació, doneu la recurrència que expressa la longitud  $L(n)$  de la graella quadrada que es necessita per representar un arbre binari complet de  $4^n$  fulles en aquesta construcció.

(1 punt) Si el vostre objectiu és minimitzar l'àrea de la graella per un arbre binari complet de  $4^n$  fulles, expliqueu quina de les dues construccions escollireu (per relacionar les dues construccions noteu que  $4^n = 2^{2n}$ ). No cal que resolguis les recurrències de forma exacta; una estimació asimptòtica és suficient.

## Problema 2

(2 punts)

Considereu els quatre codis a continuació:

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

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

```
void f3 (int n) {
    int x = 2;
    int y = 1;
```

```
void f4 (int n) {
    int x = 2;
    int y = 1;
```

<pre> while (y ≤ n) {     y = y*x;     x = 2*x; } </pre>	<pre> while (y ≤ n) {     y = y*x;     x = x*x; } </pre>
--	--

Quin cost té  $f_1$ ? Resposta:  $\Theta(\quad)$

Quin cost té  $f_2$ ? Resposta:  $\Theta(\quad)$

Quin cost té  $f_3$ ? Resposta:  $\Theta(\quad)$

Quin cost té  $f_4$ ? Resposta:  $\Theta(\quad)$

Una resposta correcta sense una justificació adequada **no rebrà cap punt**.

### Problema 3

(3 punts)

Una *inversió* en un vector d'enters  $T[0 \dots n-1]$  és un parell de posicions del vector en desordre, és a dir, un parell  $(i, j)$  amb  $0 \leq i < j < n$  tal que  $T[i] > T[j]$ .

(1,5 punts) Sigui  $T[0 \dots n-1]$  un vector d'enters. Demostreu que si en  $T$  hi ha una inversió  $(i, j)$ , aleshores  $T$  té almenys  $j - i$  inversions.

(1,5 punts) Fixem un valor  $N$  tal que  $N \geq 0$  (és a dir,  $N$  en aquest exercici és una constant). Usant Divideix i Venceràs, implementeu en C++ una funció

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

que, donats un enter  $x$  i un vector d'enters  $T$  de mida  $n$  amb com a molt  $N$  inversions, digui si  $x$  és a  $T$ . La funció ha de tenir cost en el cas pitjor  $\Theta(\log n)$ . Demostreu que té el cost desitjat.

**Pista:** per respondre l'apartat 2, useu l'apartat 1. A més, considereu com a cas base el tractament de subvectors de mida  $\leq 2N$ , i com a cas recursiu el complementari.

### Problema 4

(2 punts)

Quatre preguntes curtes:

1. Sigui  $f(n) = n \log(\cos(n\pi) + 4)$ . Llavors  $f(n) = \Theta(\quad)$ .
2. Quin és l'últim dígit de  $7^{1024}$  escrit en decimal? Resposta:  $\quad$ .
3. Quina és la **recurrència** que expressa el cost de l'algorisme de Strassen per multiplicar matrius  $n \times n$ . Resposta  $T(n) = \quad$ .
4. Un algorisme pot rebre els  $2^n$  vectors de  $n$  bits com entrada. En  $2^n - 1$  d'aquestes entrades el cost de l'algorisme és  $\Theta(n^2)$  i en l'entrada restant el seu cost és  $\Theta(n^4)$ . Per



tant, el cost en el cas pitjor és  $\Theta(n^4)$  i el cost en el cas millor és  $\Theta(n^2)$ . Quin és el cost en el cas mitjà quan l'entrada s'escull uniformement a l'atzar (i per tant cada entrada té probabilitat  $1/2^n$ )?

Resposta:  $T(n) = \Theta( \text{ } )$ .

Justificacions per les preguntes 1, 2 i 4 (per la pregunta 3 **no** cal justificació):

**Examen Parcial EDA****Duració: 2.5 hores****13/10/2014****Problema 1****(2 punts)**

(1 punt) Demostreu que  $\sum_{i=0}^n i^3 = \Theta(n^4)$ .

Ajut: Hi ha diverses maneres de fer-ho, una d'elles demostrant l' $O$  i l' $\Omega$  per separat.

(1 punt) Sigui  $f(n) = 2\sqrt{\log n}$  i  $g(n) = n$ . Asimptòticament, quina creix més ràpid? Demostreu-ho.

**Problema 2****(3 punts)**

Considerem el procediment *misteri* que rep com entrada un vector  $A$  i un enter positiu  $k$  de manera que tots els elements de  $A$  estan entre 0 i  $k$ , ambdós inclosos.

```
void misteri (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 punt) Què conté  $B$  al finalitzar l'execució de *misteri*?

(b) (1 punt) Si  $n$  és la longitud del vector  $A$  i us assegurin que  $k = O(n)$ , quin és el cost asimptòtic de *misteri* en funció de  $n$ ?

(c) (1 punt) Sense escriure codi, descriu un algorisme per al problema següent: donat un enter positiu  $k$ , un vector  $A$  de  $n$  enters entre 0 i  $k$ , ambdós inclosos, i un vector  $P$  de  $m$  parells d'enters  $(a_0, b_0), \dots, (a_{m-1}, b_{m-1})$ , amb  $0 \leq a_i \leq b_i \leq k$  per cada  $i = 0, \dots, m-1$ , cal escriure el nombre d'elements de  $A$  que es troben en l'interval  $[a_i, b_i]$  per  $i = 0, \dots, m-1$  en temps  $\Theta(m + n + k)$ , i en particular temps  $\Theta(m + n)$  quan  $k = O(n)$ .

**Problema 3****(3 punts)**

Un joc de taula d'atzar té 64 posicions possibles  $1, \dots, 64$ . Les regles del joc són tals que, des de cada posició  $i \in \{1, \dots, 64\}$ , en una tirada podem anar a parar a qualsevol altra posició  $j \in \{1, \dots, 64\}$  amb probabilitat  $P_{i,j}$ . Sigui  $P = (P_{i,j})_{1 \leq i,j \leq 64}$  la matriu de probabilitats.

(a) (1 punt) Recordeu que si  $R = P^2$ , llavors  $R_{i,j} = \sum_{k=1}^{64} P_{i,k} P_{k,j}$  per cada  $i, j$ . Sabent que, com diu l'enunciat,  $P_{i,j}$  és la probabilitat d'anar a parar a  $j$  des de  $i$  en una tirada, com interpreteu  $R_{i,j}$ ?

(b) (2 punts) Dissenyeu un algorisme que, donada la matriu de probabilitats  $P = (P_{i,j})_{1 \leq i,j \leq 64}$  i donat un nombre de moviments  $t \geq 0$ , calculi la matriu  $(Q_{i,j})_{1 \leq i,j \leq 64}$  on cada  $Q_{i,j}$  és la probabilitat que el joc acabi a la posició  $j$  després de jugar exactament  $t$  tirades, començant des de la posició  $i$ .

```
typedef vector<double> Fila;
typedef vector<Fila> Matriu;
```

```
void probabilitats (const Matriu & P, int t, Matriu & Q)
```

**Nota 1:** Per obtenir la nota màxima, el vostre algorisme ha de tenir cost  $\Theta(\log t)$ .

**Nota 2:** Si us cal alguna funció auxiliar, implementeu-la al darrera.

**Nota 3:** Justifiqueu el cost al darrera.

#### Problema 4

(2 punts)

(a) (1 punt) Supposem que un algorisme pot rebre qualsevol dels  $2^n$  vectors de  $n$  bits amb igual probabilitat. Supposem que el cost de l'algorisme en el cas pitjor és  $\Theta(n^2)$  i que el cost en el cas millor és  $\Theta(n)$ . Quantes entrades cal que provoquin cost  $\Omega(n^2)$  per estar segurs que l'algorisme tindrà cost  $\Theta(n^2)$  en el cas mitjà?

(b) (0.5 punts) Considereu el següent algorisme per sumar una unitat a un nombre natural representat per un vector `vector<int> A` de  $n$  dígit decimal:

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

Si cada dígit  $A[i]$  de l'entrada és equiprobable i independent de la resta (és a dir, cada  $A[i]$  és un dels 10 dígit possibles amb probabilitat  $1/10$  sense tenir en compte la resta), quina és la probabilitat que aquest algorisme faci exactament  $k$  iteracions quan  $0 \leq k \leq n - 1$ ?

(c) (0.5 punts) Apliqueu el teorema mestre de recurrències substractores per resoldre la recurrència  $T(n) = \frac{1}{10}T(n-1) + \Theta(1)$  amb el cas base  $T(0) = \Theta(1)$ .

————— FI DE L'EXAMEN —————

(d) (extra bonus **opcional**: 1 punt addicional a la nota) A què correspon la recurrència  $T(n)$  de l'apartat (c) en relació a l'algorisme de l'apartat (b)?

## Examen Parcial EDA

Duració: 2h30m

23/3/2015

## Problema 1: Anàlisi de costos

(2 punts)

El garbell d'Eratòstenes (276–194 aC) és un mètode per generar tots els nombres primers més petits o iguals que un  $n$  donat. El mètode fa així: recorrent la seqüència dels nombres  $2, 3, 4, \dots, n$ , busca el següent nombre  $x$  que no estigui marcat, marca tots els seus múltiples  $2x, 3x, 4x, \dots$  més petits o iguals que  $n$ , i torna a començar. Quan acaba, els  $x \geq 2$  que no estan marcats són els nombres primers. En 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;
    }
}
```

Per a les tres primeres preguntes es demana una expressió exacta (no asimptòtica) *en funció de  $n$* . Si us cal feu servir la notació  $\lfloor z \rfloor$  que arrodoneix  $z$  al màxim enter més petit o igual que  $z$ ; per exemple  $\lfloor \pi \rfloor = \lfloor 3.14\dots \rfloor = 3$ .

(a) (0.33 punts) Quantes vegades s'executarà  $M[y] = \text{true}$  quan  $x$  valgui 2?

Resposta: Exactament  vegades.

(b) (0.34 punts) Quantes vegades s'executarà  $M[y] = \text{true}$  quan  $x$  valgui 15?

Resposta: Exactament  vegades.

(c) (0.33 punts) Quantes vegades s'executarà  $M[y] = \text{true}$  quan  $x$  valgui 17?

Resposta: Exactament  vegades.

(d) (0.5 punts) Se sap que

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

Feu servir això i les respostes de les preguntes anteriors per calcular el cost de l'algorisme en funció de  $n$ , en notació asimptòtica. Resposta:  $\Theta(\text{  })$ . Justificació:

(e) (0.5 punts) Una millora seria substituir la condició  $x \leq n$  del bucle extern per  $x*x \leq n$ . Millora això el cost asimptòtic? Resposta i justificació:

## Problema 2: Strassen &amp; company

(2 punts)

L'algorisme escolar per multiplicar matrius  $n \times n$  fa  $\Theta(n^3)$  operacions aritmètiques. L'any 1969 Strassen va trobar un algorisme que fa  $\Theta(n^{2.81})$  operacions aritmètiques. Vint-i-un anys més tard, Coppersmith i Winograd van descobrir un mètode que fa  $\Theta(n^{2.38})$  operacions.

Suposant (per simplificar) que les constants implícites en la notació  $\Theta$  són 1, 10 i 100, respectivament, i que s'apliquen a cada  $n \geq 1$  (és a dir, que els costos són  $n^3$ ,  $10n^{2.81}$  i  $100n^{2.38}$ ,

respectivament, per a tot  $n \geq 1$ ), calculeu el mínim  $n$  a partir del qual un d'aquests algorismes fa menys operacions que un altre.

(a) (1 punt) Per a  $n \geq \boxed{\phantom{000}}$ , Strassen millora l'escolar.

(b) (1 punt) Per a  $n \geq \boxed{\phantom{000}}$ , Coppersmith-Winograd millora Strassen.

Justificacions:

**Nota:** Si no porteu calculadora (estàveu avisats!), deixeu indicada la solució en forma d'expressió aritmètica.

### Problema 3: Un de dissenyar algorismes

(3 punts)

Donada una seqüència de  $n$  intervals no buits  $[a_1, b_1], \dots, [a_n, b_n]$ , volem calcular, en temps  $O(n \log n)$ , la seva unió representada com una seqüència d'intervals disjunts ordenada segons l'extrem esquerre dels intervals. Per exemple, si els intervals de l'entrada són

$$[17, 19] \ [-3, 7] \ [4, 9] \ [18, 21] \ [-4, 15]$$

llavors la sortida ha de ser  $[-4, 15] \ [17, 21]$ .

(a) (1 punt) Descriviu un algorisme que resolgui aquest problema. Expliqueu l'algorisme en paraules, sense escriure codi, però de manera prou clara perquè es pugui implementar. Supposeu que l'entrada ve donada pels vectors  $(a_1, \dots, a_n)$  i  $(b_1, \dots, b_n)$ , amb  $a_i \leq b_i$  per a tot  $i = 1, \dots, n$ .

(b) (1 punt) En aquest apartat, a més de la seqüència de  $n$  intervals, ens donen una seqüència de  $m$  reals diferents  $p_1, \dots, p_m$  i volem determinar quants cauen en algun interval de la unió (només ens cal el número; no pas quins són). Fent servir l'algorisme de l'apartat anterior, descriviu un algorisme que resolgui aquest problema en temps  $O(n \log n)$  quan  $m = n$ . Supposeu que l'entrada ve donada pels vectors  $a$  i  $b$  de l'apartat anterior, i pel vector  $(p_1, \dots, p_m)$  amb  $p_i \neq p_j$  si  $i \neq j$ .

(c) (1 punt) Si sabéssiu que  $m$  està fitat per una constant petita i independent de  $n$ , per exemple  $m \leq 5$ , faríeu servir el mateix algorisme? Si no, quin faríeu servir? Si decidíu canviar d'algorisme, expliciteu-ne el cost.

### Problema 4: Preguntes curtes

(3 punts)

- (0,5 punts) Determineu si són iguals ( $=$ ) o diferents ( $\neq$ ) i demostreu-ho:

$$\Theta(3^{\log_2(n)}) \quad \boxed{\phantom{000}} \quad \Theta(3^{\log_4(n)}).$$

- (0,5 punts) Calculeu  $2^1 \cdot 2^2 \cdot 2^3 \cdot \dots \cdot 2^{99} \cdot 2^{100} \pmod{9}$ . Aquest problema no està pensat per fer amb calculadores.
- (1 punt) Ordeneu les funcions següents segons el seu ordre de creixement asimptòtic:  $n^4 - 3n^3 + 1, (\ln(n))^2, \sqrt{n}, n^{1/3}$ . Excepcionalment, no cal que ho justifiqueu.

- (1 punt) Considereu les tres alternatives següents per resoldre un mateix problema:
  - A: divideix una instància de talla  $n$  en cinc instàncies de talla  $n/2$ , resol recursivament cada instància, i combina les solucions en temps  $\Theta(n)$ .
  - B: donada una instància de talla  $n$ , resol recursivament dues instàncies de talla  $n - 1$  i combina les solucions en temps constant.
  - C: divideix una instància de talla  $n$  en nou instàncies de talla  $n/3$ , resol recursivament cada instància, i combina les solucions en temps  $\Theta(n^2)$ .

Escriuiu les recurrències corresponents i resoleu-les. Quina alternativa és la més eficient?

## Examen Parcial EDA

Duració: 2.5 hores

19/10/2015

## Problema 1

(3.5 punts)

Els nombres de Fibonacci estan definits per la recurrència  $f_k = f_{k-1} + f_{k-2}$  per a  $k \geq 2$ , amb  $f_0 = 0$  i  $f_1 = 1$ . Responen els següents apartats:

- (a) (0.5 punts) Considereu la següent funció *fib1* que donat un enter no negatiu  $k$  retorna  $f_k$ :

```
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];  
}
```

Descriviu de la manera més precisa possible el cost asimptòtic *en temps i en espai* de *fib1* ( $k$ ) en funció de  $k$ .

- (b) (1 punt) Demostreu que, per a  $k \geq 2$ , se satisfà la identitat matricial següent:

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

- (c) (1 punt) Completeu els blancs del codi a continuació per tal que la funció *fib2* (*k*) calculi  $f_k$ , donat un  $k \geq 0$ .

```
typedef vector<vector<int>> matrix;

matrix mult(const matrix& A, const matrix& B) {
    // Pre: A i B són matrius quadrades de les mateixes 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 misteri (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 = misteri (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 = misterí (M, k-1);
    return  ;
}

```

- (d) (1 punt) Descriu de la manera més precisa possible el cost asimptòtic en temps de  $fib2(k)$  en funció de  $k$ .

## Problema 2

(3.25 punts)

Donats un vector d'enters  $v$  i un enter  $x$ , la funció

```

int posicio (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;
}

```

examina les  $n = v.size()$  posicions de  $v$  i retorna la primera que conté  $x$ , o  $-1$  si no n'hi ha cap.

- (a) (0.75 punts) Descriviu en funció de  $n$  el cost asimptòtic en temps de *posicio* en el cas millor de la forma més precisa possible. Quan es pot donar aquest cas millor?

- (b) (0.75 punts) Descriviu en funció de  $n$  el cost asimptòtic en temps de *posicio* en el cas pitjor de la forma més precisa possible. Quan es pot donar aquest cas pitjor?

- (c) (0.75 punts) Demostreu que per a tot enter  $n \geq 1$ , es compleix:

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



- (d) (1 punt) Suposem que tenim una distribució de probabilitat sobre els paràmetres d'entrada. Concretament, la probabilitat que  $x$  sigui l'element  $v[i]$  és  $\frac{1}{2^{i+1}}$  per a  $0 \leq i < n - 1$ , i que sigui l'element  $v[n - 1]$  és  $\frac{1}{2^{n-1}}$ . En particular, aquestes probabilitats sumen 1, de manera que  $x$  sempre és un dels  $n$  valors de  $v$ .

Descriviu en funció de  $n$  el cost asimptòtic en temps de *posicio* en el cas mig de la forma més precisa possible.

**Problema 3****(3.25 punts)**

En aquest exercici abordarem el problema de, donats dos enters positius  $a$  i  $b$ , calcular el seu màxim comú divisor  $\gcd(a, b)$ . Recordeu que  $\gcd(a, b)$  és, per definició, l'únic enter positiu  $g$  tal que:

1.  $g \mid a$  ( $g$  divideix  $a$ ),
2.  $g \mid b$ ,
3. si  $d \mid a$  i  $d \mid b$ , llavors  $d \mid g$ .

(a) (1.25 punts) Demostreu que les identitats següents són certes:

$$\gcd(a, b) = \begin{cases} 2\gcd(a/2, b/2) & \text{si } a, b \text{ són parells} \\ \gcd(a, b/2) & \text{si } a \text{ és senar i } b \text{ és parell} \\ \gcd((a-b)/2, b) & \text{si } a, b \text{ són senars i } a > b \end{cases}$$

*Pista:* podeu fer servir que donats dos enters positius  $a$  i  $b$  tals que  $a > b$ , es té  $\gcd(a, b) = \gcd(a-b, b)$ .

- (b) (1 punt) Escriuiu una funció `int gcd(int a, int b)` en C++ que, usant `divideix` i `vençeràs` i l'apartat (a), calculi el màxim comú divisor  $\gcd(a, b)$  de dos enters positius  $a, b$  donats.

*Pista:* podeu fer servir també que per tot enter positiu  $a$ ,  $\gcd(a, a) = a$ .

- (c) (1 punt) Suposant que  $a$  i  $b$  són enters positius representats cadascun amb un vector de  $n$  bits, descriu de la manera més precisa possible el cost en temps en el cas pitjor de  $\gcd(a, b)$  en funció de  $n$ . Quan es pot donar aquest cas pitjor?

Assumiu que el cost de les següents operacions amb enters de  $n$  bits: sumar, restar, comparar, multiplicar/dividir per 2 és  $\Theta(n)$ , i que calcular el residu mòdul 2 triga temps  $\Theta(1)$ .

## Examen Parcial EDA

Duració: 2.5 hores

31/03/2016

## Problema 1

(1 punt)

Responen les següents qüestions. En aquest exercici, **no** cal justificar les respostes.

1. (0.2 punts) El cost de l'algorisme d'Strassen per multiplicar dues matrius de mida  $n \times n$  és  $\Theta(\text{ } )$

2. (0.6 punts) El teorema mestre de recurrències substractores diu que donada una recurrència  $T(n) = aT(n - c) + \Theta(n^k)$  amb  $a, c > 0$  i  $k \geq 0$  llavors:

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

3. (0.2 punts) L'algorisme d'ordenació mergesort usa  $\Theta(\text{ } )$  espai auxiliar per ordenar un vector de mida  $n$ .

## Problema 2

(3 punts)

En aquest problema només es consideren els costos *en temps*. Considereu el següent programa:

```
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 punts) Responen: si tant el cost de  $f(m)$  com el de  $g(m)$  és  $\Theta(m)$ , llavors el cost de  $h(n)$  en funció de  $n$  és  $\Theta(\text{ } )$

Justificació:

(b) (1.5 punts) Responen: si el cost de  $f(m)$  és  $\Theta(1)$  i el cost de  $g(m)$  és  $\Theta(m^2)$ , llavors el cost de  $h(n)$  en funció de  $n$  és  $\Theta( \text{ } )$

Justificació:



**Problema 3****(3 punts)**

Es diu que una matriu quadrada  $M$  de nombres enters de mida  $n \times n$  és *simètrica* si  $M_{i,j} = M_{j,i}$  per qualsevol  $i, j$  tals que  $0 \leq i, j < n$ .

Considerem la següent implementació de matrius simètriques amb vectors unidimensionals, que només guarda el “triangle inferior” per minimitzar l’espai consumit:

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

$$\Downarrow$$

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

Per exemple, la matriu simètrica  $3 \times 3$

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

s'implementaria amb el vector unidimensional

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

- (a) (0.5 punts) Responen: el cost en espai d'aquesta representació per a una matriu simètrica  $n \times n$  en funció de  $n$  és  $\Theta(\text{ })$ .

Justificació:

- (b) (1 punt) Donats  $n > 0$  i un valor  $k$ , la funció

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

retorna el parell  $(i, j)$  on  $i$  és la fila i  $j$  és la columna del coeficient apuntat per  $k$  en un vector unidimensional que implementa una matriu simètrica  $n \times n$ . Si  $k$  no és un índex vàlid, retorna  $(-1, -1)$ . Per exemple, `fila_columna(3,2)` retorna  $(1,1)$ , `fila_columna(3,3)` retorna  $(2,0)$ , i `fila_columna(3,6)` retorna  $(-1,-1)$ .

Completeu la següent implementació de `fila_columna`:

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

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

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

- (c) (1 punt) Analitzeu el cost en temps en el cas pitjor de la implementació de `fila_columna(int n, int k)` de l'apartat anterior en funció de  $n$ .

- (d) (0.5 punts) Usant les funcions **double** *sqrt*(**double** *x*) i **int** *floor* (**double** *x*) que respectivament calculen  $\sqrt{x}$  i  $\lfloor x \rfloor$  en temps  $\Theta(1)$ , doneu una implementació alternativa de *fila\_columna* amb cost  $\Theta(1)$ .

**Problema 4****(3 punts)**

Assumiu que  $n$  és una potència de 2, és a dir, de la forma  $2^k$  per a un cert  $k \geq 0$ .

Una matriu quadrada  $A$  de nombres reals de mida  $n \times n$  és una *matriu de Monge* si per qualssevol  $i, j, k$  i  $l$  tals que  $0 \leq i < k < n$  i  $0 \leq j < l < n$ , es compleix que

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

En altres paraules, sempre que agafem dues files i dues columnes d'una matriu de Monge i considerem els quatre elements a les interseccions de les files i les columnes, la suma dels elements de les cantonades superior esquerra i inferior dreta és més petita o igual que la suma dels elements de les cantonades superior dreta i inferior esquerra. Per exemple, la següent matriu és de Monge:

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

- (a) (1 punt) Sigui  $f_A(i)$  l'índex de la columna on apareix l'element mínim de la fila  $i$ -èsima (desempatant si cal agafant el de la columna de més a l'esquerra). Per exemple, a la matriu de dalt  $f_A(0) = 0$ ,  $f_A(1) = 0$ ,  $f_A(2) = 2$  i  $f_A(3) = 2$ .

Demostreu que  $f_A(0) \leq f_A(1) \leq \dots \leq f_A(n-1)$  per qualsevol matriu de Monge  $A$  de mida  $n \times n$ .

(b) (1 punt) A continuació es descriu un algorisme de dividir i vèncer que calcula la funció  $f_A$  per a totes les files d'una matriu de Monge  $A$ :

- (1) Construïm dues submatrius quadrades  $B_1$  i  $B_2$  de la matriu  $A$  de mida  $\frac{n}{2} \times \frac{n}{2}$  de la forma següent:  $B_1$  està formada per les files d' $A$  amb índex parell i les columnes entre 0 i  $\frac{n}{2} - 1$ , i  $B_2$  està formada per les files d' $A$  amb índex parell i les columnes entre  $\frac{n}{2}$  i  $n - 1$ .
- (2) Recursivament determinem la columna on apareix el mínim més a l'esquerra per a tota fila de  $B_1$  i de  $B_2$ .
- (3) Calculem la columna on apareix el mínim més a l'esquerra de tota fila d' $A$ .

Expliqueu com, a partir del resultat de (2), es pot fer (3) en temps  $\Theta(n)$ .

- (c) (1 punt) Calculeu el cost en funció de  $n$  de l'algorisme proposat a l'apartat anterior per calcular la funció  $f_A$  per a totes les files d'una matriu de Monge  $A$  de mida  $n \times n$ . Assumiu que el pas (1) es pot dur a terme en temps  $\Theta(n)$ .

## Examen Parcial EDA

Duració: 2.5 hores

07/11/2016

## Problema 1

(2 punts)

En aquest problema no cal justificar les respostes.

- (a) (0.8 pts.) Ompliu els buits de la taula següent (excepte la casella marcada amb **No ompliu**) amb els costos en temps per ordenar un vector d'enters de mida  $n$  usant els algorismes que s'indiquen. Assumiu probabilitat uniforme en el cas mig.

	<i>Cas millor</i>	<i>Cas mig</i>	<i>Cas pitjor</i>
Quicksort (amb partició de Hoare)			
Mergesort			
Inserció		<b>No ompliu</b>	

- (b) (0.2 pts.) La solució de la recurrència  $T(n) = 2T(n/4) + \Theta(1)$  és

- (c) (0.2 pts.) La solució de la recurrència  $T(n) = 2T(n/4) + \Theta(\sqrt{n})$  és

- (d) (0.2 pts.) La solució de la recurrència  $T(n) = 2T(n/4) + \Theta(n)$  és

- (e) (0.3 pts.) Què calcula l'algorisme de Karatsuba? Quin cost té?

- (f) (0.3 pts.) Què calcula l'algorisme de Strassen? Quin cost té?

**Problema 2****(3 punts)**

Donat un  $n \geq 2$ , diem que una seqüència de  $n$  enters  $a_0, \dots, a_{n-1}$  és *bicreixent* si  $a_{n-1} < a_0$  i existeix un índex  $t$  (amb  $0 \leq t < n$ ) que satisfà les següents condicions:

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

Per exemple, la seqüència 12,12,15,20,1,3,3,5,9 és bicreixent (preneu  $t = 3$ ).

(a) (2 pts.) Implementeu en C++ una funció

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

que, donats un vector  $a$  que conté una seqüència bicreixent i un enter  $x$ , retorni si  $x$  apareix a la seqüència o no. Si useu funcions auxiliars que no siguin de la llibreria estàndard de C++, implementeu-les també. Cal que la solució tingui cost  $\Theta(\log(n))$  en temps en el cas pitjor.

(b) (1 pt.) Justifiqueu que el cost en temps en el cas pitjor de la vostra funció *search* és  $\Theta(\log(n))$ . Quan es dona aquest cas pitjor?

## Problema 3

(2 punts)

Considereu la següent funció:

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

- (a) (1 pt.) Donats dos enters  $m, n \geq 0$ , què calcula  $mystery(m, n)$ ? No cal justificar la resposta.



(b) (1 pt.) Analitzeu el cost en temps en el cas pitjor en funció de  $n$  de  $mystery(m, n)$ .

**Problema 4****(3 punts)**

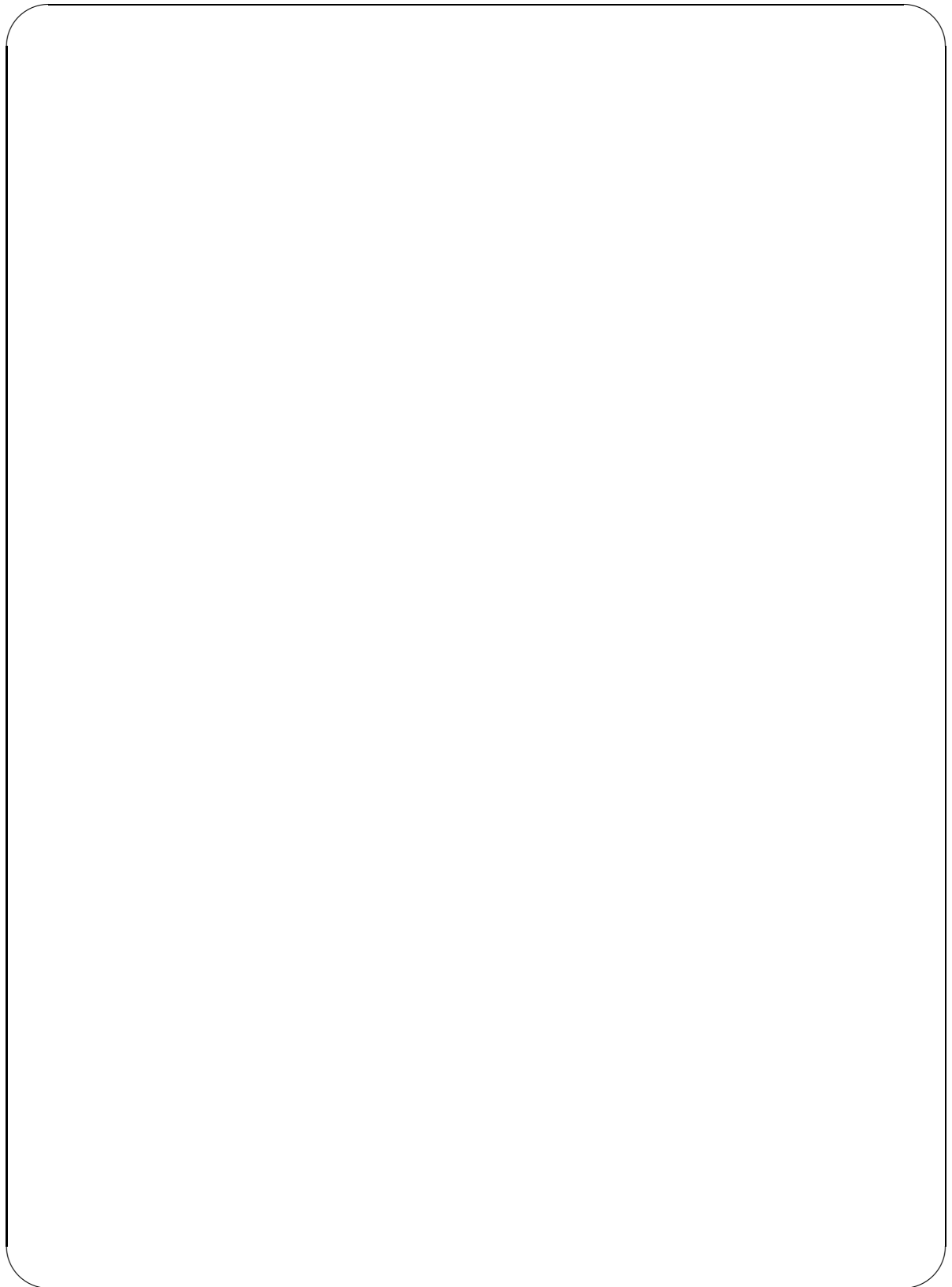
La successió de Fibonacci ve definida per la recurrència  $F(0) = 0$ ,  $F(1) = 1$  i

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

(a) (0.5 pts.) Sigui  $\phi = \frac{\sqrt{5}+1}{2}$  l'anomenat *nombre d'or*. Demostreu que  $\phi^{-1} = \phi - 1$ .

(b) (1.5 pts.) Demostreu que per a tot  $n \geq 0$  es té que

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



(c) (1 pt.) Demostreu que  $F(n) = \Theta(\phi^n)$ .



## Examen Parcial EDA

Duració: 2.5 hores

20/04/2017

## Problema 1

(2 punts)

- (a) (0.5 pts.) Calculeu el cost mig de l'algorisme d'ordenació per inserció per ordenar un vector de  $n$  enters diferents quan amb probabilitat  $\frac{\log n}{n}$  s'escull un vector ordenat del revés i amb probabilitat  $1 - \frac{\log n}{n}$  s'escull un vector ordenat.

- (b) (0.5 pts.) Considereu la següent funció:

```
bool mystery(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;  
}
```

Completeu: la funció *mystery* calcula  i el seu cost és  $O(\text{  })$ . No cal justificar la resposta.

- (c) (0.5 pts.) Demostreu que  $n! = O(n^n)$  aplicant la definició (sense usar límits).

(d) (0.5 pts.) Demostreu que  $n! = \Omega(2^n)$  aplicant la definició (sense usar límits).

## Problema 2

(3 punts)

Donat un  $n \geq 1$ , es diu que una seqüència de  $n$  enters  $a_0, \dots, a_{n-1}$  és *unimodal* si existeix  $t$  amb  $0 \leq t < n$  tal que  $a_0 < \dots < a_{t-1} < a_t$  i  $a_t > a_{t+1} > \dots > a_{n-1}$ . A l'element  $a_t$  se l'anomena el *cim* de la seqüència.

Per exemple, la seqüència 1,3,5,9,4,1 és unimodal, i el seu cim és 9 (preneu  $t = 3$ ).

(a) (1.5 pts.) Implementeu una funció `int top(const vector<int>& a)` en C++ que, donat un vector no buit  $a$  que conté una seqüència unimodal, retorni l'índex del cim de la seqüència. Si useu funcions auxiliars que no siguin de la llibreria estàndard de C++, implementeu-les també. Cal que la solució tingui cost  $\Theta(\log n)$  en temps en el cas pitjor. Justifiqueu que el cost és en efecte  $\Theta(\log n)$ , i doneu una situació en què es pot donar aquest cas pitjor.

- (b) (1.5 pts.) Implementeu una funció `bool search(const vector<int>& a, int x)` en C++ que, donat un vector no buit  $a$  que conté una seqüència unimodal i un enter  $x$ , retorni si  $x$  apareix a la seqüència o no. Si useu funcions auxiliars que no siguin de la llibreria estàndard de C++, implementeu-les també. Cal que la solució tingui cost  $\Theta(\log n)$  en temps en el cas pitjor. Justifiqueu que el cost és en efecte  $\Theta(\log n)$ , i doneu una situació en què es pot donar aquest cas pitjor.

**Problema 3****(2 punts)**

La següent classe *vect* és una simplificació de la classe **vector** de la STL:

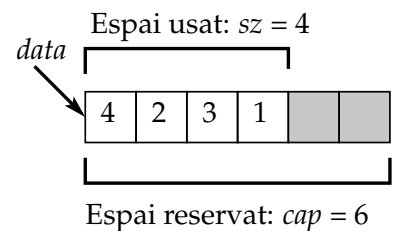
```
class vect {  
    int sz, *data, cap;  
    int new_capacity ();  
    void reserve (int new_cap);  
public:
```

```

vect ()                { sz = cap = 0; data = nullptr; }
int& operator[] (int index) { return data[index]; }
int size ()            { return sz; }
void push_back (int value) {
    if (sz ≥ cap) reserve (new_capacity ());
    data[sz] = value;
    ++sz;
}

```

En un objecte de la classe *vect*, el camp *sz* guarda el nombre d'elements que actualment el vector conté. La memòria reservada per al vector, apuntada pel camp *data*, sempre té espai per guardar els *sz* elements actuals, i possiblement algun més. Al nombre màxim d'elements que es podrien guardar en la memòria reservada se l'anomena capacitat, i és el valor del camp *cap*. El diagrama de la dreta il·lustra una possible implementació d'un vector amb contingut (4,2,3,1). Noteu que les caselles grises estan reservades però no s'usen.



La raó d'aquesta estructura de dades és que cridar al sistema per demanar memòria és una operació costosa que no es vol fer sovint. La funció **void** *reserve* (**int** *new\_cap*), la implementació de la qual no es detalla, s'encarrega d'això: demana un nou fragment de memòria prou gran com per poder-hi encabir *new\_cap* elements, hi copia el contingut del vector antic i actualitza convenientment els camps *sz*, *data* i *cap*.

Observeu que, cada vegada que es crida la funció *push\_back*, si no hi ha prou espai en reserva, es crida la funció **int** *new\_capacity* () que, a partir de la capacitat actual, determina la nova capacitat per a la funció *reserve*.

(a) (0.5 pts.) Considereu la següent implementació de la funció *new\_capacity*:

```

int new_capacity () {
    const int A; // A és un paràmetre prefixat tal que A ≥ 1
    return cap + A; }

```

Quant val exactament *cap* en un vector que inicialment té *cap* = 0 i al qual s'ha aplicat l'operació *reserve* *m* cops? Anomenem  $C(m)$  a aquesta quantitat.

Si fem *n* crides a *push\_back* sobre un vector inicialment buit (*sz* = *cap* = 0), quants cops **exactament** s'ha hagut de fer la crida a *reserve* (és a dir, quin és el valor de *m* tal que  $n \leq C(m)$  i  $C(m-1) < n$ )? Doneu-ne també una expressió asimptòtica el més precisa i simple possible.



- (b) (0.75 pts.) Demostreu que la solució de la recurrència definida per  $C(0) = 0$ ,  $C(m+1) = AC(m) + B$ , on  $A > 1$  i  $B \geq 1$ , és  $C(m) = \frac{BA^m - B}{A-1}$  per a  $m \geq 0$ .

- (c) (0.75 pts.) El mateix que a l'apartat (a), però amb la següent funció *new\_capacity*:

```
int new_capacity() {  
    const int A, B; // A, B són paràmetres prefixats tals que  $A > 1$ ,  $B \geq 1$   
    return A*cap + B; }
```

**Problema 4****(3 punts)**

Volem tenir una funció

`int stable_partition (int x, vector<int>& a)`

que, donats un enter  $x$  i un vector de  $n$  enters diferents  $a = (a_0, a_1, \dots, a_{n-1})$ , reordena el contingut del vector de forma que tots els elements del subvector  $a[0..k]$  són menors o iguals a  $x$ , i tots els elements del subvector  $a[k + 1..n - 1]$  són majors que  $x$ , i també torna l'índex  $k$  ( $k = -1$  si tots els elements de  $a$  són majors que  $x$ ).

A més, l'ordre relatiu original dels elements de  $a$  es respecta:

- si  $a[i]$  i  $a[j]$  amb  $i < j$  són tots dos menors o iguals que  $x$ , llavors en el vector final  $a[i]$  estarà abans que  $a[j]$ , i tots dos s'hauran col·locat a la "part"  $a[0..k]$  que conté els elements  $\leq x$ ;
- si  $a[i]$  i  $a[j]$  amb  $i < j$  són tots dos majors que  $x$ , llavors en el vector final  $a[i]$  estarà abans que  $a[j]$ , i tots dos s'hauran col·locat a la "part"  $a[k + 1..n - 1]$  que conté els elements  $> x$ .

Per exemple, donats  $x = 2$  i la seqüència  $a = (1, 5, 3, 0, 4)$ , voldríem que la funció actualitzés  $a$  a  $(1, 0, 5, 3, 4)$ , i que retornés  $k = 1$ .

- (a) (1 pt.) Implementeu la funció `stable_partition` en C++. Cal que el cost en temps sigui  $\Theta(n)$ . Justifiqueu el cost. Quin és el cost en espai de la memòria auxiliar que fa servir la vostra implementació?

- (b) (0.5 pts.) Què fa la següent funció *mystery*? No cal justificar la resposta.

```

void mystery_aux(vector<int>& a, int l, int r) {
    // Pre:  $0 \leq l \leq r < a.size()$ 
    for (int i = l, j = r; i < j; ++i, --j) swap(a[i], a[j]);
}

void mystery(vector<int>& a, int l, int p, int r) {
    // Pre:  $0 \leq l \leq p \leq r < a.size()$ 
    mystery_aux(a, l, p);
    mystery_aux(a, p+1, r);
    mystery_aux(a, l, r);
}

```

- (c) (1.5 pts.) Ompliu els buits de la següent implementació alternativa de *stable\_partition* i analitzeu-ne el cost en temps en el cas pitjor. Assumiu que, en el cas pitjor, a cada crida recursiva de *stable\_partition\_rec* es té que  $q - p = \Theta(r - l + 1)$ .

```

int stable_partition (int x, vector<int>& a) {
    return stable_partition_rec (x, a, 0, a.size ()-1);
}

int stable_partition_rec (int x, vector<int>& a, int l, int r) {
    if (l == r) {
        if (a[l] ≤ x) return l;
        else return  ;
    }
    int m = (l+r)/2;
    int p = stable_partition_rec (x, a, l, m);
    int q = stable_partition_rec (x, a, , r);
    mystery(a, , m, );
    return  ; }

```

Anàlisi del cost:

## Examen Parcial EDA

Duració: 2.5 hores

06/11/2017

## Problema 1

(3 punts)

Responen les següents preguntes. No cal justificar les respostes.

(a) (0.5 pts.) El cost en temps del següent fragment de codi en funció de  $n$ :

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

és  $\Theta(\quad)$ .

(b) (1 pt.) Donats un vector d'enters  $v$  i un enter  $x$ , la funció

```
int posicio (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;
}
```

examina les  $n = v.size()$  posicions de  $v$  i retorna la primera que conté  $x$ , o  $-1$  si no n'hi ha cap.

Suposem que  $x$  apareix al vector  $v$ . El cost asimptòtic en temps de *posicio* en el cas pitjor és  $\Theta(\quad)$ , i en el cas mig (assumint probabilitat uniforme) és  $\Theta(\quad)$ .

(c) (0.5 pts.) Doneu l'ordre de magnitud de la següent funció de la forma més senzilla possible:  $5(\log n)^2 + 2\sqrt{n} + \cos(n^8) = \Theta(\quad)$ .

(d) (0.5 pts.) La solució de la recurrència

$$T(n) = \begin{cases} 1 & \text{si } 0 \leq n < 2 \\ 4 \cdot T(n/2) + 3n^2 + 2\log(\log n) + 1, & \text{si } n \geq 2 \end{cases}$$

és  $T(n) = \Theta(\quad)$ .

(e) (0.5 pts.) La solució de la recurrència

$$T(n) = \begin{cases} 1 & \text{si } 0 \leq n < 2 \\ 4 \cdot T(n-2) + 3n^2 + 2\log(\log n) + 1, & \text{si } n \geq 2 \end{cases}$$

és  $T(n) = \Theta(\quad)$ .

## Problema 2

(2.5 punts)

Considereu el següent programa, que llegeix un enter estrictament positiu  $m$  i una seqüència de  $n$  enters  $a_0, a_1, \dots, a_{n-1}$  que es garanteix que es troben entre 1 i  $m$ :

```
int main() {
    int m;
    cin >> m;
    vector<int> a;
    int x;
    while (cin >> x)
        a.push_back(x);

    vector<int> b(m + 1, 0);
    int n = a.size();
    for (int i = 0; i < n; ++i)
        ++b[a[i]];

    for (int j = 1; j <= m; ++j)
        b[j] += b[j-1];
}
```

(a) (0.5 pts.) Donat un  $y$  tal que  $0 \leq y \leq m$ , quant val  $b[y]$  al final del *main* en termes de la seqüència  $a$ ?

(b) (1 pt.) Definim la *mediana* de la seqüència  $a$  com el valor  $p$  entre 1 i  $m$  tal que  $b[p] \geq \frac{n}{2}$  i  $b[p-1] < \frac{n}{2}$ , on  $b$  és el vector calculat com en el codi mostrat a dalt.

Escriviu en C++ una funció

```
int median(int n, const vector<int>& b);
```

que, a partir de  $n$ , la mida de la seqüència  $a$ , i de  $b$ , calculat com a dalt, trobi la mediana de  $a$ . No es consideraran com a respostes vàlides les solucions amb cost  $\Omega(m)$ . Si feu servir funcions auxiliars, implementeu-les també.

(c) (1 pt.) Analitzeu el cost de la vostra funció *median* de l'apartat anterior en funció de  $m$ .

**Problema 3****(2 punts)**

Considereu el tipus

```
struct complex {  
    int real ;  
    int imag;  
};
```

per a implementar nombres complexos (amb components enteres), on *real* és la part real i *imag* és la part imaginària del nombre complex.

Per exemple, si  $z$  és un objecte de tipus *complex* que representa el nombre  $2 + 3i$ , llavors  $z.\text{real} = 2$  i  $z.\text{imag} = 3$ .

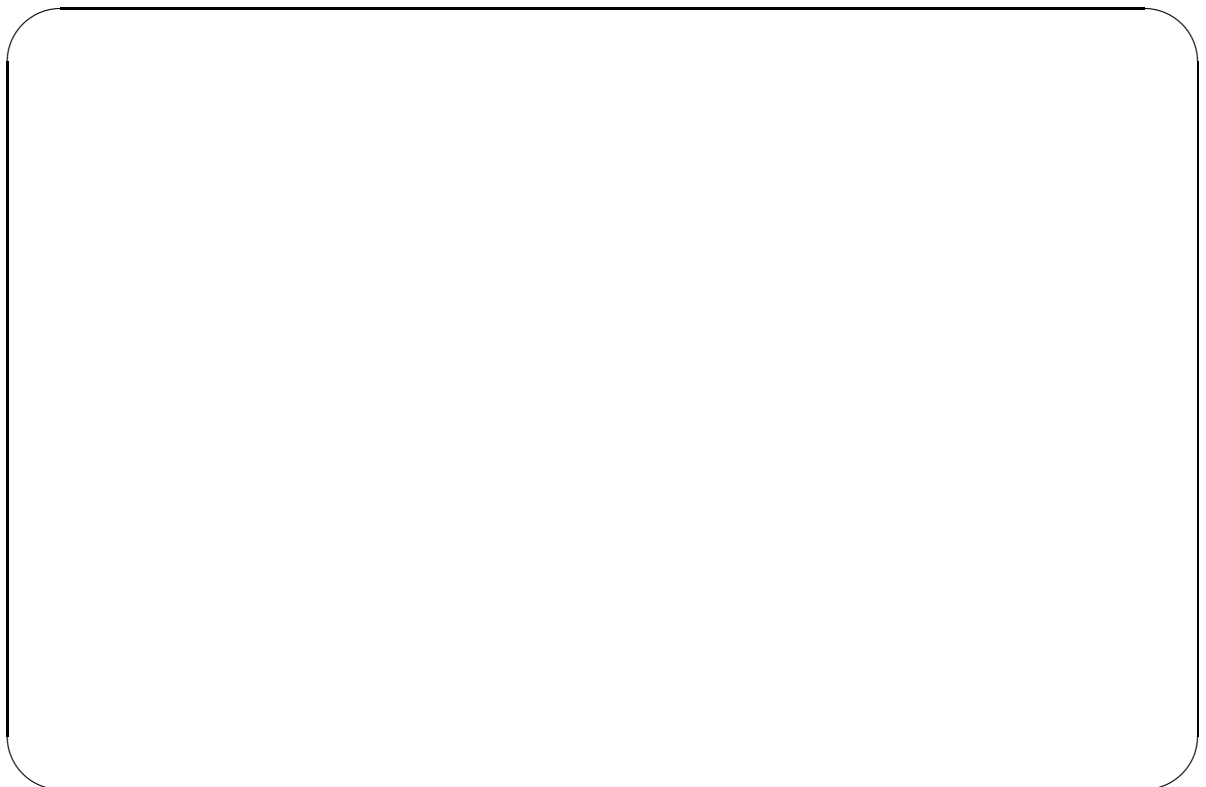
(a) (1 pt.) Implementeu en C++ una funció

```
complex exp(complex z, int n);
```

que, donat un nombre complex  $z$  i un enter  $n \geq 0$ , calculi el nombre complex  $z^n$ . Cal que la solució tingui cost  $\Theta(\log n)$  en temps. Si feu servir funcions auxiliars, implementeu-les també.

*Nota.* Recordeu que el producte de nombres complexos es defineix així:

$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$



(b) (1 pt.) Justifiqueu que el cost en temps de la vostra funció *exp* és  $\Theta(\log n)$ .

**Problema 4****(2.5 punts)**

Considereu la següent recurrència:

$$T(n) = T(n/2) + \log n$$

(a) (1 pt.) Definim la funció  $U$  com  $U(m) = T(2^m)$ . A partir de la recurrència de  $T$ , deduiu una recurrència per a  $U$ .



- (b) (0.5 pts.) Resoleu asimptòticament la recurrència per a  $U$  de la resposta de l'apartat anterior.

- (c) (1 pt.) Resoleu asimptòticament la recurrència per a  $T$ .

Examen Parcial EDA

Duració: 2.5 hores

19/04/2018

## Problema 1

(1.5 punts)

(a) (0.5 pts.) Considereu les funcions  $f(n) = n^{n^2}$  i  $g(n) = 2^{2^n}$ .

Quina funció de les dues creix asimptòticament més de pressa?

Justificació:

*Nota:* Cal entendre  $n^{n^2}$  com  $n^{(n^2)}$ , i similarmet  $2^{2^n}$  com  $2^{(2^n)}$ .

(b) (0.5 pts.) Supposeu que les entrades de mida  $n$  d'un cert algorisme són dels tipus següents:

- Tipus 1: per cada entrada d'aquest tipus, l'algorisme triga temps  $\Theta(n^4)$ . A més, la probabilitat que l'entrada sigui d'aquest tipus és  $\frac{1}{n^3}$ .
- Tipus 2: per cada entrada d'aquest tipus, l'algorisme triga temps  $\Theta(n^3)$ . A més, la probabilitat que l'entrada sigui d'aquest tipus és  $\frac{1}{n}$ .
- Tipus 3: per cada entrada d'aquest tipus, l'algorisme triga temps  $\Theta(n)$ . A més, la probabilitat que l'entrada sigui d'aquest tipus és  $1 - \frac{1}{n^3} - \frac{1}{n}$ .

Aleshores el cost de l'algorisme en el cas mig és .

Justificació:

(c) (0.5 pts.) Resoleu la recurrència  $T(n) = 2T(n/4) + \Theta(\sqrt{n})$ .

Resposta:  $T(n) = \Theta(\text{  })$ .

Justificació:

**Problema 2****(2 punts)**

En aquest problema prendrem  $n$  com a una constant. Donats un vector  $x_0 \in \mathbb{Z}^n$  i una matriu quadrada  $A \in \mathbb{Z}^{n \times n}$ , definim la successió  $x(0), x(1), x(2), \dots$  així:

$$x(k) = \begin{cases} x_0 & \text{si } k = 0 \\ A \cdot x(k-1) & \text{si } k > 0 \end{cases}$$

Per exemple, per a  $n = 3$ , si

$$x_0 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \text{i} \quad A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix},$$

tenim que

$$x(0) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad x(1) = \begin{pmatrix} 6 \\ 1 \\ 2 \end{pmatrix}, \quad x(2) = \begin{pmatrix} 9 \\ 6 \\ 1 \end{pmatrix}, \quad \dots$$

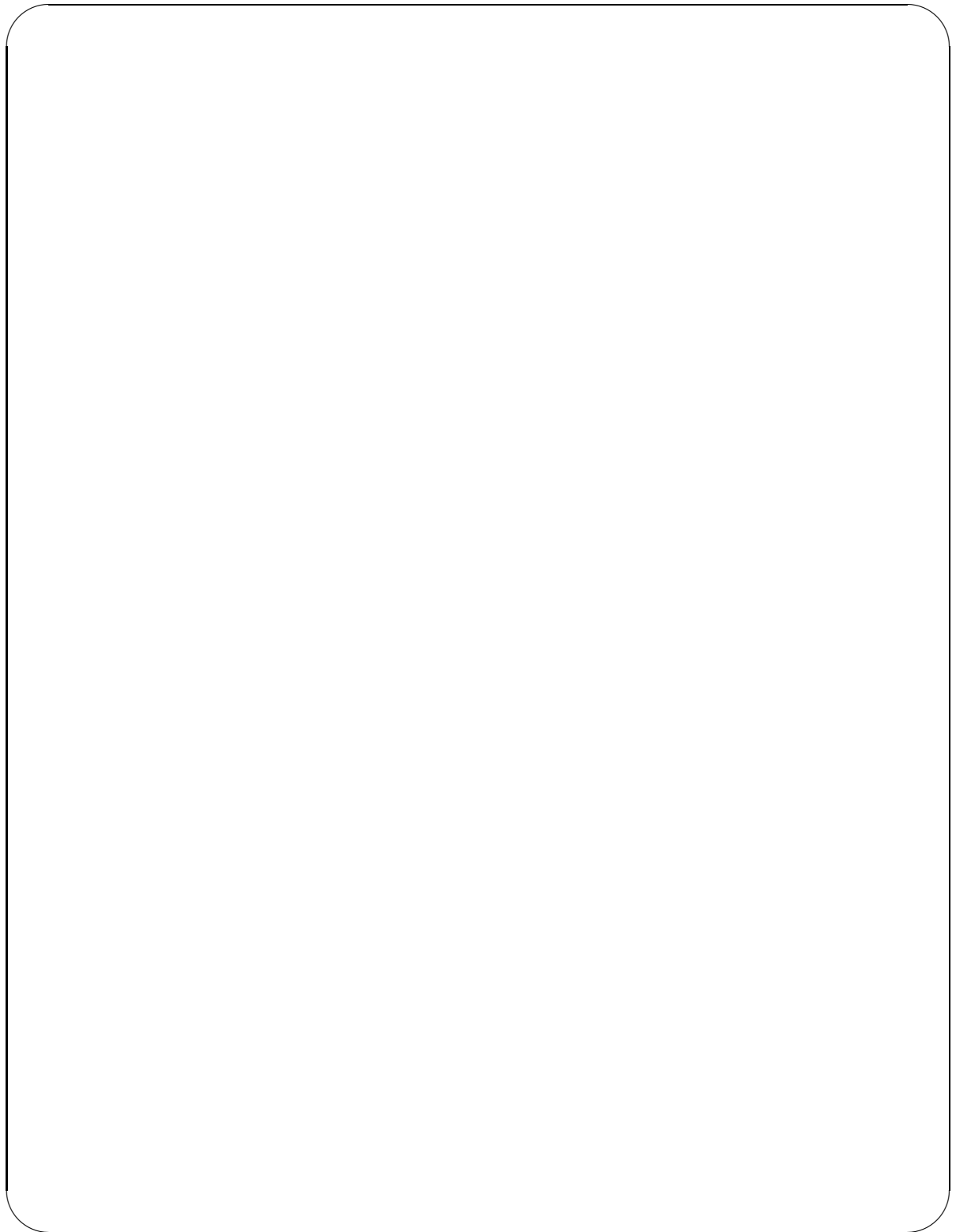
(a) (1 pt.) Demostreu per inducció que  $x(k) = A^k \cdot x_0$  per a tot  $k \geq 0$ .

(b) (1 pt.) Expliqueu a alt nivell com implementaríeu una funció

**vector<int> k\_th(const vector<vector<int>>& A, const vector<int>& x0, int k);**

per calcular el terme  $k$ -èsim  $x(k)$  de la successió definida per la matriu  $A$  i el vector  $x_0$ . Analitzeu-ne també el cost en temps en funció de  $k$ . Cal que el cost sigui millor que  $\Theta(k)$ .



**Problema 4****(3 punts)**

- (a) (1 pt.) Considereu una recurrència de la forma

$$T(n) = T(n/b) + \Theta(\log^k n)$$

on  $b > 1$  i  $k \geq 0$ . Demostreu que la seva solució és  $T(n) = \Theta(\log^{k+1} n)$ .

Justificació:

*Nota:*  $\log^k n$  és una forma breu d'escriure  $(\log(n))^k$ .

*Pista:* feu un canvi de variable.

- (b) (1 pt.) Donat un  $n \geq 1$ , es diu que una seqüència de  $n$  enters  $a_0, \dots, a_{n-1}$  és *unimodal* si existeix  $t$  amb  $0 \leq t < n$  tal que  $a_0 < \dots < a_{t-1} < a_t$  i  $a_t > a_{t+1} > \dots > a_{n-1}$ . A l'element  $a_t$  se l'anomena el *cim* de la seqüència. Per exemple, la seqüència 1,3,5,9,4,1 és unimodal, i el seu cim és 9 (preneu  $t = 3$ ).

Ompliu els buits del codi a continuació per tal que la funció

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

donat un vector no buit  $a$  que conté una seqüència unimodal i un enter  $x$ , retorni si  $x$  apareix a la seqüència o no. No cal justificació.

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



```
bool search(const vector<int>& a, int x) { return search(a, x, 0,  ); }
```

*Nota:* Donats un element  $x$  i iteradors  $first, last$  tals que l'interval  $[first, last)$  està ordenat (creixentment o decreixentment), la funció  $binary\_search(first, last, x)$  retorna **true** si  $x$  apareix a l'interval  $[first, last)$  i **false** altrament, en temps logarítmic en la mida de l'interval en el cas pitjor.

- (c) (1 pt.) Analitzeu el cost en temps en el cas pitjor d'una crida  $search(a, x)$ , on  $a$  és un vector de mida  $n > 0$  que conté una seqüència unimodal i  $x$  és un enter. Descriviu una situació en què es pugui donar aquest cas pitjor.

## Examen Parcial EDA

Duració: 2.5 hores

05/11/2018

## Problema 1

(2 punts)

Per cada afirmació donada a continuació, marqueu amb una X la casella corresponent segons si és certa o falsa.

*Nota:* Cada resposta correcta sumarà 0.2 punts; cada resposta equivocada restarà 0.2 punts, llevat del cas que hi hagi més respostes equivocades que correctes, en què la nota de l'exercici serà 0.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
CERT										
FALS										

- (1) El cost en temps de quicksort en el cas pitjor és  $\Theta(n^2)$ .
- (2) El cost en temps de quicksort en el cas mig és  $\Theta(n^2)$ .
- (3) Qualsevol algorisme que calculi la suma  $x + y$  de dos naturals  $x, y$  de  $n$  bits cadascun, ha de tenir cost en temps  $\Omega(n)$ .
- (4) Existeix un algorisme que calcula la suma  $x + y$  de dos naturals  $x, y$  de  $n$  bits cadascun, en temps  $O(n)$ .
- (5) Qualsevol algorisme que calculi el producte  $x \cdot y$  de dos naturals  $x, y$  de  $n$  bits cadascun, ha de tenir cost en temps  $\Omega(n^2)$ .
- (6) Existeix un algorisme que calcula el producte  $x \cdot y$  de dos naturals  $x, y$  de  $n$  bits cadascun, en temps  $O(n^2)$ .
- (7) Qualsevol algorisme que, donats un enter  $k \geq 0$  i una matriu  $A$  de  $n \times n$  nombres enters, calculi la matriu  $A^k$  ha de fer  $\Omega(k)$  productes de matrius.
- (8)  $2^{2n} \in O(2^n)$ .
- (9)  $2n \in O(n)$ .
- (10)  $\log(2n) \in O(\log n)$ .

## Problema 2

(3 punts)

Donades dues matrius de booleans  $A$  i  $B$  de mida  $n \times n$ , definim el seu *producte lògic*  $P = A \cdot B$  com la matriu  $n \times n$  que al coeficient de la fila  $i$ -èsima i columna  $j$ -èsima ( $0 \leq i, j < n$ ) conté:

$$p_{ij} = \bigvee_{k=0}^{n-1} (a_{ik} \wedge b_{kj})$$

- (a) (0.5 pts.) Considereu la següent funció per a calcular el producte lògic:

```

vector<vector<bool>> product1(const vector<vector<bool>>& A,
                             const vector<vector<bool>>& B) {
    int n = A.size ();
    vector<vector<bool>> P(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                P[i][j] = P[i][j] or (A[i][k] and B[k][j]);
    return P;
}

```

Quin és el cost en temps en el cas pitjor en termes de  $n$ ? El cost és  $\Theta(\text{ })$ .

Doneu un exemple de cas pitjor.

(b) (1 pt.) Considereu la següent alternativa per a calcular el producte lògic:

```

vector<vector<bool>> product2(const vector<vector<bool>>& A,
                             const vector<vector<bool>>& B) {
    int n = A.size ();
    vector<vector<bool>> P(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n and not P[i][j]; ++k)
                if (A[i][k] and B[k][j]) P[i][j] = true;
    return P;
}

```

Quin és el cost en temps en el cas millor en termes de  $n$ ? El cost és  $\Theta(\text{ })$ .

Doneu un exemple de cas millor.

Quin és el cost en temps en el cas pitjor en termes de  $n$ ? El cost és  $\Theta(\text{ })$ .

Doneu un exemple de cas pitjor.

- (c) (1.5 pt.) Expliqueu a alt nivell com implementaríeu una funció per a calcular el producte lògic de matrius que fos més eficient asimptòticament en el cas pitjor que les proposades als apartats (a) i (b). Quin és el seu cost en temps en el cas pitjor?

**Problema 3****(2 punts)**

Considereu la següent funció:

```
int mystery_rec(int n, int l, int r) {  
    if (r == l+1) return l;  
    int m = (l+r)/2;  
    if (m*m ≤ n) return mystery_rec(n, m, r);  
    else return mystery_rec(n, l, m);  
}
```

```
int mystery(int n) {  
    return mystery_rec(n, 0, n+1);  
}
```

- (a) (1 pt.) Donat un enter  $n \geq 0$ , què calcula la funció *mystery*?

- (b) (1 pt.) Quin cost en temps té *mystery*( $n$ ) en funció de  $n$ ? El cost és  $\Theta(\text{  })$

Raoneu la resposta.

#### Problema 4

(3 punts)

Considerem el problema de, donat un vector d'enters (possiblement amb repeticions), ordenar-lo de forma creixent. Els costos a continuació són en temps i es demanen en funció de  $n$ , la mida del vector.

- (a) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per inserció en el cas millor? El cost és  $\Theta(\text{  })$ .
- (b) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per inserció en el cas pitjor? El cost és  $\Theta(\text{  })$ .

- (c) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per fusió en el cas millor? El cost és  $\Theta(\text{ })$ .
- (d) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per fusió en el cas pitjor? El cost és  $\Theta(\text{ })$ .
- (e) (0.75 pts.) Ompliu els buits de la funció *my\_sort* definida a continuació, per tal que donat un vector d'enters *v*, l'ordini de forma creixent:

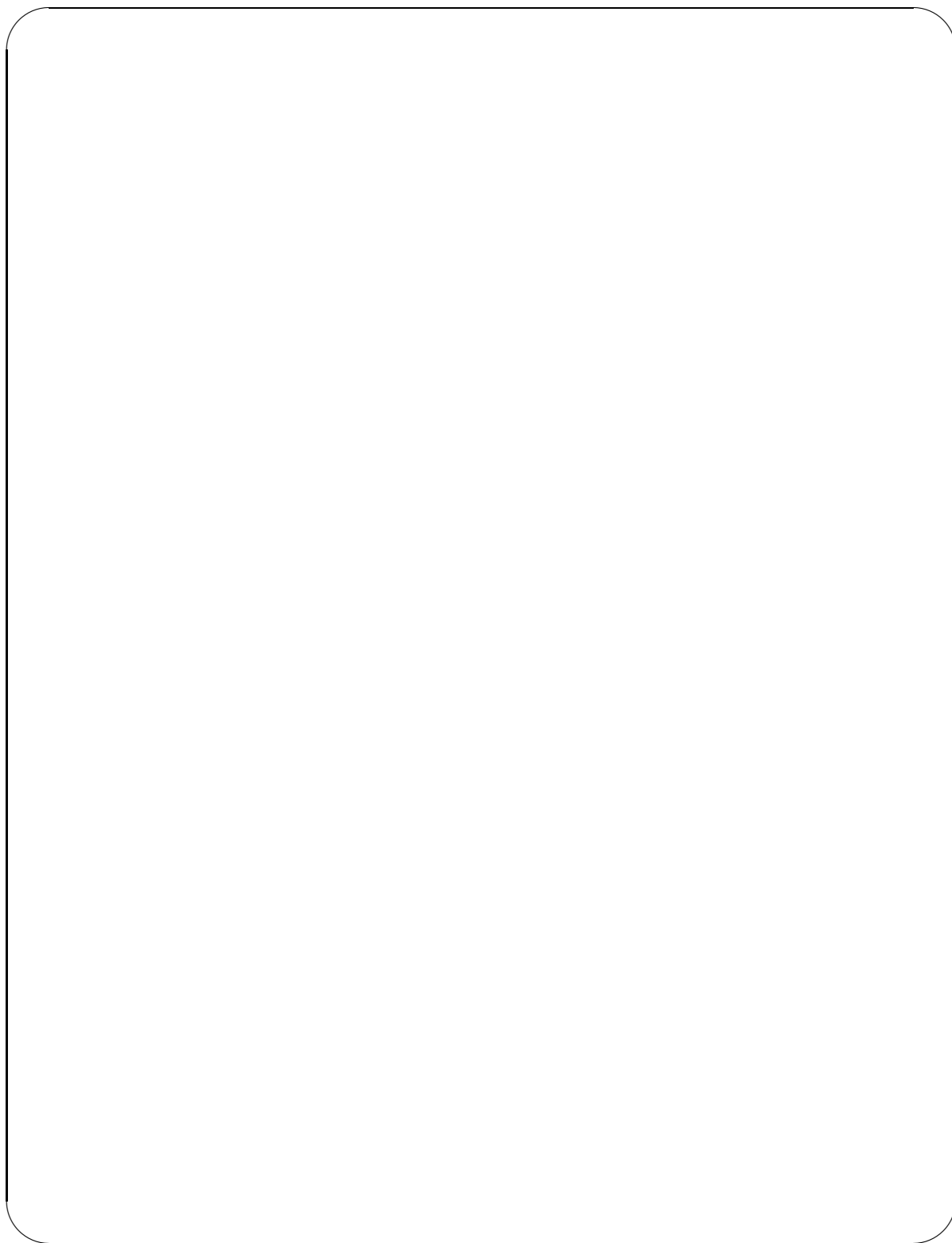
```
void my_sort(vector<int>& v) {  
    int n = v.size ();  
    double lim = n * log(n);  
    int c = 0;  
    for (int i = 1; i < n; ++i) {  
        int x = v[i];  
        int j;  
        for (j = i; j > 0 and  ; --j) {  
            v[j] =  ;  
            ++c;  
        }  
        v[j] =  ;  
        if (c > lim) {  
            merge_sort(v);  
            return;  
        }  
    }  
}
```

La funció auxiliar *merge\_sort* és una implementació de l'algorisme d'ordenació per fusió, i la funció *log* calcula el logaritme neperià (això és, en base *e*).

- (f) (1.25 pts.) Quin és el cost de *my\_sort* en el cas millor? El cost és  $\Theta(\text{ })$ .

Quin és el cost de *my\_sort* en el cas pitjor? El cost és  $\Theta(\text{ })$ .

Raoneu les vostres respostes i doneu un exemple de cas millor i un de cas pitjor.



## Examen Parcial EDA

Duració: 2.5 hores

25/04/2019

## Problema 1

(3 punts)

En aquest problema no cal justificar les respostes.

- (a) (1 pt.) Considereu les funcions  $f(n) = n \log n$  i  $g(n) = n^\alpha$ , on  $\alpha$  és un paràmetre real. Determineu per a quins valors d' $\alpha$ :

(1)  $f = O(g)$  :

(2)  $f = \Omega(g)$  :

(3)  $g = O(f)$  :

(4)  $g = \Omega(f)$  :

- (b) (0.5 pts.) La solució de la recurrència  $T(n) = 2T(n-2) + \Theta(n)$  és asimptòticament  $T(n) = \Theta(\text{  })$ .

- (c) (0.5 pts.) La solució de la recurrència  $T(n) = 2T(n/2) + \Theta(n)$  és asimptòticament  $T(n) = \Theta(\text{  })$ .

- (d) (0.5 pts.) El cost en el cas pitjor de mergesort per ordenar un vector de mida  $n$  en funció de  $n$  és  $\Theta(\text{  })$ .

- (e) (0.5 pts.) El cost en el cas millor de mergesort per ordenar un vector de mida  $n$  en funció de  $n$  és  $\Theta(\text{  })$ .



## Problema 2

(2 punts)

Considereu el programa següent:

```
int partition (vector<int>& v, int l, int r) {
    int x = v[l];
    int i = l - 1;
    int j = r + 1;
    while (true) {
        while (x < v[--j]);
        while (v[++i] < x);
        if (i ≥ j) return j;
        swap(v[i], v[j]);
    }
}

void mystery(vector<int>& v, int l, int r, int m) {
    if (l < r) {
        int q = partition (v, l, r);
        mystery(v, l, q, m);
        int p = q-1;
        if (p < m)
            mystery(v, q+1, r, m-p);
    }
}
```

(a) (1 pt.) Donats un vector  $v$  i un enter  $m \geq 0$ , expliqueu què fa una crida  $mystery(v, 0, v.size() - 1, m)$ .

(b) (1 pt.) Suposem que  $v$  és un vector d'enters diferents ordenats de forma creixent. Sigui  $n = v.size()$ . Analitzeu el cost de cridar  $mystery(v, 0, n-1, n)$  en funció de  $n$ .

**Problema 3****(2 punts)**

En aquest problema volem multiplicar nombres naturals arbitràriament grans. Un natural s'implementa amb un `vector<bool>` usant la seva representació en bits, de més a menys significatiu. Així, per exemple, el vector (1, 1, 0) representa el nombre  $4 + 2 = 6$ .

Suposem que ja hem implementat les funcions següents, amb els costos indicats:

```
// Calcula  $x \times 2$ .
```

```
// Cost:  $\Theta(n)$ , on  $n = x.size()$ .
```

```
vector<bool> twice(const vector<bool>& x)
```

```
// Calcula  $x \div 2$  (divisió entera per defecte).
```

```
// Cost:  $\Theta(n)$ , on  $n = x.size()$ .
```

```
vector<bool> half(const vector<bool>& x)
```

```
// Calcula  $x + y$ .
```

```
// Cost:  $\Theta(n)$ , on  $n = \max(x.size(), y.size())$ .
```

```
vector<bool> sum(const vector<bool>& x, const vector<bool>& y)
```

- (a) (1.5 pts.) Usant aquestes funcions, completeu la implementació de la funció

**vector<bool> prod(const vector<bool>& x, const vector<bool>& y)**

següent per a calcular el producte de  $x$  per  $y$ :

**vector<bool> prod(const vector<bool>& x, const vector<bool>& y) {**

**if** ( $x.size() == 0$  **or**  $y.size() == 0$ ) **return**

**vector<bool> z = twice(twice(prod(half(x), half(y))));**

**vector<bool> one = vector<bool>(1, 1);**

**if** ( $x.back() == 0$  **and**  $y.back() == 0$ ) **return**

**else if** ( $x.back() == 1$  **and**  $y.back() == 0$ ) **return**

**else if** ( $y.back() == 1$  **and**  $x.back() == 0$ ) **return**

**else {**

**} }**

**Nota:** Si  $v$  és un **vector** de la STL, llavors  $v.back()$  és equivalent a  $v[v.size() - 1]$ .

Si  $n = x.size() = y.size()$ , justifiqueu que el cost de cridar  $prod(x, y)$  en funció de  $n$  és  $\Theta(n^2)$ .

- (b) (0.5 pts.) Doneu el nom d'un conegut algorisme per calcular el producte de dos naturals  $x$  i  $y$  de  $n$  bits cadascun més eficient que la funció *prod* anterior. Quin és el cost d'aquest algorisme en funció de  $n$ ? (no cal justificar el cost)

## Problema 4

(3 punts)

Una matriu d'enters de dimensions  $N \times N$  és *biordenada* si cada columna té els elements ordenats en ordre no decreixent de dalt a baix, i cada fila té els elements ordenats en ordre no decreixent d'esquerra a dreta. Per exemple, una matriu d'aquest tipus seria:

$$\begin{pmatrix} 1 & 4 & 9 \\ 9 & 9 & 9 \\ 12 & 14 & 15 \end{pmatrix}$$

Volem, donats un enter  $x$  i una matriu biordenada  $A$ , decidir si  $x$  apareix a  $A$  o no.

- (a) (1.5 pts.) En aquest apartat suposarem que  $N$  és una potència de 2. Donats un enter  $x$ , una matriu biordenada  $A$ , i tres enters  $i, j, n$  tals que  $n$  és una potència de 2 i que  $1 \leq n \leq N$  i  $0 \leq i, j \leq N - n$ , la funció

**bool search1(int x, const vector<vector<int>>& A, int i, int j, int n)**

retorna si  $x$  apareix a la submatriu  $n \times n$  de  $A$  que conté de la fila  $i$  a la fila  $i + n - 1$ , i de la columna  $j$  a la  $j + n - 1$ . Files i columnes s'indexen des de 0.

Completeu **usant recursivitat** la implementació següent de *search1*:

```
bool search1(int x, const vector<vector<int>>& A, int i, int j, int n) {
    if (n == 1) return  ;
    int mi = i + n/2 - 1;
    int mj = j + n/2 - 1;

    if (A[mi][mj] < x) return
        

    if (A[mi][mj] > x) return
        

    return  ;
}
```

**Nota:** a cada caixa es poden fer diverses crides recursives, que podeu combinar amb operadors booleans.

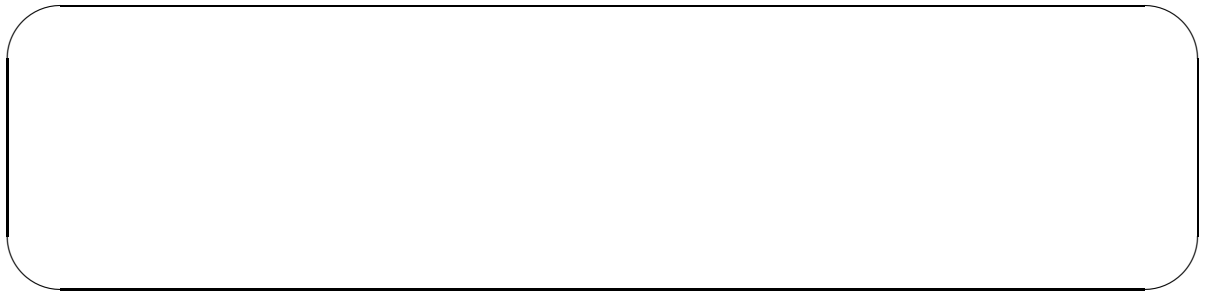
Com es pot utilitzar la funció *search1* per determinar si  $x$  apareix a  $A$  o no? Analitzeu el cost en el cas pitjor del vostre algorisme per fer-ho en funció de  $N$ .

(b) (0.75 pts.) Considereu la funció *search2* següent per dir si  $x$  apareix a  $A$  o no:

```
bool search2(int x, const vector<vector<int>>& A) {  
    int i = A.size() - 1;  
    int j = 0;  
    while (i ≥ 0 and j < A.size())  
        if (A[i][j] > x) --i;  
        else if (A[i][j] < x) ++j;  
        else return true;  
    return false;  
}
```

Si és correcta, justifiqueu-ho. Si no ho és, doneu-ne un contraexemple.

(c) (0.75 pts.) Analitzeu el cost en el cas pitjor de *search2* en funció de  $N = A.size()$ .



Examen Parcial EDA

Duració: 2h45min

11/11/2019

## Problema 1

(1 punt)

- (a) (0.5 pts.) La solució de la recurrència  $T(n) = 2T(n/4) + \Theta(\sqrt{n})$  és asimptòticament  $T(n) = \Theta(\text{ })$ . No cal que justifiqueu la resposta.
- (b) (0.5 pts.) Per a quines  $X \in \{O, \Omega, \Theta\}$  es compleix que  $\log_2(n) \in X(\log_2(\log_2(n^2)))$ ?

## Problema 2

(2.5 punts)

Donat un natural  $n \geq 1$ , qualsevol funció  $f: \{0, 1, \dots, n-1\} \rightarrow \{0, 1, \dots, n-1\}$  es pot representar com un vector d'enters  $[f(0), f(1), \dots, f(n-1)]$ .

Per exemple, si  $n = 5$  i  $f(0) = 2, f(1) = 1, f(2) = 2, f(3) = 4, f(4) = 3$ , la funció  $f$  es pot representar pel vector  $[2, 1, 2, 4, 3]$ . En tot aquest problema, assumirem aquesta representació de funcions.

- (a) (0.75 pts.) Considereu el codi següent:

```
void misteri_aux(const vector<int>& f, const vector<int>& g,
                int i, vector<int>& r) {
    if (i < f.size()) {
```

```
        r[i] = f[g[i]];
        misteri_aux(f,g,i+1,r);
    }
}

vector<int> misteri(const vector<int>&f, const vector<int>&g) {
    // Precondició: f i g tenen la mateix mida i contenen nombres
    // entre 0 i f.size() - 1
    vector<int> r(f.size());
    misteri_aux(f,g,0,r);
    return r;
}
```

Què retorna la funció *misteri*? No cal que justifiqueu la resposta.

Si assumim que  $n$  és la mida de  $f$ , quin és el cost de *misteri* en funció de  $n$ ?

(b) (0.75 pts.) Considereu ara el codi següent:

```
vector<int> misteri_2(const vector<int>&f, int k) {
    if (k == 0) {
        vector<int> r(f.size());
        for (int i = 0; i < f.size(); ++i) r[i] = i;
        return r;
    }
    else return misteri(f, misteri_2(f,k-1));
}
```

Assumint que  $k \geq 0$ , què retorna la funció *misteri\_2*? No cal que justifiqueu la resposta.



Quin és el cost de *misteri\_2* en funció només de *k*?

- (c) (1 pt.) Completeu la funció següent per tal que calculi el mateix que *misteri\_2* però sigui més eficient asimptòticament. Analitzeu-ne el cost en funció de *k*.

```
vector<int> misteri_2_quick(const vector<int>& f, int k) {  
    if (k == 0) {  
        vector<int> r(f.size ());  
        for (int i = 0; i < f.size (); ++i) r[i] = i;  
        return r;  
    }  
}
```

}

Anàlisi del cost en funció de  $k$ :

**Problema 3****(3.25 punts)**

Donat un conjunt  $S$  de  $m = 2n$  enters diferents, volem agrupar-los en parelles de manera que la suma dels seus productes sigui màxima. És a dir, busquem la màxima expressió de la forma  $x_0 * x_1 + x_2 * x_3 + \dots + x_{2n-2} * x_{2n-1}$ , on el  $x_i$ 's són tots els elements de  $S$ .

Per exemple, si  $S = \{5, 6, 1, 3, 8, 4\}$ , dues possibles expressions són  $1 * 5 + 6 * 3 + 4 * 8$ , que suma 55, i  $5 * 4 + 1 * 8 + 3 * 6$ , que suma 46. D'entre aquestes dues preferim la primera, tot i que encara n'hi ha d'altres de millors.

La funció `max_suma` calcula la màxima suma de productes de  $S$ :

```
int pos_max (const vector<int>& v, int l, int r) {
    int p = l;
    for (int j = l + 1; j ≤ r; ++j)
        if (v[j] > v[p]) p = j;
    return p;
}
```

```
int max_suma (vector<int>& S) {
    int suma = 0;
    int m = S.size ();
    for (int i = 0; i < m; ++i) {
        int p = pos_max(S, i, m-1);
        swap(S[i], S[p]);
        if (i%2 == 1) suma += S[i-1]*S[i];
    }
    return suma;
}
```

- (a) (1 pt.) Analitzeu el cost en cas pitjor de `max_suma` en funció de  $m$ , el nombre d'elements del vector  $S$ .

- (b) (1 pt.) Expliqueu a alt nivell com implementaríeu una funció que solucionés el mateix problema però que, en el cas pitjor, fos més eficient asimptòticament que *max\_suma*. Indiqueu clarament quin és el cost resultant.

- (c) (1.25 pts.) Demostreu que la funció *max\_suma* retorna la suma de productes màxima.

*Ajuda:* demostreu primer que si  $x_0$  i  $x_1$  son els dos nombres més grans de  $S$ , aleshores una expressió que conté els productes  $x_0 * y$  i  $x_1 * z$ , per certs  $y, z \in S$  no pot ser màxima. A continuació utilitzeu aquest fet per demostrar la correctesa de *max\_suma* per inducció sobre  $m$ .

## Problema 4

(3.25 punts)

Durant els propers  $n \geq 3$  dies, se celebrarà un important esdeveniment esportiu, pel qual existeix un enorme mercat de compra-venda d'entrades del que ens en volem aprofitar. Sabem que cada dia podrem comprar o vendre una entrada, i també sabem el preu de les entrades en cada dia, donat com una seqüència  $(p_0, p_1, \dots, p_{n-1})$ .

- (a) (1.25 pts.) Ens assabentem que la seqüència de preus segueix una forma ben particular. Hi ha un únic dia  $0 \leq d \leq n-1$  amb preu mínim  $p_d$  i sabem que  $p_0 > p_1 > \dots > p_d$  i  $p_d < p_{d+1} < \dots < p_{n-1}$ .

El nostre objectiu és comprar una entrada el dia  $c$  i vendre-la el dia  $v$ , amb  $0 \leq c \leq v \leq n-1$  de manera que maximitzem els nostres guanys. És a dir, volem que  $p_v - p_c$  sigui màxim. A tal efecte, ompliu els buits del codi següent per tal que la funció *max\_guany* retorni aquest parell  $\langle c, v \rangle$  en temps  $\Theta(\log n)$  i analitzeu per què la funció resultant té aquest cost.

```
int f(const vector<int>& p, int l, int r){
    if (l + 1 ≥ r) return (p[l] ≤ p[r] ? l : r);
    else {
        int m = (l+r)/2;
        if (  ) return f(p, ,  );
        else if (  ) return f(p, ,  );
        else return m;
    }
}

pair<int,int> max_guany (const vector<int>& p) {
    return { ,  };
}
```

*Nota:* recordeu que l'expressió  $(B ? E_T : E_F)$  equival a  $E_T$  si l'expressió booleana  $B$  és certa i equival a  $E_F$  altrament.

Anàlisi del cost:

- (b) (1 pt.) En el que resta d'exercici, assumiu que la seqüència  $p$  no necessàriament té la forma mencionada a l'apartat anterior, sinó que és una seqüència arbitrària de nombres naturals.

En aquest apartat, donat un dia  $k$  en el que necessitem disposar d'una entrada, volem saber quin és el màxim benefici que podem obtenir comprant l'entrada en un cert dia  $c$  i venent-la en un cert dia  $v$ , però que ens garanteixi tenir l'entrada el dia  $k$ . És a dir, no ens val qualsevol parell  $(c, v)$  sinó que necessitem que  $0 \leq c \leq k \leq v \leq n - 1$ . Implementeu una funció amb cost  $\Theta(n)$  per calcular aquest benefici.

```
int max_guany (const vector<int>& p, int k) {
```

```
}
```

- (c) (1 pt.) Afrontem finalment el problema general, en el que la seqüència  $p$  pot tenir qual-sevol forma i volem calcular el màxim guany possible  $p_v - p_c$  que correspon a comprar una entrada en el dia  $c$  i vendre-la posteriorment en el dia  $v$ . Expliqueu a alt nivell com implementaríeu una funció que calculés aquest màxim guany i analitzeu-ne el cost. Solucions amb cost  $\Omega(n^2)$  rebran 0 punts.

*Ajuda:* la funció de l'apartat anterior us pot ser útil per implementar una solució basada en dividir i vèncer.

**Examen Parcial EDA****Duració: 1h15min****23/04/2020****Problema 1****(3 punt)**

Test [https://jutge.org/problems/X71865\\_ca](https://jutge.org/problems/X71865_ca) a omplir a través de Jutge.org. Teniu 20 minuts per entregar les respostes.

**Problema 2****(3 punts)**

Un problema escollit aleatòriament d'entre:

Jutge.org, Problema X35804: Creuers  
([https://jutge.org/problems/X35804\\_ca](https://jutge.org/problems/X35804_ca)).

Jutge.org, Problema X22314: Donacions  
([https://jutge.org/problems/X22314\\_ca](https://jutge.org/problems/X22314_ca)).

Jutge.org, Problema X79163: Efemèrides  
([https://jutge.org/problems/X79163\\_ca](https://jutge.org/problems/X79163_ca)).

**Problema 3****(4 punts)**

Un problema escollit aleatòriament d'entre:

Jutge.org, Problema X30043: Vector xulo  
([https://jutge.org/problems/X30043\\_ca](https://jutge.org/problems/X30043_ca)).

Jutge.org, Problema X74873: Vector xulo  
([https://jutge.org/problems/X74873\\_ca](https://jutge.org/problems/X74873_ca)).

Jutge.org, Problema X83303: Vector xulo  
([https://jutge.org/problems/X83303\\_ca](https://jutge.org/problems/X83303_ca)).

Jutge.org, Problema X90362: Vector xulo  
([https://jutge.org/problems/X90362\\_ca](https://jutge.org/problems/X90362_ca)).

## Examen Parcial EDA

Duració: 1h 30min

06/11/2020

## Problema 1

(2.5 pts.)

Respon a les següents preguntes:

- (a) (1.5 pts.) Donat un vector  $v$  d'enters ordenats creixentment i un enter  $x$ , volem determinar si  $x$  apareix a  $v$ . En lloc d'implementar una cerca binària, ens proposen la idea d'escollir dos elements que parteixin el vector en tres parts iguals i determinar en quina d'aquestes tres parts cal buscar  $x$ . Completa el següent codi perquè sigui una implementació correcta d'aquesta idea:

```
bool tri_search (const vector<int>& v, int l, int r, int x) {  
    if (l > r) return false;  
    else {  
        int n_elems = (r-l+1);  
        int f = l + n_elems/3;  
        int s = r - n_elems/3;  
  
        if (  ) return true;  
        if (  ) return tri_search(v,  ,  , x);  
        if (  ) return tri_search(v,  ,  , x);  
        return tri_search (v,  ,  , x);  
    }  
}  
  
bool tri_search (const vector<int>& v, int x) {  
    return tri_search (v, 0, v.size ()-1, x);  
}
```

Si  $n$  és el nombre d'elements del vector  $v$ , analitzeu el cost en cas pitjor d'una crida  $tri\_search(v, x)$  en funció de  $n$ .



- (b) (1 pt.) Doneu dues funcions  $f$  i  $g$  amb  $f \notin \Theta(g)$  tals que ambdues siguin  $\Omega(n)$  i  $O(n \log n)$  però que cap sigui  $\Theta(n)$  ni  $\Theta(n \log n)$ .

### Problema 2

(7.5 pts.)

Donat un vector  $v$  d' $n$  naturals volem determinar si existeix un element *dominant*, és a dir, si existeix un element que apareix més de  $n/2$  vegades. Per exemple:

- Si  $v = \{5, 2, 5, 2, 8, 2, 2\}$ , aleshores 2 és l'element dominant perquè apareix  $4 > 7/2$  vegades.
- Si  $v = \{3, 2, 3, 3, 2, 3\}$ , aleshores 3 és l'element dominant perquè apareix  $4 > 6/2$  vegades.
- Si  $v = \{6, 1, 6, 1, 6, 2, 9\}$ , no hi ha cap element dominant perquè cap d'ells apareix més de  $7/2$  vegades.

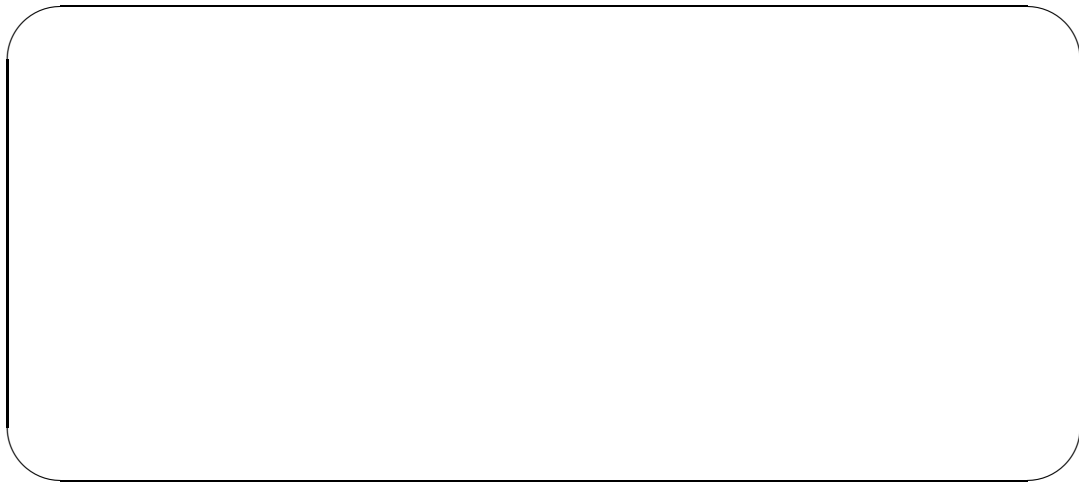
Volem obtenir una funció en C++ que rebi el vector  $v$  i retorni l'element dominant de  $v$ , o el nombre  $-1$  en cas que no existeixi cap element dominant.

- (a) (2.5 pts.) Un estudiant de PRO2 ens suggereix la següent solució:

```
int dominant_pro2 (vector<int> v) {
    int n = v.size ();
    for (int i = 0; i < n; ++i) {
        if (v[i] != -1) {
            int times = 0;
            int candidate = v[i];
            for (int j = i; j < n; ++j) {
                if (v[j] == candidate) {
                    ++times;
                    v[j] = -1;
                }
            }
        }
    }
}
```

```
        if (times > n/2) return candidate;  
    } } } }  
    return -1;  
}
```

Analitzeu el seu cost en cas pitjor en funció de  $n$ . Expliqueu com construiríeu un vector de mida  $n$  pel qual es doni aquest cas pitjor.



Analitzeu el seu cost en cas millor en funció de  $n$ . Expliqueu com construiríeu un vector de mida  $n$  pel qual es doni aquest cas millor.



Si ens asseguren que, per a qualsevol  $n$ , el vector  $v$  sempre tindrà com a molt 100 naturals diferents, canviaria el seu cost en cas pitjor?

- (b) (2 pts.) Un altre estudiant de *PRO2* se n'adona que si primer ordenem el vector existeix un algorisme ben senzill:

```
int dominant_sort (vector<int> v) {  
    int n = v.size ();  
    own_sort(v.begin (), v.end ());  
    int i = 0;  
    while (i < n) {  
        int times = 0, j = i;  
        while (j < n and v[j] == v[i]){  
            ++times;  
            ++j;  
        }  
        if (times > n/2) return v[i];  
        i = j;  
    }  
    return -1;  
}
```

Si *own\_sort* es correspon a una ordenació per inserció, quin és el cost en cas millor i pitjor de *dominant\_sort* en funció de  $n$ ?

Si *own\_sort* implementa un *quicksort*, quin és el cost en cas millor i pitjor de *dominant\_sort* en funció de  $n$ ?

- (c) (3 pts.) Finalment, un estudiant d'EDA molt aplicat, encara que no brillant, ens suggereix una solució basada en dividir i vèncer. No obstant, s'han perdut parts del codi i us demanem que completeu la següent funció:

```
int times (const vector<int>& v, int l, int r, int x) {
    if (l > r) return 0;
    return (v[l] == x) + times(v, l+1, r, x);
}

int dominant_divide (const vector<int>& v, int l, int r) {
    if (l == r) return v[l];
    int n_elems = (r-l+1);
    int m = (l+r)/2;
    int maj_left = dominant_divide(v, ,  );
    if ( maj_left != -1 and times(  ) > n_elems/2) return  ;
    int maj_right = dominant_divide(v, ,  );
    if ( maj_right != -1 and times(  ) > n_elems/2) return  ;
    return -1;
}

int dominant_divide (const vector<int>& v) {
    return dominant_divide(v, 0, v.size ()-1);
}
```

Analitzeu el cost de *dominant\_divide* en cas pitjor en funció de  $n$ .



## Examen Parcial EDA

Duració: 1h 30min

15/04/2021

## Problema 1

(5 pts.)

Responen a les següents preguntes:

- (a) (1.25 pts.) Escriviu C o F dins de cada casella indicant si l'afirmació corresponent és certa o falsa, respectivament:

$2^{2n} \in O(2^n)$  ☐

$\log(2n) \in O(\log(n))$  ☐

$2^{2n} \in \Omega(2^n)$  ☐

$\log(2n) \in \Omega(\log(n))$  ☐

$2^{2n} \in \Theta(2^n)$  ☐

$\log(2n) \in \Theta(\log(n))$  ☐

Justifiqueu la vostra resposta:

- (b) (1.25 pts.) Considereu el codi següent:

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

En funció de  $n$ , el seu cost és  $\Theta(\text{  })$ . Justifiqueu la vostra resposta:

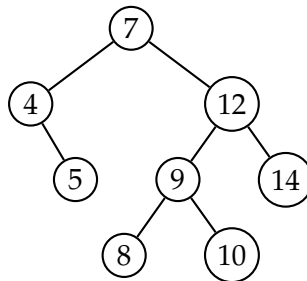
(c) (1.25 pts). Considereu el codi següent:

```
bool f(const map<int,int>& M, const vector<int>& v) {  
    for (int x : v)  
        if (M.find(x) != M.end()) return true;  
    return false;  
}
```

```
int main() {  
    int n; cin >> n;  
  
    map<int,int> M;  
    for (int i = 0; i < n; ++i) {  
        int x; cin >> x;  
        ++M[x];  
    }  
  
    vector<int> v(n);  
    for (int i = 0; i < n; ++i) cin >> v[i];  
  
    cout << f(M,v) << endl;  
}
```

Què fa el codi anterior? Quin és el cost en cas pitjor d'una crida a  $f$  en funció d' $n$ ?

- (d) (1.25 pts.) Escriuiu l'arbre AVL resultant d'afegir l'element amb clau 11 a l'arbre AVL següent. No cal justificar la resposta.



### Problema 2

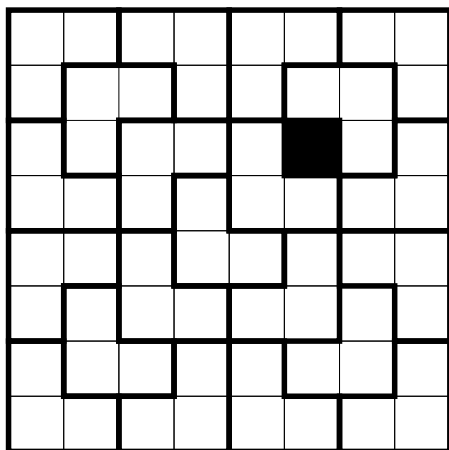
(5 pts.)

Donada una graella de mida  $2^n \times 2^n$ , amb  $n \geq 0$ , i que té exactament una casella bloquejada, ens demanen omplir la resta de la caselles amb les següents peces:



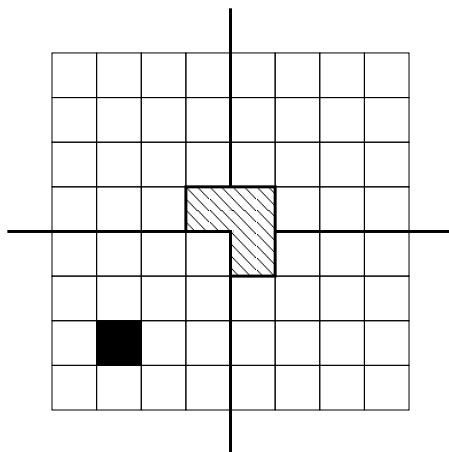
Com és d'esperar, les peces no es poden solapar ni sortir de la graella. Per exemple, per una graella  $8 \times 8$  i la casella bloquejada en negre, una possible solució és:





- (a) (2 pts.) Demostreu que per a tot  $n \geq 0$ , sigui quina sigui la posició de la casella bloquejada, sempre podrem omplir la resta de les caselles amb les peces indicades.

*Pista:* Observeu la següent figura.



- (b) (2 pts.) Completeu el següent codi per tal de resoldre el problema plantejat. La matriu  $M$  representa la graella. Les files s'indexen de dalt a baix i les columnes d'esquerra a dreta.

```

void write_sol (const vector<vector<int>>& M);
typedef pair<int,int> Coord;

// Returns quadrant of pos in square [i_l,i_r] x [j_l,j_r]
// Quadrants are:
// 0 1
// 2 3
int quadrant(Coord pos, int i_l, int i_r, int j_l, int j_r) {
    int size = j_r - j_l + 1;
    int i_m = i_l + size / 2;
    int j_m = j_l + size / 2;
    if (  ) return 0;
    if (  ) return 1;
    if (  ) return 2;
    return 3;
}

void fill (vector<vector<int>>& M, int i_l, int i_r, int j_l, int j_r,
    Coord c_blocked, int& num){
    if ( i_l == i_r ) return; // 1x1
    int size = j_r - j_l + 1;
    int i_m = i_l + size / 2; // Midpoints
    int j_m = j_l + size / 2;

    vector<Coord> coords_blocked(4); // Blocked cell in each quadrant
    coords_blocked [0] = { ,  };
    coords_blocked [1] = { ,  };
    coords_blocked [2] = { ,  };
    coords_blocked [3] = { ,  };
    int q = quadrant(c_blocked, i_l, i_r, j_l, j_r);
    coords_blocked [q] = c_blocked ;

    for (int k = 0; k < 4; ++k)
        if (M[coords_blocked[k].first][coords_blocked[k].second] == )
            M[coords_blocked[k].first][coords_blocked[k].second] =  ;
        ++num;

    fill (M, , num); // Q0
    fill (M, , num); // Q1
    fill (M, , num); // Q2
    fill (M, , num); // Q3
}

int main(){

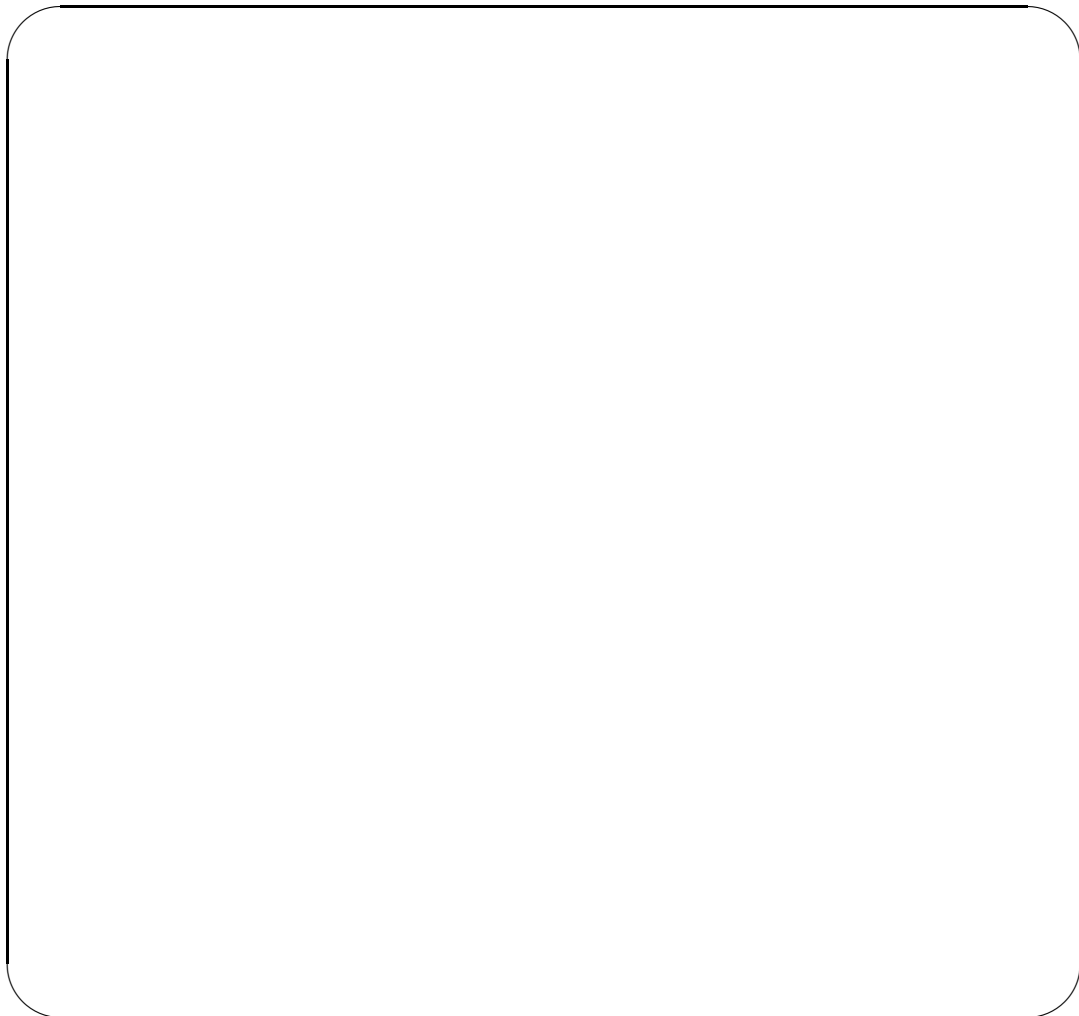
```

```
int n; cin >> n;
int size = pow(2,n);
Coord blocked;
cin >> blocked.first >> blocked.second;
vector<vector<int>>> M(size,vector<int>(size,-1));

M[blocked.first][blocked.second] = 0;
// 0 initially occupied, -1 to be filled yet, n > 0 indicates piece identifier

int num = 1; // All cells with the same num are part of the same piece
fill (M, 0, size - 1, 0, size - 1, blocked, num);
write_sol (M);
}
```

(c) (1 pt.) Quin és el cost del codi anterior en funció de  $n$ ? I en funció del nombre de caselles?



## Examen Parcial EDA

Duració: 1h 30min

08/11/2021

## Problema 1

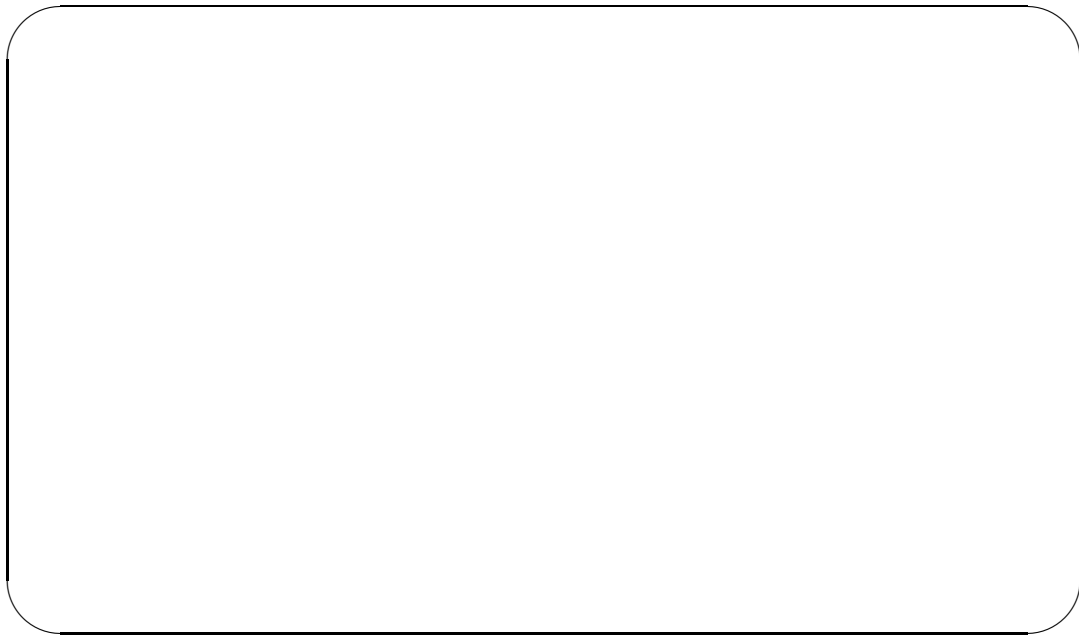
(5 pts.)

Responen les preguntes següents:

(a) (1 pt.) Considereu la funció *mystery*:

```
bool mystery (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;  
}
```

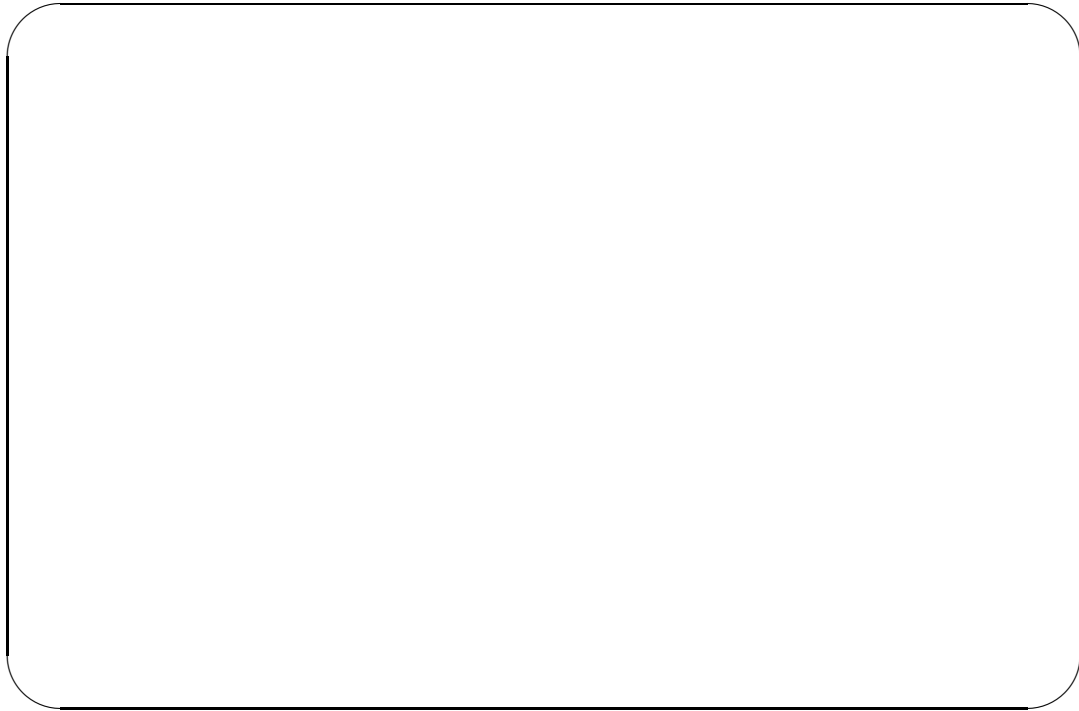
La funció *mystery* determina si  i el cost en el cas pitjor, en funció d' $n$ , és  $\Theta(\text{  })$ . Justifiqueu aquest cost:



(b) (1 pt.) Considereu ara el codi següent:

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

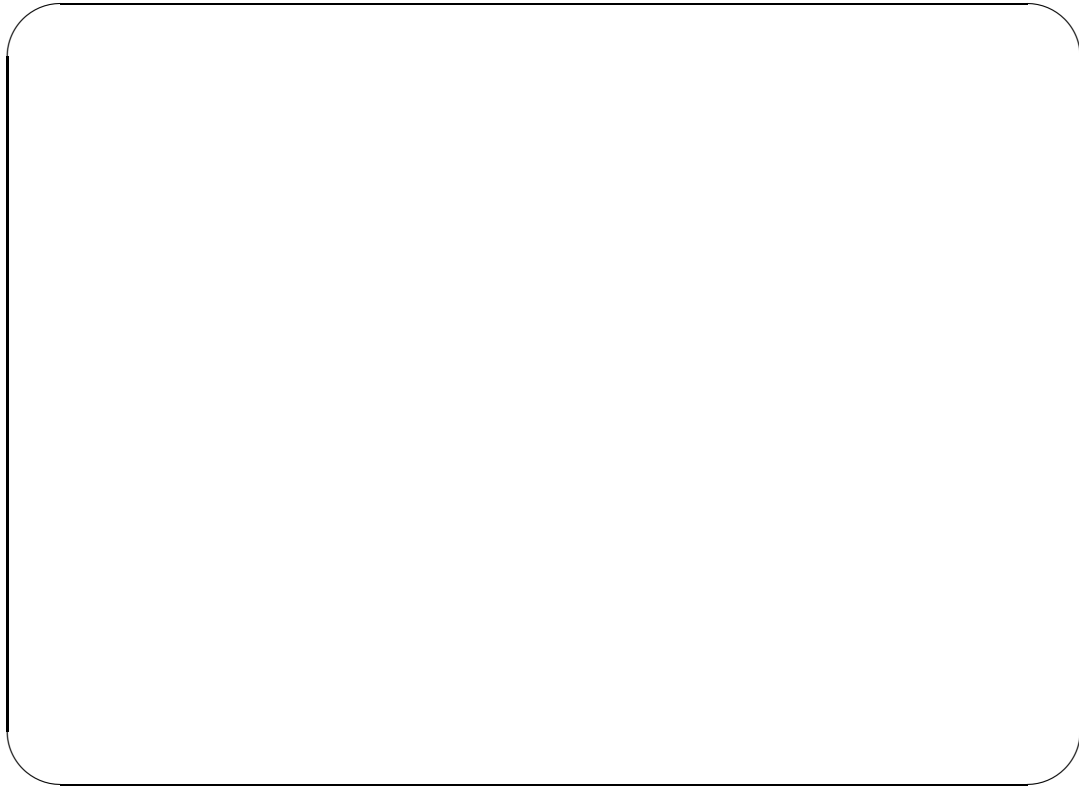
En funció d' $n$ , el cost és  $\Theta(\text{  })$ . Justifiqueu la vostra resposta:



- (c) (1 pts.) Donat un vector  $v$  d' $n$  enters, volem calcular el nombre total de parelles  $(i, j)$  tals que  $0 \leq i < j \leq n - 1$  i  $v[i] = v[j]$ . Per tal de solucionar el problema ens donen el codi següent:

```
int pairs (const vector<int>& v, int l, int r) {
    if (l ≥ r) return 0;
    else {
        int m = (l+r)/2;
        int n_left = pairs(v, l, m);
        int n_right = pairs(v, m+1, r);
        int n_crossed = 0;
        for (int i = l; i ≤ m; ++i)
            for (int j = m+1; j ≤ r; ++j)
                if (v[i] == v[j]) ++n_crossed;
        return n_left + n_right + n_crossed;
    }
}
int pairs (const vector<int>& v) {return pairs(v, 0, v.size() - 1);}
```

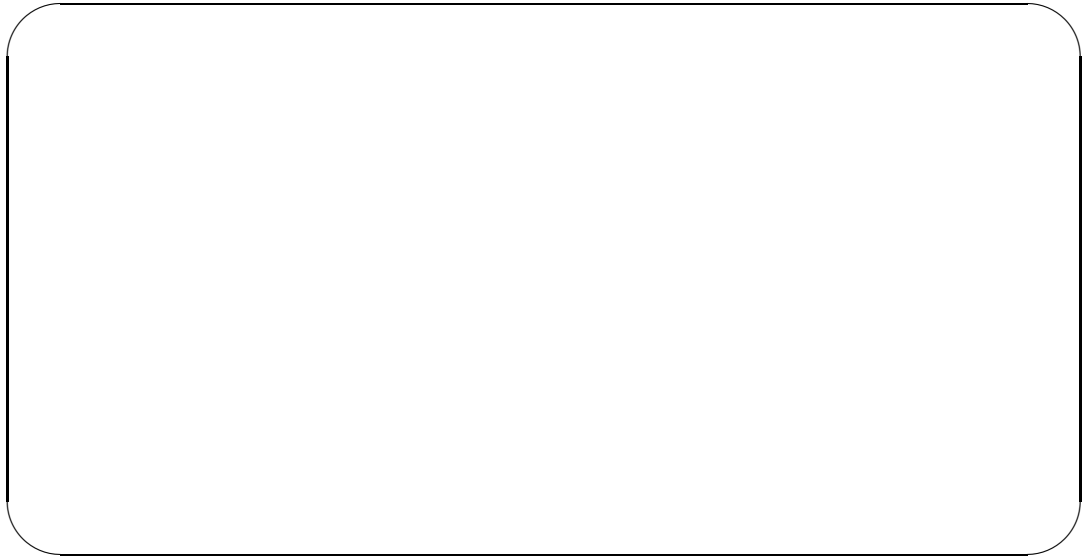
En funció d' $n$ , el cost d'una crida a  $\text{pairs}(v)$  és  $\Theta(\quad)$ . Justifiqueu la vostra resposta:



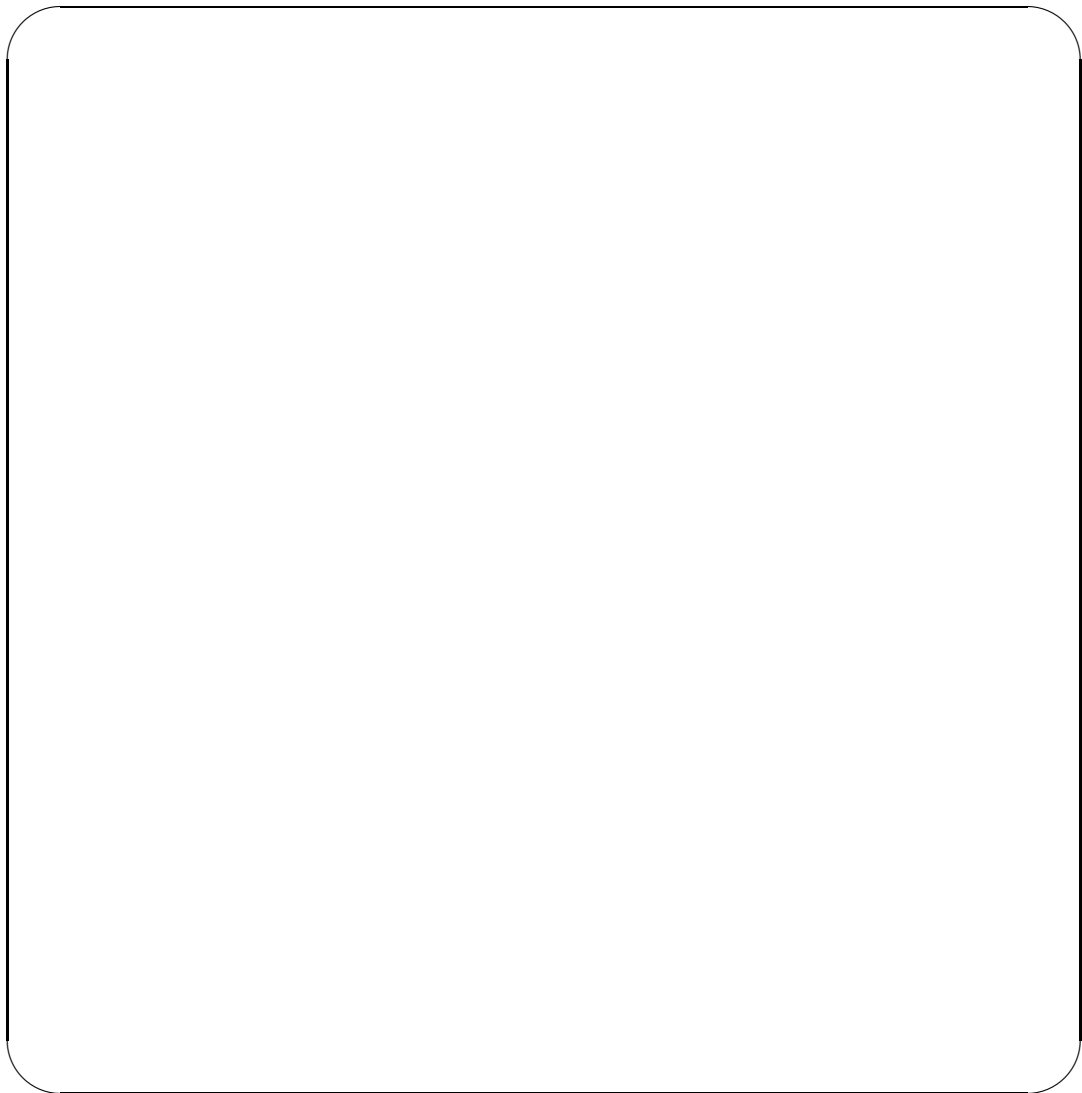
- (d) (2 pts.) Ens asseguren ara que tots els nombres de  $v$  són naturals estrictament menors que un cert paràmetre enter  $K$ , que és constant i no depèn d' $n$ . En aquesta situació, el codi següent és una solució al problema de l'apartat anterior:

```
int pairs_2 (const vector<int>& v) {  
    vector<int> times(K,0);  
    int n = v.size ();  
    for (int i = 0; i < n; ++i) ++times[v[i]];   
  
    int res = 0;  
    for (int i = 0; i < K; ++i) res += (times[i]*(times[i]-1))/2;  
    return res;  
}
```

Si  $n$  és la mida de  $v$ , el cost de `pairs_2` en funció d' $n$  és  $\Theta(\text{ } \boxed{\text{ }} \text{ })$ . Justifiqueu la vostra resposta:

A large, empty rounded rectangular box with a thin black border, intended for writing code.

Expliqueu per què el codi anterior és una solució correcta:

A large, empty rounded rectangular box with a thin black border, intended for writing an explanation.

**Problema 2****(5 pts.)**

Donat un vector  $v$  d' $n$  enters ordenats creixentment i un enter  $x$ , volem determinar el nombre de vegades que  $x$  apareix a  $v$ .

- (a) (1.5 pts.) Diem que la *primera aparició* d' $x$  dins  $v$  és el menor índex  $i$  amb  $0 \leq i < n$  i  $v[i] = x$ , o bé  $-1$  si  $x$  no apareix a  $v$ . Un amic ens comenta que si sabem trobar la primera aparició, aleshores trobar una solució eficient al nostre problema no és massa difícil. Ompliu el codi següent per tal que trobi la primera aparició d' $x$  dins  $v$  de manera que el seu cost en cas pitjor sigui  $O(\log n)$ .

```
int first_occurrence (int x, const vector<int>& v, int l, int r) {
    if (l > r) return -1;
    else {
        int m = (l+r)/2;
        if (v[m] < x) return first_occurrence (x,v,m+1,r);
        else if (v[m] > x) return first_occurrence (x,v,l,m-1);
    }
}
```

```
int first_occurrence (int x, const vector<int>& v) {
    return first_occurrence (x,v,0,v.size()-1);
}
```

- (b) (1.5 pts.) Amb la funció anterior funcionant ja correctament, ens demanen que omplim el codi següent per tal que calculi el nombre d'aparicions d' $x$  dins  $v$ :

```
int p = first_occurrence (x,v);

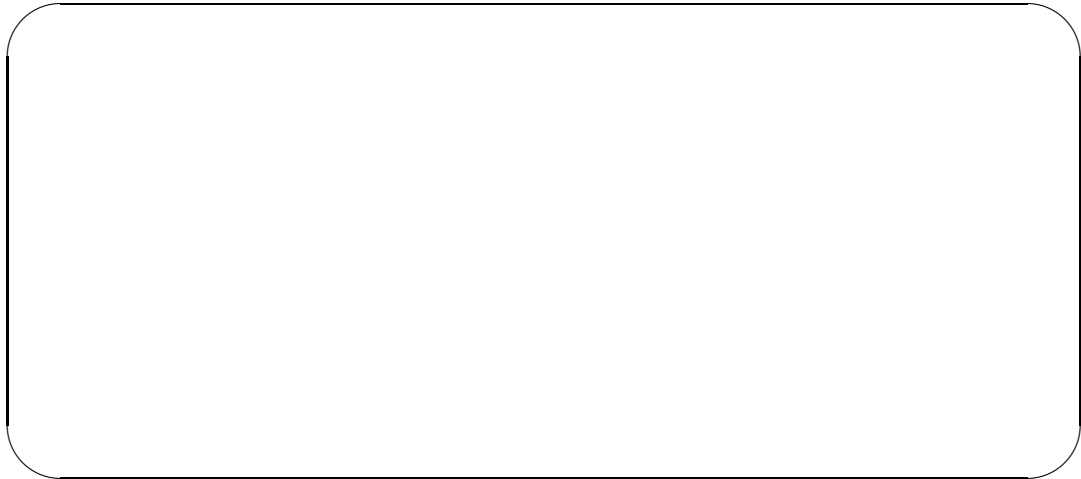
int n = v.size ();
for (int i = 0; i < n; ++i) v[i] = -v[i];
for (int i = 0; i <= n/2 - 1; ++i) swap(v[i], v[n-1-i]);
int q = first_occurrence (-x,v);

int res;
if (p == -1) res = 0;
else res = 

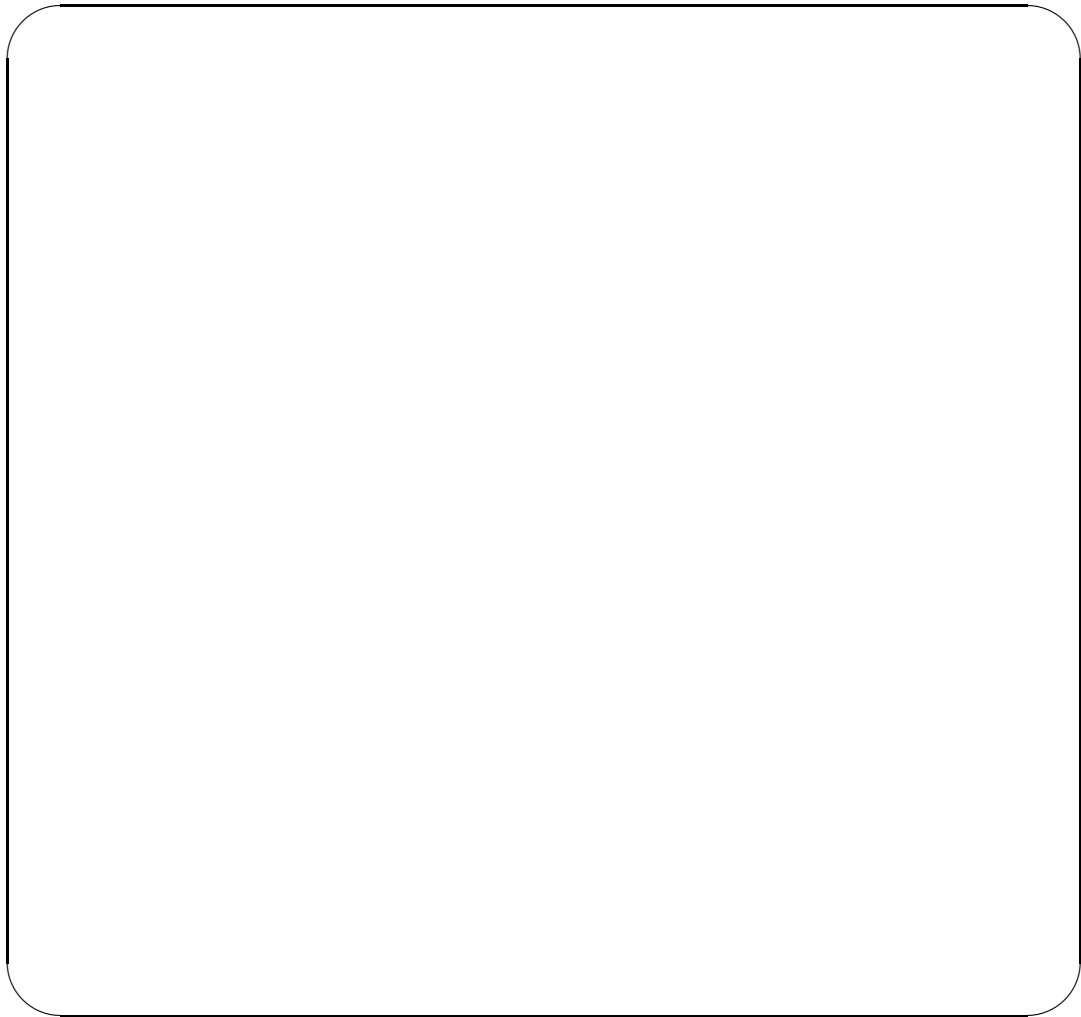
cout << res << endl;
```

Determineu de forma raonada, el cost en cas pitjor del codi anterior en funció d' $n$ .

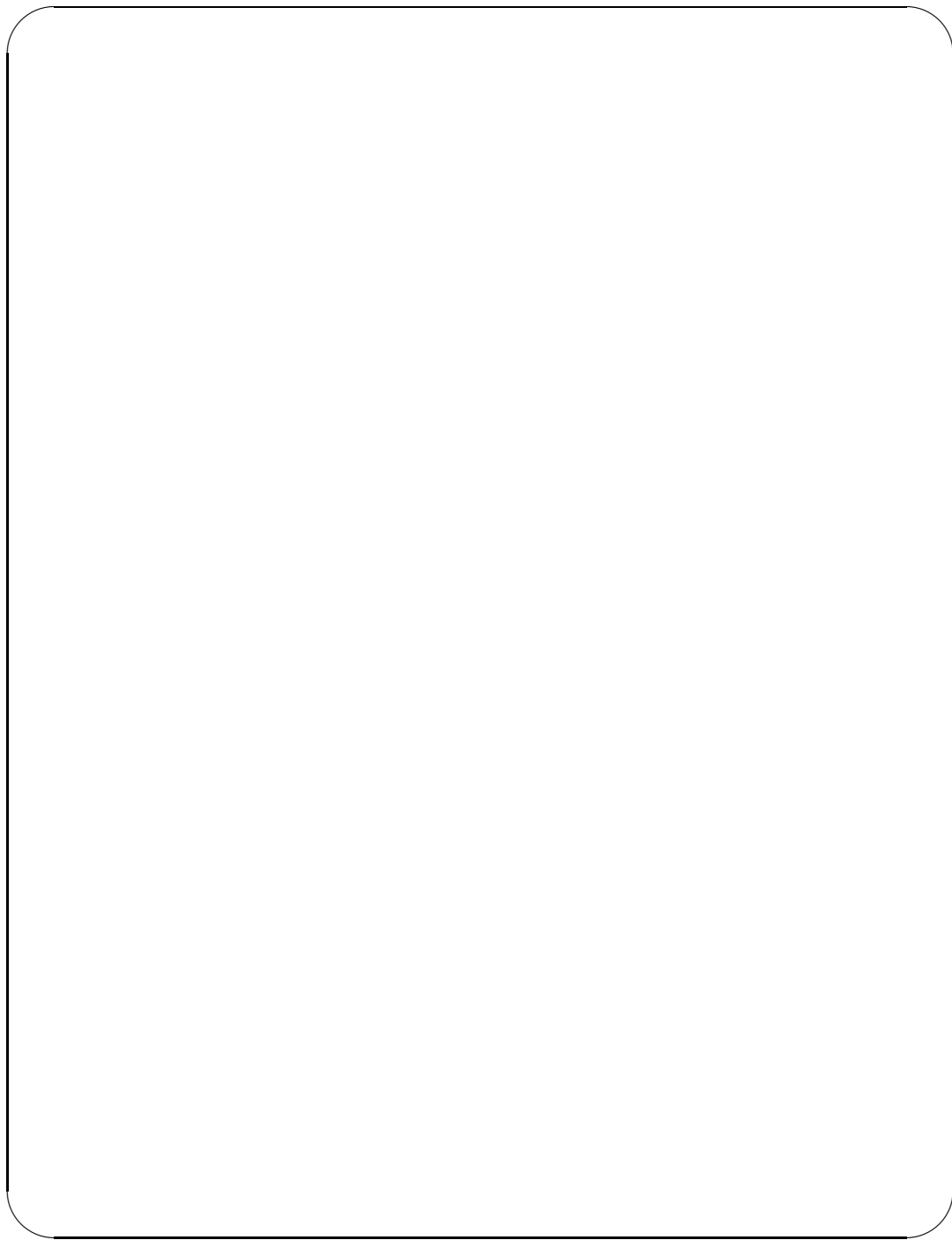




Expliqueu per què el codi anterior calcula correctament el nombre d'aparicions d' $x$  dins  $v$ .



- (c) (2 pts.) Existeix un algorisme per calcular el nombre d'aparicions que sigui, en cas pitjor, asimptòticament més eficient que el codi anterior? Si existeix, expliqueu-lo a alt nivell (no cal codi concret) i analitzeu el seu cost. Si no existeix, expliqueu per què no pot existir.



## Examen Parcial EDA

Duració: 1h 30min

31/03/2022

## Problema 1

(6 pts.)

Responen les preguntes següents:

(a) (1.5 pts.) Considereu el codi següent:

```
int n; cin >> n;
vector<int> v(n);
for (int i = 0; i < n; ++i) v[i] = i+1;
random_shuffle(v.begin(), v.end()); // Theta(n)
int s = 0;
for (int i = 0; i < n; ++i)
    for (int j = 0; j < v[i]; ++j) ++s;
cout << s << endl;
```

Recordem que, donat un vector  $v$ , la instrucció `random_shuffle(v.begin(), v.end())` reordena els elements del vector  $v$  de manera aleatòria en temps lineal en la mida del vector. En funció d' $n$ , què calcula el codi anterior i quin és el seu cost asimptòtic?

(b) (2 pts.) Considereu ara el codi següent:

```
int n; cin >> n;
for (int j = 1; j < n; ++j){
    int k = 2;
    while (k < n) k = k * k;
}
```

En funció d' $n$ , el seu cost és  $\Theta(\text{ } \boxed{\text{ }} \text{ })$ . Justifiqueu la vostra resposta:

(c) (2.5 pts.) Per a qualsevol nombre natural  $n \geq 1$ , definim el vector de naturals següent:

$$(1, 2n, 2, 2n - 1, 3, 2n - 2, 4, 2n - 3, \dots, n, n + 1)$$

En funció d' $n$ , quin és el cost de l'algorisme d'ordenació per inserció sobre aquest vector?

## Problema 2

(4 pts.)

Donat un vector  $v$  d' $n$  naturals diferents i ordenats de forma creixent, volem saber quin és el natural més petit que no apareix a  $v$ . Recordem que el 0 és un natural.

(a) (1.5 pts.) Contactem amb un amic que és ben conegut per proporcionar solucions estranyes i innecessàriament complicades, i ens comenta que la funció *inefficient* soluciona aquest problema:

```
bool find (int x, const vector<int>&v, int pos){
    if (pos < 0) return false;
    return v[pos] == x or find(x,v,pos-1);
}

int inefficient (const vector<int>& v) {
    int n = v.size ();
    for (int i = 0; i < n; ++i)
        if (not find(i,v,n-1)) return i;
    return n;
}
```

En funció d' $n$ , quin és el cost en cas pitjor d'una crida a la funció *inefficient*?

- (b) (2.5 pts.) Doneu vosaltres una funció que solucioni aquest problema i que trigui, en cas pitjor,  $\Theta(\log n)$ .

```
int efficient (const vector<int>& v, int l, int r) {
```

```
}
```

```
int efficient (const vector<int>& v) {  
    return efficient (v,0,v.size ()-1);  
}
```

Examen Parcial EDA

Duració: 1h 30min

03/11/2022

## Problema 1

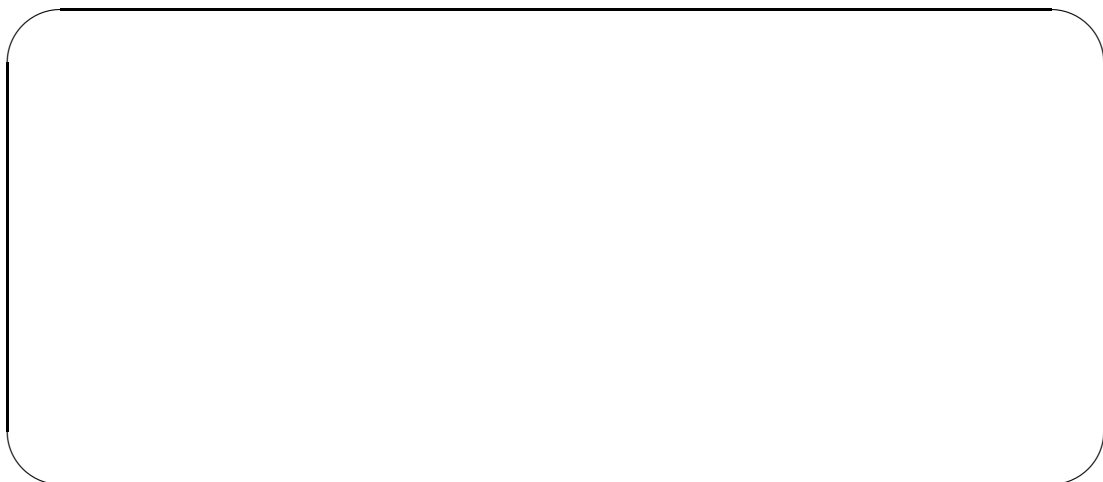
(4 pts.)

Responen les preguntes següents:

(a) (2 pts.) Considereu el codi següent:

```
int f (int x, int n) {  
    if (n == 1) return x;  
    else {  
        int tmp = f(x,n-1);  
        int res = 0;  
        for (int i = 0; i < x; ++i) res += tmp;  
        return res;  
    }  
}  
  
int main(){  
    int N;  
    cin >> N;  
    cout << f(N,N) << endl;  
}
```

Si assumim que  $N \geq 1$ , què escriu per pantalla el programa anterior? Justifica la teva resposta.



Quin és el cost del programa anterior en funció d' $N$ ?

(b) (2 pts.) Siguin  $f(n) = \ln(\ln(n^2))$  i  $g(n) = \ln(\ln n)$ . Podem afirmar que  $f(n) \in \Theta(g(n))$ ?

Definim ara

$$F(n) = \begin{cases} n^2, & \text{si } 0 \leq n \leq 10 \\ f(n) & \text{si } n > 10 \end{cases} \quad G(n) = \begin{cases} n^3, & \text{si } 0 \leq n \leq 10 \\ g(n) & \text{si } n > 10 \end{cases}$$

on  $f$  i  $g$  són les funcions anteriorment definides. Podem afirmar que  $F(n) \in \Theta(G(n))$ ?

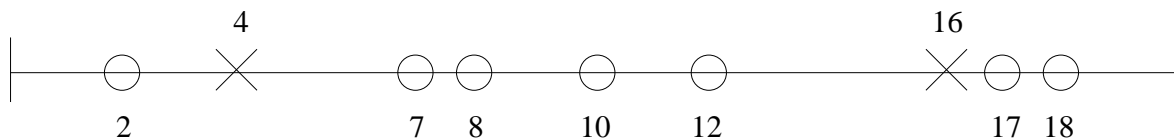
**Problema 2****(6 pts.)**

L'ajuntament d'una gran ciutat decideix comprar estufes de carrer per tal d'escalfar totes les escoles del municipi. Totes les escoles estan situades sobre una mateixa recta, i disposem d'un vector  $s$  d'enters amb les distàncies de totes elles al punt quilomètric zero. Sabem que els elements d' $s$  són tots diferents.

Els assessors en matèria energètica han decidit ja quantes estufes comprar i en quin punt de la recta situar-les. Disposem d'un vector  $h$  que indica el punt quilomètric en la recta de cadascuna de les estufes. Altra vegada, tots els elements d' $h$  són diferents.

Per estalviar energia, podem ajustar les estufes pes tal d'escalfar qualsevol escola que estigui a distància com a molt  $d$ . Si volem ajustar totes les estufes de la mateixa manera, quina és la mínima  $d$  que ens permet escalfar totes les escoles?

Gràficament, si  $s = [8, 17, 2, 12, 18, 7, 10]$  i  $h = [16, 4]$  tenim la situació



i podem veure que la solució és  $d = 6$ . Si prenem  $d = 5$ , per exemple, l'escola a la posició 10 no seria escalfada per cap estufa.

Per simplificar els raonaments, assumiren en tot aquest problema que  $n = s.size() = h.size()$ .

(a) (0.5 pts.) La següent funció ens proporciona una solució senzilla al problema:

```
int radius (const vector<int>& h, const vector<int>& s) {
    int rad = 0;
    for (int i = 0; i < s.size (); ++i) {
        int rad_s = inf; // inf és l'int més gran
        for (int j = 0; j < h.size (); ++j)
            rad_s = min(rad_s, abs(h[j] - s[i]));
        rad = max(rad, rad_s);
    }
    return rad;
}
```

En funció d' $n$ , quin és el cost d'una crida a aquesta funció?



- (b) (2 pts.) Assumim en aquest apartat que tant  $h$  com  $s$  estan ordenats de manera creixent. Ompliu els buits de la funció següent perquè resolgui el problema que tenim entre mans:

```

int radius_2 (const vector<int>& h, const vector<int>& s) {
    int r = 0, j = 0;
    for (int i = 0; i < s.size (); ++i){
        while (j < h.size () and h[j] < s[i]) ++j;
        int rad;
        if (j == h.size ()) rad = 
        else if (j == 0) rad = 
        else rad = 
        r = max(r,rad);
    }
    return r;
}

```

Quin és el cost en cas pitjor d'una crida a *radius\_2* en funció d' $n$ ?

- (c) (3 pts.) Assumim en aquest apartat que  $h$  està ordenat de forma creixent. Donada una escola a la posició  $p$ , volem trobar la mínima  $k$  tal que  $p \leq h[k]$  i  $0 \leq k < n$ . Si no existeix cap  $k$  que compleixi aquestes dues condicions (és a dir, si  $p$  és més gran que qualsevol element d' $h$ ) cal retornar  $n$ .

Ompliu els buits de la funció següent perquè retorni aquest valor  $k$  en temps  $\Theta(\log n)$  en cas pitjor. Solucions que no tinguin aquest cost rebran zero punts.

```

int find (const vector<int>& h, int p) {return find(h, 0, h.size ()-1, p);}

```

```
int find(const vector<int>& h, int l, int r, int p) {  
    if (r < l) {
```

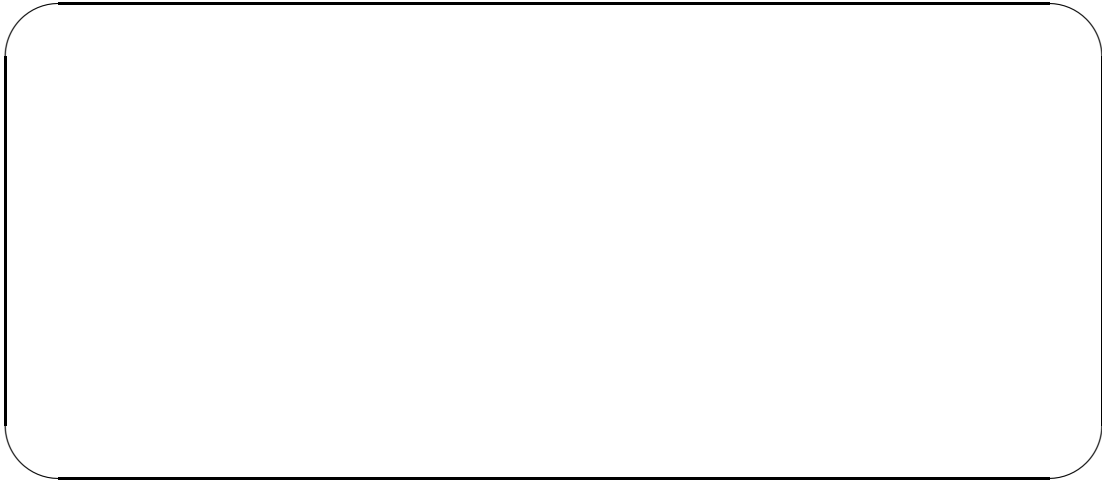
```
    }
```

```
    else {
```

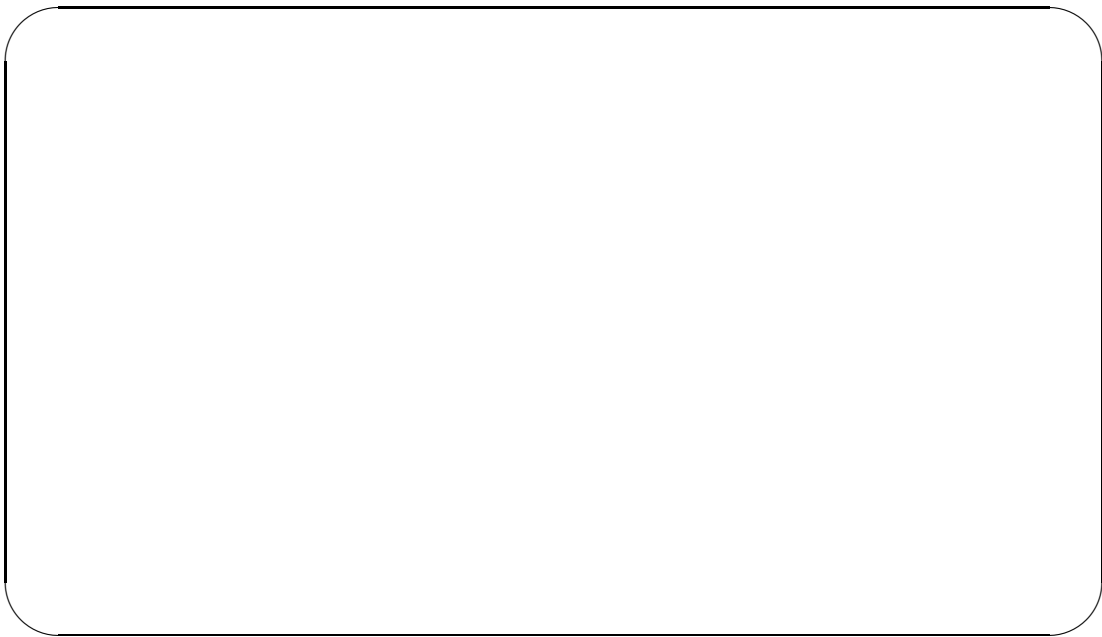
```
    }
```

```
}
```

Justifica per què la teva solució té cost, en cas pitjor,  $\Theta(\log n)$ .



- (d) (0.5 pts.) Si en la funció *radius\_2* de l'apartat (b) reemplacem la línia del **while** per  $j = \text{find}(h, s[i])$ ; quin seria el cost en cas pitjor d'una crida a *radius\_2* en funció d' $n$ ?



## Examen Parcial EDA

Duració: 2h

21/04/2023

## Problema 1

(4.5 pts.)

Donats dos vectors de nombres naturals  $v_1$  i  $v_2$  de mida  $n > 0$ , volem determinar si podem trobar un element a cada vector tal que, si els intercanviem, els dos vectors resultants sumen el mateix. Per exemple, si  $v_1 = (6, 4, 3, 9)$  i  $v_2 = (7, 0, 2, 5)$ , aleshores intercanviant el 6 i el 2 obtenim dos vectors que sumen el mateix. En canvi, si  $v_1 = (2, 0, 9, 5)$  i  $v_2 = (2, 4, 5, 7)$  no existeix cap parell d'elements amb la propietat desitjada.

(a) (1 pt.) Considereu la solució següent al problema plantejat:

```
int suma (const vector<int>& v){
    int s = 0;
    for (int x : v) s += x;
    return s;
}

pair<int,int> sol_facil (vector<int>& v1, vector<int>& v2) {
    for (int i = 0; i < v1.size (); ++i)
        for (int j = 0; j < v2.size (); ++j) {
            swap(v1[i], v2[j]);
            int s1 = suma(v1);
            int s2 = suma(v2);
            swap(v1[i], v2[j]);
            if (s1 == s2) return {v1[i], v2[j]};
        }
    return {-1, -1}; // No hi ha solució
}
```

En funció d' $n$ , quin és el cost en el cas pitjor d'una crida a *sol\_facil*?

- (b) (2 pts.) Completeu el codi següent per tal que sigui una solució vàlida al problema plantejat:

```
bool cerca (const vector<int>& v, int x, int e, int d) {  
    if (e > d) return false;  
    else {  
        int m = (e+d)/2;  
        return v[m] == x or cerca(v,x,e,m-1) or cerca(v,x,m+1,d);  
    }  
}  
  
pair<int,int> sol (const vector<int>& v1, const vector<int>& v2) {  
    int dif = suma(v2) - suma(v1); // suma és la funció de l'apartat anterior  
    if (dif%2 != 0) return {-1,-1};  
    for (int i = 0; i < v1.size (); ++i) {  
        int x =   
        if (cerca(v2, x, 0, v2.size ()-1)) return {v1[i], x};  
    }  
    return {-1,-1};  
}
```

Analitzeu, en funció d' $n$ , el cost en el cas pitjor d'una crida a *sol*.

- (c) (1.5 pts.) Expliqueu com modificaríeu la solució de l'apartat anterior per tal que el seu cost, en el cas pitjor, sigui millor asimptòticament. No cal que doneu codi concret, una descripció a alt nivell serà suficient. Quin és el cost, en el cas pitjor, de la nova solució?

## Problema 2

(5.5 pts.)

Donat dos nombres naturals  $n, k$  diferents de zero, volem expressar  $n$  com la suma ordenada d'exactament  $k$  potències de 2 amb exponent no negatiu (és a dir,  $2^{-3}$  no la considerem una potència de 2 vàlida). Per exemple, si  $n = 21$  i  $k = 4$ , una solució és  $21 = 2^3 + 2^3 + 2^2 + 2^0$ . Fixem-nos que l'ordre desitjat és decreixent.

*Observació:* la representació en binari d' $n$  ens dona una manera d'expressar  $n$  com a suma de potències de 2, però pot no tenir exactament  $k$  sumands. De fet, és la representació amb el menor nombre de sumands possible.

(a) (0.75 pts.) Escriviu solucions per a  $n = 10$  i  $k = 2, 3, 4$  i 5. No cal justificar com les obteniu.

(b) (1.25 pts.) Quines són les dues úniques situacions on no hi ha solució?

(c) (1.5 pts.) Completeu la funció següent per tal que donat un natural  $n$  retorni, ordenades de menor a major, les posicions de la representació en binari d' $n$  on apareix un 1. Per exemple, si  $n = 19$ , cal retornar el vector  $(0, 1, 4)$ .

```
vector<int> pos_uns (int n) {  
    vector<int> v;  
    int pos = 0;
```

```
    return v;
}
```

Quin és el seu cost en funció d' $n$ ?

(d) (2 pts.) Completeu el codi següent per tal que resolgui el problema plantejat:

```
void escriu_suma_potencies (int n, int k) {
    vector<int> uns = pos_uns(n);
    if (  )
        cout << "No hi ha solucio" << endl;
    else {
        priority_queue<int> Q;
```

```
        bool primer = true;
        while (not Q.empty()){
            if (not primer) cout << " + ";
            else primer = false;
            cout << "2^" << Q.top();
            Q.pop();
        }
        cout << endl;
    }
}
```

## Solució de l'Examen Parcial EDA - torn 1

18/10/2010

## Proposta de solució al problema 1

- Doneu la definició de  $O(f)$ :

$$O(f) = \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |g(n)| \leq c|f(n)|\}$$

- El teorema mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma  $T(n) = aT(n/b) + \Theta(n^k)$  amb  $b > 1$  i  $k \geq 0$ , llavors, fent  $\alpha = \log_b a$ ,

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

## Proposta de solució al problema 2

Ompliu els blancs de la forma més precisa possible.

- Quicksort ordena  $n$  elements en temps  $\Theta(n^2)$  en el cas pitjor.
- Per multiplicar dues matrius grans eficientment podem fer servir l'algorisme de Strassen.
- $2 + \cos(n) = \Theta(1)$ .
- $2\sqrt{n} + 1 + n + n^2 = \Theta(n^2)$ .
- $\log n + \log \log(n^2) = \Theta(\log n)$ .
- $\frac{n^2 - 6n}{2} + 5n = \Theta(n^2)$ .
- $n^2 - 3n - 18 = \Omega(n)$ .
- Si

$$T(n) = \begin{cases} 4 & \text{si } n = 1, \\ 3T(n/2) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(n^2)$ .

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 2T(n-1) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(2^n)$ .



**Proposta de solució al problema 3**

Sigui  $c \in \mathbb{R}$  i siguin  $f$  i  $g$  dues funcions de  $\mathbb{N}$  a  $\mathbb{R}$ , si  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$  aleshores, per definició, tenim que  $\forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : \left| \frac{f(n)}{g(n)} - c \right| \leq \epsilon$ . Fixant un  $\epsilon > 0$  arbitrari, per propietats del valor absolut si  $\left| \frac{f(n)}{g(n)} - c \right| \leq \epsilon$  aleshores  $\left| \frac{f(n)}{g(n)} \right| - |c| \leq \epsilon$  i en particular es compleix que  $|f(n)| \leq (|c| + \epsilon)|g(n)|$ . Per tant,

$$\exists c' = |c| + \epsilon > 0 : \exists m > 0 : \forall n \geq m : |f(n)| \leq c'|g(n)| \implies f \in O(g).$$

**Proposta de solució al problema 4**

Suposeu que *primer*( $n$ ) és una crida a una funció amb temps  $O(\sqrt{n})$ .

Considereu un procediment amb cos principal:

```
if (primer(n)) A;
else B;
```

Doneu fites senzilles i ajustades amb notació  $O$  per al temps d'aquest procediment, en funció de  $n$ , suposant que:

1.  $A$  triga  $O(n)$  i  $B$  triga  $O(1)$ .

$$\boxed{O(n)}$$

2.  $A$  i  $B$ , ambdòs, triguen  $O(1)$ .

$$\boxed{O(\sqrt{n})}$$

**Proposta de solució al problema 5**

```
int misteri (int n) {
    if (n == 1) return 1;
    return misteri (n-1) + 2*n - 1;
}
```

- Digueu què calcula la funció *misteri*.

$$\boxed{n^2}$$

- Doneu el seu cost en funció de  $n$ .

$$\boxed{\Theta(n)}$$

**Proposta de solució al problema 6**

Per obtenir la unió de  $A$  i  $B$  s'ha d'ordenar el vector  $B$  amb un algorisme de temps  $O(m \log m)$  en el cas pitjor (com per exemple mergesort) i cercar-hi els  $n$  elements de  $A$  amb una cerca dicotòmica. S'ha de copiar al resultat tot el conjunt  $B$  i a continuació els elements de  $A$  que no s'hagin trobat a  $B$ . El temps és  $O((n + m) \log m) = O(n \log m)$  perquè  $m \leq n$ .

Si ens demanen la intersecció, el vector  $B$  s'ordena igualment i s'han de copiar al resultat només aquells elements de  $A$  que siguin a  $B$ .

**Proposta de solució al problema 7**

L'algorisme és mergesort (ordenació per fusió) però divideix la taula d'elements a ordenar en tres parts de mida semblant i fa tres crides recursives a cada subtaula en comptes de dividir en meitats i fer dues crides recursives. El seu cost en temps,  $T(n)$ , ve donat per la recurrència:

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1, \\ 3T(n/3) + \Theta(n) & \text{si } n > 1. \end{cases}$$

i per tant  $T(n) = \Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA- torn 2

18/10/2010

## Proposta de solució al problema 1

- Doneu la definició de  $\Omega(f)$ :

$$\Omega(f) = \{g: \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |g(n)| \geq c|f(n)|\}$$

- El teorema mestre de resolució de recurrències substractores afirma que si tenim una recurrència de la forma  $T(n) = aT(n - c) + \Theta(n^k)$  amb  $c > 0$  i  $k \geq 0$ , llavors,

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

## Proposta de solució al problema 2

Ompliu els blancs de la forma més precisa possible.

- Per multiplicar dos naturals molts llargs eficientment podem fer servir l'algorisme de Karatsuba.
- Per ordenar  $n$  elements en temps  $\Theta(n \log n)$  en el cas pitjor podem servir l'algorisme de mergesort (ordenació per fusió).
- Multipliqueu  $\log n + 6 + O(1/n)$  per  $n + O(\sqrt{n})$ . El resultat simplificat tan com és possible és  $O(\underline{n \log n})$ .
- $\sin(n) + 10 + n = \Theta(\underline{n})$ .
- $\frac{1}{3}n^2 + 3n \log n + 5n^8 = \Theta(\underline{n^8})$ .
- $n\sqrt{n} + n^2 = \Theta(\underline{n^2})$ .
- $3n \log n = \underline{O}(n^2)$ .
- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 3T(n/2) + n - 2 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(\underline{n^{\log_2(3)}})$ .

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 4T(n-1) + n & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(\underline{4^n})$ .

**Proposta de solució al problema 3**

Sigui  $c \in \mathbb{R}$  i siguin  $f$  i  $g$  dues funcions de  $\mathbb{N}$  a  $\mathbb{R}$ , si  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$  aleshores,  $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \frac{1}{c} > 0$  i per definició, tenim que  $\forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : \left| \frac{g(n)}{f(n)} - \frac{1}{c} \right| \leq \epsilon$ . Fixant un  $\epsilon > 0$  arbitrari, per propietats del valor absolut si  $\left| \frac{g(n)}{f(n)} - \frac{1}{c} \right| \leq \epsilon$  aleshores  $\left| \left| \frac{g(n)}{f(n)} \right| - \left| \frac{1}{c} \right| \right| \leq \epsilon$  i en particular es compleix que  $|f(n)| \geq \frac{c}{c\epsilon+1}|g(n)|$ . Per tant,

$$\exists c' = \frac{c}{c\epsilon+1} > 0 : \exists m > 0 : \forall n \geq m : |f(n)| \geq c'|g(n)| \implies f \in \Omega(g).$$

**Proposta de solució al problema 4**

```

int misteri (int m, int n) {
    int result = 0;
    while (m > 0) {
        if (m % 2 != 0) result += n;
        m /= 2; n *= 2;
    }
    return result ;
}

```

- Diguen què calcula la funció *misteri*.

És l'algorisme de la multiplicació russa i calcula  $m \times n$

- Doneu el seu cost en funció de  $m$ .

$\Theta(\log m)$

**Proposta de solució al problema 5**

Considereu un procediment amb cos principal:

```

int k = f(n);
int s = 0;
for (int i = 1; i ≤ k; ++i) s += i;

```

amb  $f(n)$  una crida a la funció  $f$ .

Doneu fites senzilles i ajustades amb notació  $O$  per al temps d'aquest procediment, en funció de  $n$ , suposant que:

1. El temps de  $f(n)$  és  $O(n)$  i el valor de  $f(n)$  és  $n!$ .

$O(n!)$

2. El temps de  $f(n)$  és  $O(n)$  i el valor de  $f(n)$  és  $n$ .

$O(n)$

3. El temps de  $f(n)$  és  $O(n^2)$  i el valor de  $f(n)$  és  $n!$ .

$O(n!)$

4. El temps de  $f(n)$  és  $O(1)$  i el valor de  $f(n)$  és 0.

$$\boxed{O(1)}$$

#### Proposta de solució al problema 6

1. Amb un vector ordenat, si un element apareix repetit més de  $n/2$  vegades aquest element s'ha de trobar a la posició del mig. En temps  $O(\log n)$  (amb dues cerques dicotòmiques) es poden trobar la posició de la primera aparició d'aquest element a la taula i la posició de la darrera. Fent la resta de la segona menys la primera trobem el nombre de vegades que es repeteix aquest element i podem determinar si això passa més de  $n/2$  vegades.
2. Si el vector és de 3 elements (qualsevol constant pot servir) es calcula directament provant totes les possibilitats si hi ha un element repetit dues vegades o més. Si  $n > 3$  aleshores es divideix el vector  $A$  per la meitat en dos subvectors  $A_1$  i  $A_2$  de (quasi) la mateixa mida i es crida l'algorisme recursivament en  $A_1$  i  $A_2$ . Si  $A_1$  conté un element majoritari (com demana l'enunciat) es compara aquest element amb tots els de  $A_2$ , el que dona el nombre d'aparicions d'aquest element a  $A$ . Es segueix el mateix procediment amb  $A_2$ . Si en algun dels dos casos obtenim un element repetit més de  $n/2$  vegades, retornem aquest element i el nombre de vegades que es repeteix. En cas contrari no hi ha elements que compleixin amb la propietat.

#### Proposta de solució al problema 7

L'algorisme amagat és quicksort (ordenació ràpida) i quan tots els elements són iguals per cada increment de l'índex  $j$  hi ha un increment de l'índex  $i$  i es fa un intercanvi de cada element del vector amb sí mateix, al final de cada crida recursiva ambdòs índexos ténen per valor  $d$ , i es fan dues crides recursives a problemes de mida  $n - 1$  i  $1$  respectivament. Per tant, el cost en temps,  $T(n)$ , de l'algorisme anterior ve donat per la recurrència:

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1, \\ T(n-1) + \Theta(n) & \text{si } n > 1. \end{cases}$$

i per tant  $T(n) = \Theta(n^2)$ .

## Solució de l'Examen Parcial EDA

21/03/2011

## Proposta de solució al problema 1

Ompliu els blancs de la forma més precisa possible.

- Insertion sort ordena  $n$  elements en temps  $\Theta(n)$  en el cas millor.
- Trobar la mediana de  $n$  elements té una complexitat en temps  $\Omega(\sqrt{n})$ .
- Com de ràpid es poden multiplicar una matriu  $kn \times n$  per una  $n \times kn$  fent servir l'algorisme de Strassen?  $\Theta(k^2 n^{\log_2(7)})$ .
- Sigui  $F(n)$  el número de línies que imprimeix el programa anterior amb entrada  $n$ , aleshores,

$$F(n) = \begin{cases} \Theta(1) & \text{si } n \leq 1, \\ 2F(n/2) + \Theta(1) & \text{si } n > 1. \end{cases}$$

i per tant  $F(n) = \Theta(n)$ .

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n < 3, \\ 9T(n/3) + n^2 & \text{si } n \geq 3. \end{cases}$$

Aleshores,  $T(n) = \Theta(n^2 \log(n))$ .

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n = 1, \\ 3T(n-1) + n^3 & \text{si } n > 1. \end{cases}$$

Aleshores,  $T(n) = \Theta(3^n)$ .

- Siguin  $a = b = c = d = 01$ ,  $u = (a + b)(c + d) = 10 \times 10$ ,  $v = ac = 01 \times 01 = 1$  i  $w = bd = 01 \times 01 = 1$ . Per calcular el valor de  $u$  fem servir el mateix procediment amb  $a' = c' = 1$ ,  $b' = d' = 0$ ,  $u' = (1 + 0) \times (1 + 0) = 1$ ,  $v' = 1 \times 1 = 1$ , i  $w' = 0 \times 0 = 0$ ,  $u' - v' - w' = 0$ , aleshores  $u = v' \times 100 = 100$  i  $u - v - w = 10$ . Per tant,  $101 \times 101 = 1 \times 10000 + 10 \times 100 + 1 = 11001$ .

2	3	5	10	1	6	7	13	mergesort
2	3	5	10	7	13	1	6	insertion sort
1	2	5	3	7	6	13	10	quicksort
1	2	3	5	6	13	10	7	selection sort

A	B	O	$\Omega$	$\Theta$
$\log^k(n)$	$n^\epsilon$	sí	no	no
$\log n$	$\log \log(n^2)$	no	sí	no
$n^k$	$c^n$	sí	no	no
$2^{n+1}$	$2^n$	sí	sí	sí
$2^{2n}$	$2^n$	no	sí	no

**Proposta de solució al problema 2**

- El programa troba el  $k$ -èsim element més petit de taula  $T$ .
- El cas pitjor consisteix a no poder reduir la talla del problema a resoldre recursivament en més d'un element i fer una crida recursiva a una taula amb un únic element menys. D'aquesta manera es faran el màxim nombre possible de crides recursives. Si  $T(n)$  és el cost en cas pitjor de l'algorisme anterior aleshores,

$$T(n) = \begin{cases} \Theta(1) & \text{si } n \leq 1, \\ T(n-1) + \Theta(n) & \text{si } n > 1, \end{cases}$$

on  $\Theta(n)$  és el cost no recursiu de cridar a *misteri\_2*.

- En el cas millor per retornar el  $k$ -èsim valor més petit només calen tres crides recursives a *misteri\_1* si  $k > 1$ , i dues si  $k = 1$ . Si  $k > 1$ , aquest cas millor es produeix quan en la 1a crida resulta que  $q = k - 2$ . Llavors la 2a crida es fa amb  $k = 1$ , i si aquí la partició fa que només quedi una finestra amb un sol element, aleshores la 3a crida recursiva retorna pel cas base el valor buscat. Com que en qualsevol cas es fan 2-3 crides a *misteri\_2*, que és lineal, finalment resulta que el cost en el cas millor és  $\Theta(n)$ .
- A quicksort perquè la funció *misteri\_2* és la mateixa partició que fa servir quicksort.

**Proposta de solució al problema 3**

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

```
using namespace std;
```

```
int sqrt(int n, int l, int u) {
    assert (l*l <= n and n <= u*u);
    if (l+1 >= u) {
        if (u*u == n) return u;
        else return l;
    }
    else {
        int m = (l+u)/2;
        int sq = m*m;
        if (sq == n) return m;
        else if (sq < n) return sqrt(n, m, u);
        else return sqrt(n, l, m);
    }
}
```

```
int main(void) {
    int n;
    cin >> n;
    cout << sqrt(n, 0, n) << endl;
}
```

El cas pitjor es produeix quan  $n$  no és un quadrat. Si  $T(n)$  és el temps que es triga en aquest cas, tenim la recurrència  $T(n) = T(n/2) + \Theta(1)$  i per tant  $T(n) = \Theta(\log n)$ .

## Solució de l'Examen Parcial EDA

20/10/2011

## Proposta de solució al problema 1

- Atès que  $\sum_{i=1}^n i = n(n+1)/2$ ,  $101 + 102 + \dots + 999 + 1000 = \sum_{i=101}^{1000} i = \sum_{i=1}^{1000} i - \sum_{i=1}^{100} i = 1000 \cdot 1001/2 - 100 \cdot 101/2 = 500500 - 5050 = 495450$ .
- Establir un "com a mínim" fent servir la notació asimptòtica  $O$  no té sentit, perquè  $O$  no expressa fites inferiors, només superiors. En particular,  $O(n^2)$  inclou funcions que creixen tan lentament com vulguem.
- Ordenades de manera creixent:  $5\log(n+100)^{10}, (\ln(n))^2, \sqrt{n}, 0.001n^4 + 3n^3 + 1, 3^n, 2^{2n}$
- Siguin  $T_A, T_B, T_C$  els costos en el cas pitjor dels algorismes A, B, C.
  - $T_A(n) = 5T_A(n/2) + \Theta(n) = \Theta(n^{\log_2 5})$
  - $T_B(n) = 2T_B(n-1) + \Theta(1) = \Theta(2^n)$
  - $T_C(n) = 9T_A(n/3) + \Theta(n^2) = \Theta(n^2 \log n)$

L'algorisme C és el més eficient.

- El cas millor es dona quan  $x$  és un nombre parell. En aquest cas no hi ha carry i la suma es fa en temps  $\Theta(1)$ .

El cas pitjor es dona quan  $x$  és  $2^{n-1} - 1$ . En aquest cas el  $(n-1)$ -èsim bit de  $x$  és 0, i la resta són 1. De forma que quan es suma 1, el carry es propaga per tota la cadena de bits fins a posar el  $(n-1)$ -èsim bit a 1. Per tant, el cost és  $\Theta(n)$ .

## Proposta de solució al problema 2

- Atès que es recorren senceres les dues taules fent operacions de cost constant, el cost de `fusio2` és  $\Theta(na + nb)$ .
- A la  $i$ -èsima fusió, estem fusionant un vector de mida  $in$  amb un de mida  $n$ . Seguint l'apartat anterior, això té cost  $\Theta((i+1)n)$ . Així doncs, el cost d'aquest algorisme és  $\sum_{i=1}^{k-1} \Theta((i+1)n) = \sum_{i=2}^k \Theta(in) = \Theta(n \cdot \sum_{i=2}^k i) = \Theta(nk^2)$ .
- La funció seria:

```
vector<int> fusiok(const vector< vector<int> >& t, int e, int d) {
    if (e == d) return t[e];
    else {
        int m = (e + d)/2;
        return fusio2(fusiok(t, e, m), fusiok(t, m+1, d));
    }
}

vector<int> fusiok(const vector< vector<int> >& t) {
    return fusiok(t, 0, t.size()-1);
}
```

El cost segueix la recurrència  $T(k, n) = 2T(k/2, n) + \Theta(nk)$ , que té solució  $T(k, n) = \Theta(nk \log k)$ .



**Proposta de solució al problema 3**

Demostrem la identitat per inducció en  $n \geq 1$ . El cas base  $n = 1$  és cert ja que  $f_2 = 1$ ,  $f_1 = 1$  i  $f_0 = 0$ . Suposem ara que la identitat és certa per  $n \geq 1$  i la demostrarem per  $n + 1$ . Tenim:

$$\begin{aligned} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} f_{n+1} + f_n & f_{n+1} \\ f_n + f_{n-1} & f_n \end{pmatrix} \\ &= \begin{pmatrix} f_{n+2} & f_{n+1} \\ f_{n+1} & f_n \end{pmatrix}. \end{aligned}$$

Per dissenyar l'algorisme de cost  $\Theta(\log n)$  farem servir la recurrència següent per calcular potències d'una matriu  $M$ :

$$M^n = \begin{cases} I & \text{si } n = 0 \\ (M^{\lfloor n/2 \rfloor})^2 & \text{si } n > 0 \text{ i és parell} \\ (M^{\lfloor n/2 \rfloor})^2 \cdot M & \text{si } n > 0 \text{ i és senar} \end{cases}$$

Aplicarem la recurrència a la matriu

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

i retornarem  $M^n[0][1]$ . Donat que  $M$  és una matriu  $2 \times 2$ , el cost de calcular aquesta recurrència és  $T(n) = T(n/2) + \Theta(1)$ , i per tant  $T(n) = \Theta(\log n)$  pel Teorema Mestre.

L'algorisme escrit en C++ és el següent:

```
void pow(int n, vector<vector<int>> &R) {
    if (n == 0) {
        R[0][0] = R[1][1] = 1;
        R[0][1] = R[1][0] = 0;
    }
    else {
        pow(n/2, R);
        vector<vector<int>> S(2, vector<int>(2));
        S[0][0] = R[0][0]*R[0][0] + R[0][1]*R[1][0];
        S[0][1] = R[0][0]*R[0][1] + R[0][1]*R[1][1];
        S[1][0] = R[1][0]*R[0][0] + R[1][1]*R[1][0];
        S[1][1] = R[1][0]*R[0][1] + R[1][1]*R[1][1];
        if (n % 2 == 0) R = S;
        else {
            R[0][0] = S[0][0] + S[0][1];
            R[0][1] = S[0][0];
            R[1][0] = S[1][0] + S[1][1];
            R[1][1] = S[1][0];
        }
    }
}
```

```
int fib (int n) {  
    if (n == 0) return 0;  
    vector<vector<int> > R(2, vector<int>(2));  
    pow(n, R);  
    return R[0][1];  
}
```

## Solució de l'Examen Parcial EDA

19/3/2012

## Proposta de solució al problema 1

•

$f(n)$	$g(n)$	$f = O(g)$	$f = \Omega(g)$	$f = \Theta(g)$
$\sqrt{n}$	$n^{2/3}$	cert	fals	fals
$\log(2n)$	$\log(3n)$	cert	cert	cert
$n^{0.1}$	$(\log n)^{10}$	fals	cert	fals
$2^n$	$2^{n+1}$	cert	cert	cert
$100n + \log n$	$n + (\log n)^2$	cert	cert	cert

- S'ordena el vector amb un algorisme d'ordenació de cost  $O(n \log n)$  en el cas pitjor (com per exemple merge sort). Després de l'ordenació els elements repetits es troben a posicions consecutives del vector, per tant amb un recorregut lineal es poden treure les repeticions.

•

```

string pwr2bin(int n) {
    if (n == 1) return "1010"; // 10 en binari (8 + 2)
    string z = pwr2bin(n/2);
    return Karatsuba(z,z);
}

```

Sigui  $T(n)$  el cost de l'algorisme anterior amb entrada  $n$ . Aleshores  $T(n)$  satisfà la recurrència:

$$T(n) = T(n/2) + O(n^{\log_2 3})$$

perquè es fa una crida recursiva a un problema de la meitat de la mida més el cost  $O(n^{\log_2 3})$  de fer servir l'algorisme de Karatsuba amb enters de mida  $n$  a cada crida recursiva. Com que  $\log_2 3 > 0$ , domina el cost de fer servir l'algorisme de Karatsuba i per tant  $T(n) = O(n^{\log_2 3})$ .

- No. Si tenim un algorisme per multiplicar dos enters de  $n$  bits, aleshores podem elevar al quadrat un enter  $x$  de  $n$  bits amb la mateixa complexitat asimptòtica usant l'algorisme per multiplicar  $x$  per  $x$ .

## Proposta de solució al problema 2

- El nombre màxim  $C(n)$  de crides a  $f$  que pot haver-hi en un moment donat a la pila de recursió segueix la recurrència:

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

Per tant  $C(n) = \Theta(\log n)$ , i l'opció correcta és la c).

- El nombre  $C'(n)$  de crides a  $f$  en el cas pitjor (el qual es dona, per exemple, si  $n = 2^p + 1$  per a un cert  $p \geq 0$ ) segueix la recurrència:

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

Per tant  $C'(n) = \Theta(n)$ , i l'opció correcta és la b).

**Proposta de solució al problema 3**

La funció *misteri* ordena els elements de  $V[e..d]$ . Es pot raonar per inducció. Si  $e \geq d$  la funció no fa res, ja que  $V[e..d]$  ja està ordenat. Si  $e < d$ , llavors la crida *misteri*( $e, d - 1, V$ ) ordena  $V[e..d - 1]$ . Els següents bucle i intercanvi col·loquen el valor  $V[d]$  a la posició  $i$  que li correspon. Ara  $V[e..i]$  està ordenat, i tots els elements de  $V[i + 1..d]$  són més grans o iguals que els de  $V[e..i]$ . La crida *misteri*( $i + 1, d, V$ ) acaba d'ordenar finalment  $V[e..d]$ .

El cas pitjor es dona quan, després de barrejar, a la posició  $d$  hi queda l'element més petit. Si  $n = d - e + 1$ , llavors tant la primera crida com la segona a *misteri* es fan sobre vectors de mida  $n - 1$ . El cost  $T(n)$  ve determinat doncs per:

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

i per tant  $T(n) = \Theta(2^n)$ .

Simètricament, el cas millor es dona quan, després de barrejar, a la posició  $d$  hi queda l'element més gran. Aleshores la primera crida a *misteri* es fa sobre un vector de mida  $n - 1$ , i la segona sobre un vector buit, que té cost constant. Així, el cost  $T(n)$  ve determinat per:

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

i per tant  $T(n) = \Theta(n^2)$ .

**Proposta de solució al problema 4**

- Sí. Com que  $f \in O(g)$ , hi ha  $k > 0$  i  $n_1 \in \mathbb{N}$  tals que  $n \geq n_1$  implica  $f(n) \leq kg(n)$ . Com que  $\lim_{n \rightarrow \infty} g(n) = \infty$ , hi ha  $n_2 \in \mathbb{N}$  tal que si  $n \geq n_2$  llavors  $g(n) \geq k$ , i per tant  $\log g(n) \geq \log k$ . Si  $n \geq \max(n_1, n_2)$ , prenent logaritmes  $\log f(n) \leq \log k + \log g(n) \leq 2 \log g(n)$ . Així doncs,  $\log f \in O(\log g)$ .
- No. Considerem  $f(n) = 2n + 2$  i  $g(n) = n + 1$ . És clar que  $\lim_{n \rightarrow \infty} g(n) = +\infty$ . A més,  $f \in O(g)$ , però  $2^f \notin O(2^g)$  ja que  $2^{f(n)} = 2^{2n+2} = (2^{n+1})^2 = (2^{g(n)})^2$ .

**Proposta de solució al problema 5**

Descripció de l'algorisme:

```

int max_rec(const seq& A, const seq& B, int l, int u) {
    // Pre: l < u and Bl ⊆ A and Bu ⊈ A
    // Post: max{ i | l ≤ i < u and Bi ⊆ A }
    if (l == u-1) return l;
    else {
        int h = (l+u)/2;
        if (is_subseq(Bh, A)) return max_rec(A, B, h, u);
        else return max_rec(A, B, l, h);
    }
}

int max(const seq A&, const seq& B) {
    int n = A.length();
    int m = B.length();
    return max_rec(A, B, 0, n/m + 1);
}

```

Justificació de la correctesa:

Vegem que la funció *max\_rec* és correcta per inducció. Si  $l = u - 1$ , llavors de la precondició es dedueix que el valor buscat és  $l$ . Si  $l < u - 1$  llavors  $l < h < u$ . A més, si  $B^h \subseteq A$  llavors *max\_rec*( $A, B, h, u$ ) satisfà la precondició, i per hipòtesi d'inducció el valor retornat és el buscat. Si en canvi  $B^h \not\subseteq A$  llavors *max\_rec*( $A, B, l, h$ ) satisfà la precondició, i per hipòtesi d'inducció el valor retornat és el buscat.

Finalment, *max\_rec*( $A, B, 0, n/m + 1$ ) satisfà la precondició, ja que  $B^0 = \lambda \subseteq A$ , i raonant sobre les longituds sabem que  $B^{n/m+1} \not\subseteq A$ . Com que *max\_rec* és correcta, el valor retornat és el buscat.

Justificació del cost:

Calculem primer el cost de la part no recursiva d'una crida a *max\_rec*. El cost de construir  $B^h$  és  $O(hm)$ , i com que  $B^h$  té longitud  $hm$ , el cost de *is\_subseq*( $B^h, A$ ) és  $O(hm + n)$ . Com que  $h \leq n/m$ , el cost és  $O(n)$ .

Calculem ara el nombre de crides que es fan a *max\_rec*. Sigui  $p = u - l$ . Llavors el nombre de crides  $C(p)$  ve determinat per la recurrència

$$C(p) = C(p/2) + 1.$$

Per tant, el nombre de crides és  $C(p) = O(\log p)$ .

En conclusió el cost de l'algorisme és  $O(n \log p) = O(n \log(1 + n/m))$ .

**Solució de l'Examen Parcial EDA****22/10/2012****Proposta de solució al problema 1**

- – Cert.
- Cert.
- Fals.
- Fals.
- $T(n) = \Theta(\sqrt{n} \log(n))$ .

**Proposta de solució al problema 2**

En primer lloc observem que

$$\begin{aligned} n! &= 1 \cdot 2 \cdots n \leq n^n \\ n! &= 1 \cdot 2 \cdots n \geq (n/2)^{(n/2)}. \end{aligned}$$

En segon lloc, utilitzant el fet que  $\log_2(x)$  és creixent tenim que

$$\log_2(n!) \leq \log_2(n^n) \leq n \log_2(n) = O(n \log(n))$$

$$\log_2(n!) \geq \log_2((n/2)^{(n/2)}) \geq (n/2) \log_2(n/2) \geq (n/2) \log_2(n) - (n/2) = \Omega(n \log(n)).$$

Per tant  $\log_2(n!)$  és alhora  $O(n \log(n))$  i  $\Omega(n \log(n))$  i per tant és  $\Theta(n \log(n))^1$ .

**Proposta de solució al problema 3**

- La iteració més interior suma  $n - j$  a  $r$ . Per tant, la iteració d'enmig suma

$$\sum_{j=i+1}^n (n-j) = n(n-i) - \left( \sum_{j=1}^n j - \sum_{j=1}^i j \right) = n(n-i) - \frac{n(n+1)}{2} + \frac{i(i+1)}{2}$$

a  $r$ . Aleshores, la iteració més exterior suma

$$\begin{aligned} \sum_{i=1}^n \sum_{j=i+1}^n (n-j) &= n^3 - n \sum_{i=1}^n i - \frac{n^2(n+1)}{2} + \frac{1}{2} \sum_{i=1}^n i^2 + \frac{1}{2} \sum_{i=1}^n i \\ &= n^3 - \frac{n \cdot n(n+1)}{2} - \frac{n^2(n+1)}{2} + \frac{n(n+1)(2n+1)}{6 \cdot 2} + \frac{n(n+1)}{2 \cdot 2} \\ &= \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} \end{aligned}$$

a  $r$ , que inicialment val 0. El grau del polinomi és doncs 3 i els coeficients (de menor a major grau) són  $0, 1/3, -1/2, 1/6$ .

Una solució alternativa consisteix en observar que la funció compta tots els possibles subconjunts de 3 elements que es poden formar en un conjunt de  $n$  elements. Per tant, la funció calcula  $\binom{n}{3} = \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3}$ .

- El cos de la iteració més interior s'executa exactament  $n(n^2 - 3n + 2)/6$  vegades. Per tant, la funció té cost  $\Theta(n^3)$ .

---

<sup>1</sup>La inducció no va bé per demostrar fites en notació asimptòtica. Per exemple, es pot "demostrar per inducció en  $n$ " que  $n = \Theta(1)$  la qual cosa és òbviament absurda. "Demostració: Cas base ( $n = 1$ ):  $1 = \Theta(1)$ . Cas inductiu (de  $n$  a  $n + 1$ ):  $n + 1 = \Theta(1) + 1 = \Theta(1)$ ." Penseu què hi ha d'incorrecte en aquesta demostració.

**Proposta de solució al problema 4**

La funció *misteri\_a* és l'algorisme d'exponenciació ràpida, que calcula  $x^n$  amb cost  $\Theta(\log(n))$  (el mateix en els casos pitjor, mitjà i millor). Per tant, *misteri\_b* eleva a la potència  $n$  cada element del vector  $V$ , és a dir, calcula  $V^n$ , amb cost  $\Theta(m \log(n))$  (el mateix en els casos pitjor, mitjà i millor).

**Proposta de solució al problema 5**

S'aplica un algorisme d'exponenciació ràpida on els productes es fan amb l'algorisme de Karatsuba.

Sigui  $k = \log_2(3)$ , l'exponent de l'algorisme de Karatsuba. Llavors (ignorant els productes dels senars que, com a molt són el doble):

1. Es multiplica  $m \times m$  per obtenir  $m^2$ . És un Karatsuba de  $n$  bits per  $n$  bits, per tant cost  $n^k$ .
2. Es multiplica  $m^2 \times m^2$  per obtenir  $m^4$ . És un Karatsuba de  $2n$  bits per  $2n$  bits, per tant cost  $(2n)^k$ .
3. Es multiplica  $m^4 \times m^4$  per obtenir  $m^8$ . És un Karatsuba de  $4n$  bits per  $4n$  bits, per tant cost  $(4n)^k$ .
- ...
- log\_2(m/2).** Es multiplica  $m^{(m/2)} \times m^{(m/2)}$  per obtenir  $m^m$ . És un Karatsuba de  $(m/2)n$  bits per  $(m/2)n$  bits, per tant cost  $((m/2)n)^k$ .

En total,

$$T(n) = \sum_{i=0}^{n-1} (2^i n)^k = n^k \sum_{i=0}^{n-1} 2^{ik} = \Theta(n^k 2^{kn})$$

on s'utilitza el fet que  $\sum_{i=0}^n 2^{ik} = \Theta(2^{kn})$ .

Alternativament, es podria fer la majoració següent:

$$T(n) \leq n((m/2)n)^k = O(n^{k+1} 2^{kn}).$$

**Proposta de solució al problema 6**

```
double zero(double a, double b, double tol) {
    double c = (a+b)/2;
    if      (b - a ≤ tol or f(c) == 0) return c;
    else if (f(c) * f(b) < 0)         return zero(c, b, tol);
    else                               return zero(a, c, tol);
}
```

En el cas pitjor a cada crida recursiva la longitud de l'interval es redueix en un factor  $\frac{1}{2}$ , i després de  $k$  crides l'interval té doncs longitud  $\frac{b-a}{2^k}$ . El programa s'atura quan aquesta longitud és més petita o igual que  $tol$ . Per tant, el nombre de crides recursives és com a molt  $O(\log(\frac{b-a}{tol}))$ . Com que a cada crida recursiva el treball que es fa és  $O(1)$ , el programa costa en el cas pitjor  $O(\log(\frac{b-a}{tol}))$ .

**Solució de l'Examen Parcial EDA****18/3/2013****Proposta de solució al problema 1**

A:  $T(n) = 5T(n/2) + \Theta(n)$ . És a dir  $T(n) = \Theta(n^{\log_2 5})$ .

B:  $T(n) = 2T(n-1) + \Theta(1)$ . És a dir  $T(n) = \Theta(2^n)$ .

C:  $T(n) = 9T(n/3) + \Theta(n^2)$ . És a dir  $T(n) = \Theta(n^2 \log n)$ .

D: El B és exponencial, i els altres com a molt polinòmics; per tant el descartem. Entre A i C ens quedem amb C perquè  $\log_2 5 > 2.01$  i  $\lim_{n \rightarrow \infty} n^{2.01} / n^2 \log n = \lim_{n \rightarrow \infty} n^{0.01} / \log n = \infty$ .

**Proposta de solució al problema 2**

a) Cert. Justificació: El nombre de compostos a l'interval  $[1, \dots, n]$  és  $n - \pi(n)$ . Per tant, és suficient comprovar que  $n - \pi(n) > \pi(n)$  per  $n$  suficientment gran. Fem-ho. Del fet que  $\pi(n) = \Theta(n / \log n)$  podem concloure que

$$(n - \pi(n)) / \pi(n) = n / \pi(n) - 1 = \Theta(\log n).$$

Donat que  $\lim_{n \rightarrow \infty} \log n = \infty$  en deduïm que  $n - \pi(n) > \pi(n)$  per  $n$  suficientment gran.

b) Fals. Justificació: El nombre de quadrats a l'interval  $[1, \dots, n]$  és com a molt  $\sqrt{n}$ . Per tant, és suficient comprovar que  $\sqrt{n} < \pi(n)$  per  $n$  suficientment gran. Fem-ho. Del fet que  $\pi(n) = \Theta(n / \log n)$  podem concloure que

$$\pi(n) / \sqrt{n} = \Theta(\sqrt{n} / \log n).$$

Donat que  $\lim_{n \rightarrow \infty} \sqrt{n} / \log n = \infty$  en deduïm que  $\pi(n) > \sqrt{n}$  per  $n$  suficientment gran.

**Proposta de solució al problema 3**

a) Això són tres bucles imbricats, cadascun d' $n$  iteracions, i amb cost  $\Theta(1)$  per cada iteració. Total:  $\Theta(n^3)$ .

b) Això tornen a ser tres bucles imbricats, però ara el cost és

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=j}^{n-1} \Theta(1) \leq \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \Theta(1) = \Theta(n^3).$$

Per altra banda tenim que

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=j}^{n-1} \Theta(1) \geq \sum_{i=0}^{n/3} \sum_{j=n/3}^{2n/3} \sum_{k=2n/3}^{n-1} \Theta(1) = \Theta((n/3)^3) = \Theta(n^3).$$

Total: el cost és alhora  $O(n^3)$  i  $\Omega(n^3)$  i per tant  $\Theta(n^3)$ .

c) Construir  $Q$  té cost  $\Theta(n^2)$ . Ordenar  $Q$  té cost  $\Theta(n^2 \log n^2) = \Theta(n^2 \log n)$ . L'última fase té cost  $\Theta(n \log n^2) = \Theta(n \log n)$  perquè per a cada  $k$  busquem  $s - V[k]$  a  $Q$  fent una cerca dicotòmica. Total:  $\Theta(n^2 \log n)$ .

d) Construir  $Q$  té cost  $\Theta(n^2)$ . Ordenar  $V$  té cost  $\Theta(n \log n)$ . L'última fase té cost  $\Theta(n^2 \log n)$  perquè per a cada  $\ell$  busquem  $s - Q[\ell]$  a  $V$  fent una cerca dicotòmica. Total:  $\Theta(n^2 \log n)$ .



**Proposta de solució al problema 4**

1. El següent codi satisfà els requeriments:

```

int g1(int n) {
    if (n ≤ 2) return 1;
    int v1, v2, v3;
    v1 = v2 = v3 = 1;
    for (int i = 2; i < n; ++i) {
        int aux = v2 + v3;
        v3 = v2;
        v2 = v1;
        v1 = aux;
    }
    return v1;
}

```

Per a  $n \geq 3$  es fan  $n - 2$  iteracions, cadascuna de les quals amb cost temporal  $\Theta(1)$ . D'aquí el cost en temps  $\Theta(n)$ .

2. Aquí és el codi amb les caixetes plenes:

```

void misteri(const matrix<int>& m, int k, matrix<int>& p) {
    if (k == 0)
        for (int i = 0; i < 3; ++i) p[i][i] = 1;
    else {
        misteri(m, k/2, p);
        p = p * p;
        if (k % 2 == 1) p * m;
    }
}

```

**Proposta de solució al problema 5**

Primer busquem la posició  $p \geq 1$  d'algun  $\infty$  així:

```

int p = 1;
while (V[p] ≠ ∞) p *= 2;

```

Comencem en 1 perquè ens diuen que  $n \geq 1$ . El nombre d'iteracions serà el mínim  $i \geq 0$  tal que  $2^i \geq n$ . És a dir  $i = \Theta(\log n)$ . Un cop tenim aquest  $p$  fem una cerca dicotòmica entre  $v[p/2]$  i  $v[p]$  buscant la posició del primer  $\infty$  així:

```

int l = p/2;
int r = p;
while (l < r - 1) {
    int q = (l + r)/2;
    if (V[q] == ∞) r = q;
    else l = q;
}
return r;

```

El nombre d'iteracions d'aquesta segona fase és  $\Theta(\log(p/2))$ . Donat que  $p/2 < n \leq p$ , tenim  $p/2 = \Theta(n)$ . Per tant això són  $\Theta(\log n)$  iteracions. Cost total:  $\Theta(\log n) + \Theta(\log n) = \Theta(\log n)$ .

**Solució de l'Examen Parcial EDA****21/10/2013****Proposta de solució al problema 1**

- $T(n) = \Theta(n)$ .
- Les tres:  $O(n^2)$ ,  $\Theta(n \log n)$  i  $\Omega(n \log n)$ .
- $f(n) = n \log \log n$ .
- $T(n) = \Theta(n^2)$ .
- $T(n) = 3T(n/2) + \Theta(n)$ .
- $T(n) = \Theta(n)$ .
- $T(n) = \Theta(1)$ .
- $T(n) = \Theta(n \log n)$ .

**Proposta de solució al problema 2**

a) El primer fa  $n - 1$  iteracions de cost constant i, per tant, té un cost  $\Theta(n)$ . El cost del segon es pot descriure amb la recurrència  $T(n) = 2T(n/2) + \Theta(1)$ , que pel teorema mestre de les recurrències divisores, és  $\Theta(n)$ .

b) El primer compara les variables *min* i *max* amb cada posició del vector  $A[i]$ , per a  $1 \leq i \leq n - 1$ , de manera que el nombre total de comparacions és  $2n - 2$ . En el cas que  $n$  és potència de dos, suposem  $n = 2^k$ , l'arbre de recursió generat pel segon algorisme serà un arbre binari complet on cada fulla correspondrà a un subvector de dos elements. Per tant, l'alçada de l'arbre serà  $k - 1$ , el nombre de fulles  $2^{k-1} = n/2$  i el nombre de nodes interns (corresponents a crides amb subvectors de mida més gran que 2)

$$\sum_{i=0}^{k-2} 2^i = 2^{k-1} - 1.$$

Tenint en compte que en el cas base (subvectors de mida 2) es fa només una comparació i que en cada crida del cas general (subvectors de mida  $> 2$ ) se'n fan 2, el total de comparacions serà

$$2^{k-1} + 2(2^{k-1} - 1) = 2^{k-1} + 2^k - 2 = 3 \cdot 2^{k-1} - 2 = 3n/2 - 2,$$

que millora les  $2n - 2$  comparacions de l'algorisme iteratiu.

**Proposta de solució al problema 3**

a) Creem un vector addicional  $P[1, \dots, n]$  que dirà, per cada  $i = 1, \dots, n$ , la posició de  $i$  a  $V$ . Per omplir  $P$  simplement recorrem l'entrada  $i$ , per cada  $i = 1, \dots, n$ , fem  $P[V[i]] = i$ . Finalment fem  $D[i] = |P[i + 1] - P[i]|$  per cada  $i = 1, \dots, n - 1$ . El cost és  $\Theta(n)$  perquè són dos recorreguts de dos vectors de longitud  $n$ .

b) Creem un vector addicional  $P[1, \dots, n]$  que dirà, per cada  $i = 1, \dots, n$ , la posició de l' $i$ -èssim element més gran de  $V$ . Per fer-ho, ordenarem el vector  $V$  fent un mergesort modificat de manera que, quan col·loqui l'element  $V[k]$  a la seva posició definitiva  $i$ , posi  $P[i] = k$ . Finalment fem  $D[i] = |P[i + 1] - P[i]|$  per cada  $i = 1, \dots, n - 1$ . El cost és  $\Theta(n \log n)$  per la fase on s'executa el mergesort modificat, i  $\Theta(n)$  per la fase on es genera  $D$ . El cost total és doncs  $\Theta(n \log n)$ .

**Proposta de solució al problema 4**

La funció *misteri\_a* és l'algorisme d'exponenciació ràpida, que calcula  $x^n$  amb cost  $\Theta(\log(n))$  (el mateix en els casos pitjor, mitjà i millor). Per tant, *misteri\_b* eleva a la potència  $n$  cada element del vector  $V$ , és a dir, calcula  $V^n$ , amb cost  $\Theta(m \log(n))$  (el mateix en els casos pitjor, mitjà i millor).

**Proposta de solució al problema 5**

Per una banda tenim que  $S(n) \geq 1$  i per l'altra que

$$S(n) = \sum_{i=1}^n \frac{1}{i^2} = 1 + \sum_{i=2}^n \frac{1}{i^2} \leq 1 + \int_1^n \frac{1}{x^2} dx = 1 + \left[ -\frac{1}{x} \right]_1^n = 2 - \frac{1}{n}.$$

Per tant  $S(n)$  és alhora  $\Omega(1)$  i  $O(1)$  i per tant  $\Theta(1)$ .

**Proposta de solució al problema 6**

L'enunciat és fals. Un contraexemple seria  $f(n) = 2^n$  i  $c = 2$ . En efecte

$$\lim_{n \rightarrow \infty} \frac{f(cn)}{f(n)} = \lim_{n \rightarrow \infty} \frac{2^{2n}}{2^n} = \lim_{n \rightarrow \infty} 2^n = \infty$$

i per tant  $f(cn)$  no és  $\Theta(f(n))$ .

**Solució de l'Examen Parcial EDA****24/3/2014****Proposta de solució al problema 1**a)  $H(n) = H(n-1) + 1$  i  $W(n) = 2W(n-1) + 2$ .b)  $L(n) = 2L(n-1) + 2$ .

c) El teorema mestre dona  $H(n) = \Theta(n)$ ,  $W(n) = \Theta(2^n)$  i  $L(n) = \Theta(2^n)$ . Siguin  $A_1(n)$  i  $A_2(n)$  les àrees per un arbre complet de  $4^n$  fulles en la primera i la segona construcció, respectivament. Tenint en compte que  $4^n = 2^{2n}$  tenim que  $A_1(n) = H(2n)W(2n) = \Theta(2^{2n})\Theta(2n) = \Theta(2^{2n}n)$ . Per altra banda  $A_2(n) = L(n)^2 = \Theta(2^{2n})$ . Donat que  $\lim_{n \rightarrow \infty} 2^{2n}n/2^{2n} = +\infty$ , en deduïm que  $A_2(n)$  és  $O(A_1(n))$  però no  $\Theta(A_1(n))$ . Per tant, la segona construcció és asimptòticament més eficient que la primera.

**Proposta de solució al problema 2**

El cos de cada iteració de cada bucle és  $\Theta(1)$  i per tant només cal comptar quantes iteracions fa cada bucle. Si  $y_t$  denota el valor de  $y$  al final de la  $t$ -èssima iteració, el que busquem és el mínim  $t \geq 0$  tal que  $y_t > n$ . Determinem  $y_t$  per cada bucle:

1.  $y_t = 1 + 2 + 3 + 4 + \dots + (t+1) = \sum_{j=1}^{t+1} j = \Theta(t^2)$ .

2.  $y_t = 2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^t = \Theta(2^t)$ .

3.  $y_t = 2^0 \cdot 2^1 \cdot 2^2 \cdot 2^3 \dots 2^t = 2^{\sum_{j=1}^t j} = 2^{\Theta(t^2)}$ .

4.  $y_t = 1 \cdot 2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdot 2^{2^3} \dots 2^{2^{t-1}} = 2^{\sum_{j=0}^{t-1} 2^j} = 2^{\Theta(2^t)}$ .

Ara busquem el mínim  $t$  que satisfà  $y_t > n$ :

1. Volem  $\Theta(t^2) > n$  i el mínim tal  $t$  és  $t = \Theta(\sqrt{n})$ .

2. Volem  $\Theta(2^t) > n$  i el mínim tal  $t$  és  $t = \Theta(\log n)$ .

3. Volem  $2^{\Theta(t^2)} > n$ , o  $\Theta(t^2) > \log n$ , i el mínim tal  $t$  és  $t = \Theta(\sqrt{\log n})$ .

4. Volem  $2^{\Theta(2^t)} > n$ , o  $\Theta(2^t) > \log n$ , i el mínim tal  $t$  és  $t = \Theta(\log \log n)$ .

**Proposta de solució al problema 3**

a) Considerem una  $k$  arbitrària tal que  $i < k < j$ . Hi ha dos casos:

- Si  $T[i] > T[k]$ , llavors el parell  $(i, k)$  és una inversió.
- Altrament, com que  $T[k] \geq T[i] > T[j]$ , el parell  $(k, j)$  és una inversió.

En qualsevol cas, per cada  $k$  tal que  $i < k < j$  podem comptar almenys una inversió. Comptant també la inversió  $(i, j)$ , tenim almenys  $j - i$  inversions.

b) De l'apartat anterior deduïm que, si  $T$  és un vector amb com a molt  $N$  inversions, aleshores tota inversió  $(i, j)$  ha de satisfer  $j - i \leq N$ . A continuació es mostra una possible implementació que utilitza aquesta observació:

```
bool cerca(int x, const vector<int>& T, int l, int r) {
    if (r-l+1 < 2*N+1) {
        for (int i = l; i <= r; ++i) if (T[i] == x) return true;
        return false;
    }
```

```

    }
    int m = (l+r)/2;
    if (T[m] > x) {
        for (int k = 1; k ≤ N; ++k) if (T[m+k] == x) return true;
        return cerca(x, T, l, m-1);
    }
    if (T[m] < x) {
        for (int k = 1; k ≤ N; ++k) if (T[m-k] == x) return true;
        return cerca(x, T, m+1, r);
    }
    return true;
} }

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

```

Com que  $N$  és una constant, el cost dels bucles dels casos recursius és  $\Theta(1)$ . La recurrència que descriu el cost  $C(n)$  de *cerca* per a vectors de mida  $n$  en el cas pitjor (per exemple, quan l'enter  $x$  no pertany al vector  $T$ ) és doncs

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

la solució de la qual és  $C(n) = \Theta(\log n)$  d'acord amb el Teorema Mestre de Recurrències Divisores.

#### Proposta de solució al problema 4

1) Resposta:  $\Theta(n)$ . Justificació: Donat que  $-1 \leq \cos(x) \leq 1$  per a qualsevol  $x \in \mathbb{R}$ , tenim que  $\log(3) \leq \log(\cos(n\pi) + 4) \leq \log(5)$  per a qualsevol  $n \geq 0$ . Per tant  $\log(\cos(n\pi) + 4) = \Theta(1)$  i  $f(n) = n\Theta(1) = \Theta(n)$ .

2) Resposta: 1. Justificació: Si ja hem calculat  $7^{2^i}$  podem obtenir  $7^{2^{i+1}}$  simplement elevant-lo al quadrat. Fem-ho, tot mòdul 10:  $7^2 = 9$ ,  $7^4 = 9^2 = 1$ ,  $7^8 = 1^2 = 1$ . I d'aquí no ens movem perquè  $1^2 = 1$ . Per tant  $7^{1024} = 7^{2^{10}} = 1$ .

3)  $T(n) = 7T(n/2) + \Theta(n^2)$ .

4) Resposta:  $\Theta(n^2)$ . Justificació: El cost en el cas mitjà és

$$\frac{1}{2^n} \cdot \Theta(n^4) + \left(1 - \frac{1}{2^n}\right) \cdot \Theta(n^2)$$

que és  $\Theta(n^4/2^n) + \Theta(n^2) - \Theta(n^2/2^n)$  i per tant  $\Theta(n^2)$  perquè les altres dues funcions tendeixen a 0 quan  $n$  tendeix a infinit.

**Solució de l'Examen Parcial EDA****13/10/2014****Proposta de solució al problema 1**

a) Veurem que la suma és  $O(n^4)$  i  $\Omega(n^4)$ . Primer,  $\sum_{i=0}^n i^3 \leq n \cdot n^3 = n^4 = O(n^4)$ . Segon, si  $n$  és parell, llavors  $\sum_{i=0}^n i^3 \geq \frac{n}{2} \cdot \left(\frac{n}{2}\right)^3 \geq \frac{1}{16}n^4 = \Omega(n^4)$ . I si  $n$  és senar, llavors  $\sum_{i=0}^n i^3 = \sum_{i=0}^{n-1} i^3 + n^3 \geq \frac{1}{16}(n-1)^4 + n^3 = \Omega(n^4)$ . En qualsevol cas obtenim que  $\sum_{i=0}^n i^3 = \Omega(n^4)$ .

b)  $g$  creix més ràpid que  $f$ . Per justificar-ho veurem que  $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ . Sigui  $h(n) = \log_2(f(n)/g(n))$ . Simple manipulació dóna  $h(n) = \sqrt{\log n} - \log_2 n$  i per tant  $\lim_{n \rightarrow +\infty} h(n) = -\infty$ . Això implica que  $\lim_{n \rightarrow +\infty} 2^{h(n)} = 0$  i per tant  $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ . Pel criteri del límit en concloem que  $f(n) = O(g(n))$  i  $f(n) \neq \Omega(g(n))$ . Per tant  $g$  creix més ràpid que  $f$ .

**Proposta de solució al problema 2**

a)  $B$  conté els elements de  $A$  ordenats de més petit a més gran. Per justificar-ho, fixeuvos que, abans d'executar l'últim bucle,  $C[i]$  conté el nombre d'elements de  $A$  que estan entre 0 i  $i$ , ambdós inclosos, per  $0 \leq i \leq k$ . Per tant, l'últim bucle posa a  $B$  tantes còpies de cada element de  $A$  com hi ha al vector  $A$ , de dreta a esquerra i de més gran a més petit.

b) Quan  $k = O(n)$ , el cost és  $\Theta(n)$ . Per justificar-ho, analitzem línia a línia. La primera línia té cost  $\Theta(1)$ . La segona línia té cost  $\Theta(k)$ . La tercera línia té cost  $\Theta(n)$ . La quarta línia té cost  $\Theta(k)$ . La cinquena línia té cost  $\Theta(n)$ . I l'últim bucle té cost  $\Theta(n)$ . El cost global és doncs  $\Theta(n+k)$  i, quan  $k = O(n)$ , el terme  $n$  guanya i obtenim cost  $\Theta(n)$ .

c) Primer computem  $C[0, \dots, k]$  com a *misteri* de manera que  $C[i]$  és el nombre d'elements de  $A$  que estan entre 0 i  $i$ , ambdós inclosos. I després, per cada  $i = 0, \dots, m-1$ , si  $a_i > 0$  escrivim  $C[b_i] - C[a_i - 1]$ , i si  $a_i = 0$  escrivim  $C[b_i]$ . El cost per calcular  $C$  és  $\Theta(n+k)$  i el cost per tractar els  $(a_i, b_i)$  és  $\Theta(m)$ . El cost global és doncs  $\Theta(n+k+m)$ .

**Proposta de solució al problema 3**

a)  $R_{i,j}$  és la probabilitat d'anar de  $i$  a  $j$  en exactament dues tirades. Per justificar-ho, només cal interpretar  $k$  com el pas intermig i adonar-se que  $P_{i,k}P_{k,j}$  és la probabilitat que en la primera tirada anem de  $i$  a  $k$  i en la segona anem de  $k$  a  $j$ . Sumant sobre  $k = 1, \dots, 64$  obtenim el que volem.

b)

```
void probabilities (const Matrix& P, int t, Matrix& Q) {
    if (t == 0) identity (Q, 64);
    else {
        Matrix R, S;
        probabilities (P, t/2, R);
        multiply(R, R, S);
        if (t % 2 == 1) multiply(S, P, Q);
        else Q = S;
    }
}
```

```
void identity (Matrix& Q, int n) {
    Q = Matrix(n, Row(n, 0.0));
    for (int i = 0; i < n; ++i) Q[i][i] = 1.0;
```

```

}

void multiply(const Matrix& A, const Matrix& B, Matrix& C) {
    C = Matrix(A.size (), Row(B[0].size (), 0.0));
    for (int i = 0; i < A.size (); ++i) {
        for (int j = 0; j < B[0].size (); ++j) {
            for (int k = 0; k < B.size (); ++k) {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}

```

Cost:  $T(t) = T(t/2) + \Theta(1)$  i per tant  $T(t) = \Theta(\log t)$  pel cas  $\alpha = k = 0$  del teorema mestre de recurrències divisores.

#### Proposta de solució al problema 4

a) Resposta:  $\Omega(2^n)$ .

Justificació: Sigui  $B$  el número d'entrades que causen cost asimptòtic  $\Theta(n^2)$ , i sigui  $G = 2^n - B$  el número d'entrades amb cost asimptòticament inferior a  $n^2$ . Llavors el cost en el cas mitjà és  $(B/2^n)\Theta(n^2) + (1 - B/2^n)f(n)$  on  $f(n)$  és una funció de creixement asimptòticament inferior a  $n^2$ . Per a què aquest cost sigui  $\Theta(n^2)$  és necessari que, per  $n$  suficientment gran,  $B$  sigui al menys una fracció de  $2^n$ , és a dir,  $B \geq c2^n$  per alguna  $c > 0$ , o  $B = \Omega(2^n)$ . La mateixa fórmula permet veure que la condició  $B = \Omega(2^n)$  també és suficient per garantir cost  $\Theta(n^2)$ .

b) Resposta:  $\frac{9}{10} \left(\frac{1}{10}\right)^k$

Justificació: Perquè l'algorisme faci **exactament**  $k$  iteracions cal que  $A$  acabi amb  $k$  9's i el dígit just anterior sigui diferent de 9. La probabilitat d'aquest esdeveniment és el què hem escrit.

c) Resposta:  $T(n) = \Theta(1)$ .

Justificació: Amb la notació de classe per les recurrències substractores tenim que  $a = 1/10$ ,  $c = 1$  i  $k = 0$ . Per tant, donat que  $a < 1$ , el resultat és  $T(n) = \Theta(n^k) = \Theta(1)$ .

d)  $T(n)$  correspon al cost asimptòtic en el cas mitjà amb la distribució de probabilitat especificada a l'apartat b).

**Solució de l'Examen Parcial EDA**  
**Proposta de solució al problema 1**

23/03/2015

a)  $\lfloor n/2 \rfloor - 1$

b) 0

c)  $\lfloor n/17 \rfloor - 1$

d) El cost de l'algorisme es pot expressar com:

$$\sum_{\substack{x=2 \\ x \text{ primer}}}^n \Theta(\lfloor n/x \rfloor - 1) + \sum_{\substack{x=2 \\ x \text{ no primer}}}^n \Theta(1).$$

El segon sumatori és  $O(n)$ . Per altra banda, com que  $\lfloor n/x \rfloor - 1$  és  $\Theta(n/x)$ , el primer sumatori és equivalent a

$$\sum_{\substack{x=2 \\ x \text{ primer}}}^n \Theta(n/x) = n \sum_{\substack{x=2 \\ x \text{ primer}}}^n \Theta(1/x) = \Theta(n \log \log n).$$

El resultat és doncs

$$\Theta(n \log \log n) + O(n) = \Theta(n \log \log n).$$

e) El cost no millora, continua essent  $\Theta(n \log \log n)$ , perquè el cost en aquest cas té una expressió similar a l'anterior amb l'única diferència que ara els sumatoris arriben només fins a  $\sqrt{n}$ . Per tant, l'expressió final que un obté és  $\Theta(n \log \log \sqrt{n})$ , que és el mateix que  $\Theta(n \log \log n)$ , ja que  $\log \log \sqrt{n} = \log(\frac{1}{2} \log n) = \log \frac{1}{2} + \log \log n = \Theta(\log \log n)$ .

**Proposta de solució al problema 2**

a) El mínim  $n$  tal que  $n^3 \geq 10n^{2.81}$ , és a dir,  $n^{0.19} \geq 10$ , és  $n = \lceil 10^{\frac{1}{0.19}} \rceil = 183299$ .

b) El mínim  $n$  tal que  $10n^{2.81} \geq 100n^{2.38}$ , és a dir,  $n^{0.43} \geq 10$ , és  $n = \lceil 10^{\frac{1}{0.43}} \rceil = 212$ .

**Proposta de solució al problema 3**

a) Una solució consisteix en ordenar els intervals en ordre creixent per l'extrem esquerre en temps  $\Theta(n \log n)$ , i després processar-los de la manera descrita a continuació. Recorrem la seqüència d'esquerra a dreta mantenint l'extrem esquerre mínim (*eem*) i l'extrem dret màxim (*edM*) vistos des de l'últim interval que hem escrit a la sortida. Si el següent interval de la seqüència té un extrem esquerre més gran que l'*edM* llavors podem *tancar* l'interval [*eem*, *edM*], afegir-lo a la sortida, i actualitzar *eem* i *edM* als extrems esquerre i dret de l'interval que acabem de processar. En cas contrari actualitzem l'*edM* si és necessari, és a dir, si l'extrem dret de l'interval processat és més gran que l'*edM*. El cost d'aquesta fase és  $\Theta(n)$  i per tant el cost total és  $\Theta(n \log n)$ .<sup>2</sup>

b) En primer lloc calculem la unió dels intervals igual que en el primer apartat, amb cost  $\Theta(n \log n)$ . Després determinem, per cada punt  $p_i$ , si està dins d'algun interval de la unió

<sup>2</sup>Una segona solució seria un algorisme de dividir-i-vèncer semblant a l'ordenació per fusió.



o no. Quan  $m$  és gran, com és el cas si  $m = n$ , una bona solució consisteix a ordenar  $p$  en ordre creixent, i després “fusionar” intervals i punts en temps lineal. A cada pas de la fusió considerem un interval  $[a_i, b_i]$  i un punt  $p_j$ . Si  $b_i < p_j$ , l'interval es descarta i avancem a la seqüència d'intervals. Si  $a_i \leq p_j \leq b_j$ , incrementem el comptador i avancem a les dues seqüències. Si  $p_j < a_i$ , el punt es descarta i avancem a la seqüència de punts. Quan no quedin punts, el procés acaba. El cost de la segona fase és  $\Theta(m \log m) + \Theta(n + m)$ . Per  $m = n$ , això és  $\Theta(n \log n)$ .

c) Farem servir un altre algorisme. Un punt pertany a la unió si i només si pertany a algun dels intervals de la seqüència d'entrada, i per tant podem determinar si pertany a la unió en temps  $\Theta(n)$  simplement recorrent la seqüència d'intervals tal com ens ve donada (sense processar-la prèviament). Donat que  $m \leq 5$ , això són no més de 5 recorreguts de cost  $\Theta(n)$  cadascun i per tant el cost total és  $\Theta(n)$ .<sup>3</sup>

#### Proposta de solució al problema 4

a) Resposta:  $\Theta(3^{\log_2(n)}) \neq \Theta(3^{\log_4(n)})$ . Justificació: Siguin  $f(n) = 3^{\log_2(n)}$  i  $g(n) = 3^{\log_4(n)} = 3^{\log_2(n)/2}$  de manera que  $f(n)/g(n) = 3^{\log_2(n) - \log_2(n)/2} = 3^{\log_2(n)/2}$ . Com que  $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$ ,  $f(n)$  creix estrictament més ràpid que  $g(n)$ .

b) Cal calcular  $2^1 \cdot \dots \cdot 2^{100} = 2^{5050} \pmod{9}$ . Com que  $2^6 = 1 \pmod{9}$  i  $5050 = 4 \pmod{6}$ , tenim  $2^{5050} = 2^4 = 7 \pmod{9}$ .

c) Ordenades d'ordre de creixement més petit a més gran, les funcions són

$$(\ln(n))^2, n^{1/3}, \sqrt{n}, n^4 - 3n^3 + 1.$$

d) Les tres recurrències són:

$$A(n) = \Theta(n) + 5A(n/2) = \Theta(n^{\log_2 5}).$$

$$B(n) = \Theta(1) + 2B(n-1) = \Theta(2^n).$$

$$C(n) = \Theta(n^2) + 9C(n/3) = \Theta(n^2 \log n).$$

C és la més eficient perquè  $\log n$  creix més lentament que  $n^{\log_2 5 - 2} = n^{0.3219\dots}$ .

---

<sup>3</sup>Una segona solució, menys eficient, seria primer calcular la unió en forma d'intervals disjunts ordenats com en el primer apartat en temps  $\Theta(n \log n)$ , i després fer  $m$  cerques dicotòmiques en temps  $\Theta(m \log n)$ . Quan  $m$  és una constant, el cost total és  $\Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

### Proposta de solució al problema 1

19/10/2015

(a) Anàlisi de cost:

- *En temps*: el cost de les inicialitzacions és  $\Theta(k)$ , per la creació del vector  $f$ . Com que el bucle s'executa  $\Theta(k)$  vegades i cada iteració costa temps  $\Theta(1)$ , la contribució al cost del bucle és  $\Theta(k)$ . En total el cost és doncs  $\Theta(k)$ .
- *En espai*:  $\Theta(k)$ , ja que el consum de memòria està dominat per la creació del vector  $f$ , de mida  $k + 1$ .

(b) Per inducció.

- *Cas base*:  $k = 2$ . Aleshores efectivament

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k-1} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_2 & f_1 \\ f_1 & f_0 \end{pmatrix} = \begin{pmatrix} f_k & f_{k-1} \\ f_{k-1} & f_{k-2} \end{pmatrix}.$$

- *Cas inductiu*:  $k > 2$ . Per hipòtesi d'inducció:

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

(c) Una possible solució:

```
typedef vector<vector<int>> matrix;
```

```
matrix mult(const matrix& A, const matrix& B) {
    assert (A.size () == A[0].size ());
    assert (B.size () == B[0].size ());
    assert (A.size () == B.size ());
    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)
                C[i][j] += A[i][k] * B[k][j];
    return C;
}
```

```
matrix misteri (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) R[i][i] = 1;
        return R;
    }
    else {
```

```

matrix P = misteri (M, q/2);
if (q % 2 == 0) return mult(P, P);
else return mult(mult(P, P), M);
} }

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

```

- (d) Primer analitzem el cost de  $misteri(M, k)$  en funció de  $k$ , que anomenarem  $C(k)$ . Observem que les crides a  $mult$  sempre es fan amb matrius  $2 \times 2$ , i per tant triguen temps constant. Per tant el cost no recursiu és constant i tenim que  $C(k) = C(k/2) + \Theta(1)$ , d'on aplicant el Teorema Mestre de Recurrències Divisores s'obté que  $C(k) = \Theta(\log k)$ .

Així doncs, el cost de  $fib2(k)$  és  $C(k-1) + \Theta(1) = \Theta(\log k)$ .

### Proposta de solució al problema 2

- (a) Un cas millor es dona quan  $x$  és a la primera posició, és a dir,  $x$  és  $v[0]$ . Aleshores només s'entra un cop al bucle, i el cost total és  $\Theta(1)$ .
- (b) Un cas pitjor es dona quan  $x$  no apareix al vector  $v$ . Aleshores es fan  $\Theta(n)$  iteracions del bucle, cadascuna de les quals triga temps  $\Theta(1)$ . El cost total és  $\Theta(n)$ .
- (c) Per inducció.

- *Case base:*  $n = 1$ . Tenim  $\sum_{i=1}^n \frac{i}{2^i} = \frac{1}{2}$ , i  $2 - \frac{n}{2^n} - \frac{1}{2^{n-1}}|_{n=1} = 2 - \frac{1}{2} - 1 = \frac{1}{2}$ .
- *Cas inductiu:*  $n > 1$ . Tenim que  $\sum_{i=1}^{n-1} \frac{i}{2^i} = 2 - \frac{n-1}{2^{n-1}} - \frac{1}{2^{n-2}}$ . Per tant  $\sum_{i=1}^n \frac{i}{2^i} = \frac{n}{2^n} + \sum_{i=1}^{n-1} \frac{i}{2^i} = \frac{n}{2^n} + 2 - \frac{n-1}{2^{n-1}} - \frac{1}{2^{n-2}} = \frac{n}{2^n} + 2 - \frac{2n-2}{2^{n-2}} - \frac{4}{2^n} = 2 + \frac{n-2n+2-4}{2^n} = 2 - \frac{n}{2^n} - \frac{1}{2^{n-1}}$ .

- (d) Si  $x$  és l'element  $v[i]$ , aleshores l'algorisme té cost  $\Theta(i)$ . Per tant el cost mig és

$$\begin{aligned}
 & \sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot \text{Cost}(x = v[i]) = \\
 & \sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot \Theta(i) = \\
 & \Theta\left(\sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot i\right) = \\
 & \Theta\left(\sum_{1 \leq i < n} \text{Prob}(x = v[i]) \cdot i\right) = \\
 & \Theta(\text{Prob}(x = v[n-1]) \cdot (n-1) + \sum_{1 \leq i < n-1} \text{Prob}(x = v[i]) \cdot i) = \\
 & \Theta\left(\frac{n-1}{2^{n-1}} + \sum_{1 \leq i < n-1} \frac{i}{2^{i+1}}\right) = \\
 & \Theta\left(\frac{n-1}{2^{n-1}} + \frac{1}{2} \cdot \sum_{i=1}^{n-2} \frac{i}{2^i}\right) = \\
 & \Theta\left(\frac{n-1}{2^{n-1}} + \frac{1}{2} \cdot \left(2 - \frac{n-2}{2^{n-2}} - \frac{1}{2^{n-3}}\right)\right) = \\
 & \Theta(1)
 \end{aligned}$$

donat que  $\lim_{n \rightarrow \infty} \frac{1}{2^n} = \lim_{n \rightarrow \infty} \frac{n}{2^n} = 0$ .

### Proposta de solució al problema 3

- (a) Considerem cadascun dels casos:

- Assumim  $a, b$  parells. Tenim  $\gcd(a/2, b/2) \mid a/2$ , i així  $2\gcd(a/2, b/2) \mid a$ . Similment,  $2\gcd(a/2, b/2) \mid b$ . Per tant  $2\gcd(a/2, b/2) \mid \gcd(a, b)$ . Per altra ban-

da, com que  $a$  i  $b$  són parells,  $\gcd(a, b)$  és parell. Però  $\gcd(a, b) \mid a$  implica que  $\gcd(a, b)/2 \mid a/2$ , i similarment  $\gcd(a, b)/2 \mid b/2$ . Per tant  $\gcd(a, b)/2 \mid \gcd(a/2, b/2)$ , i  $\gcd(a, b) \mid 2\gcd(a/2, b/2)$ , d'on finalment  $\gcd(a, b) = 2\gcd(a/2, b/2)$ .

- Assumim  $a$  senar i  $b$  parell. Per una banda  $\gcd(a, b) \mid a$ . Per una altra  $\gcd(a, b) \mid b$ , i com que  $a$  és senar,  $\gcd(a, b) \mid b/2$ . Així doncs tenim que  $\gcd(a, b) \mid \gcd(a, b/2)$ . I com que  $\gcd(a, b/2) \mid a$  i  $\gcd(a, b/2) \mid b$ , tenim  $\gcd(a, b/2) \mid \gcd(a, b)$ . Per tant  $\gcd(a, b) = \gcd(a, b/2)$ .
- Assumim  $a, b$  senars i  $a > b$ . Si  $a$  i  $b$  són senars,  $a - b$  és parell. Per tant  $\gcd(a, b) = \gcd(a - b, b) = \gcd((a - b)/2, b)$  per la pista i l'apartat anterior.

(b) Una possible solució:

```
int gcd(int a, int b) {
    if (a == b) return a;
    if (a == 1 or b == 1) return 1;
    if (a % 2 == 0 and b % 2 == 0) return 2*gcd(a/2, b/2);
    if (a % 2 == 1 and b % 2 == 0) return gcd(a, b/2);
    if (a % 2 == 0 and b % 2 == 1) return gcd(a/2, b);
    else
        if (a > b) return gcd((a-b)/2, b);
        else return gcd(a, (b-a)/2);
}
```

- (c) Un cas pitjor es dona, per exemple, quan  $a$  és una potència de 2 i  $b$  és senar. Aleshores a cada crida recursiva només decreix el primer argument, i  $b$  sempre és el segon argument. Quan el primer argument té  $i$  bits, el cost és  $\Theta(i)$ , per la divisió entre 2. Per tant el cost és  $\sum_{i=1}^n \Theta(i) = \Theta(n^2)$ .

## Solució de l'Examen Parcial EDA

31/03/2016

## Proposta de solució al problema 1

(a)  $\Theta(n^{\log 7})$ 

(b) El teorema mestre de recurrències substractores afirma que si tenim una recurrència de la forma  $T(n) = aT(n - c) + \Theta(n^k)$  amb  $a, c > 0$  i  $k \geq 0$ , llavors

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

(c)  $\Theta(n)$ 

## Proposta de solució al problema 2

(a) El cost del programa queda determinat per la suma de costos de dos grups d'instruccions:

- A.** Les instruccions que s'executen a cada iteració del bucle: l'avaluació de la condició  $i \leq n$ , la crida  $f(i)$ , l'avaluació de la condició  $i == p$  i l'increment  $i++$ .
- B.** Les instruccions que s'executen quan la condició de l'if és certa: la crida  $g(n)$  i l'increment  $p *= 2$ .

Comptem per separat la contribució al cost total dels dos grups d'instruccions:

- A.** Si el cost de  $f(m)$  és  $\Theta(m)$ , llavors a la iteració  $i$ -èsima el cost de **A** és  $\Theta(i)$ . Per tant, en total la contribució al cost és  $\sum_{i=1}^n \Theta(i) = \Theta(\sum_{i=1}^n i) = \Theta(n^2)$ .
- B.** Si el cost de  $g(m)$  és  $\Theta(m)$ , llavors cada vegada que la condició de l'if és certa el cost de **B** és  $\Theta(n)$ . Com que això passa  $\Theta(\log n)$  vegades, la contribució al cost total és  $\Theta(n \log n)$ .

En total doncs el cost de  $h(n)$  és  $\Theta(n^2) + \Theta(n \log n) = \Theta(n^2)$ .

(b) Considerem els dos mateixos grups d'instruccions de l'apartat anterior, i de nou comptem per separat la seva contribució al cost total:

- A.** Si el cost de  $f(m)$  és  $\Theta(1)$ , llavors el cost de **A** a cada iteració és  $\Theta(1)$ . Com que es fan  $\Theta(n)$  voltes, el cost en total és  $\Theta(n)$ .
- B.** Si el cost de  $g(m)$  és  $\Theta(m^2)$ , llavors cada vegada que la condició de l'if és certa el cost de **B** és  $\Theta(n^2)$ . Com que això passa  $\Theta(\log n)$  vegades, el cost és  $\Theta(n^2 \log n)$ .

En total doncs el cost de  $h(n)$  és  $\Theta(n) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$ .

**Proposta de solució al problema 3**

(a) La fila  $i$ -èsima ocupa  $i + 1$  caselles ( $0 \leq i < n$ ). En total, l'espai usat és

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

(b) Una possible solució:

```

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

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

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

```

(c) Sigui  $C(N)$  el cost en el cas pitjor d'una crida a la funció  $\text{misteri}(k, l, r)$ , on  $N = r - l + 1$ . La recurrència que descriu  $C(N)$  és

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

ja que es fa una crida recursiva sobre un interval de mida la meitat, més operacions que tenen cost constant. Usant el teorema mestre de recurrències divisores, es té que la solució de la recurrència és  $C(N) = \Theta(\log N)$ .

Per tant, el cost en el cas pitjor d'una crida a la funció  $\text{fila\_columna}(n, k)$  és  $\Theta(\log n)$ .

(d) L'índex  $i$  de la fila buscada és el natural  $0 \leq i < n$  tal que  $p(i) \leq k < p(i + 1)$ . El podem calcular resolent l'equació de segon grau  $p(x) = k$  i prenent  $i = \lfloor x \rfloor$ :

```

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

pair<int,int> fila_columna(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)};
}

```

**Proposta de solució al problema 4**

(a) Demostrem-ho per reducció a l'absurd. Suposem que  $i$  és tal que  $0 \leq i < n - 1$  i  $f_A(i + 1) < f_A(i)$ . Prenem  $k = i + 1$ ,  $j = f_A(i + 1)$  i  $l = f_A(i)$ , de forma que  $0 \leq i < k < n$  i  $0 \leq j < l < n$ . Com que  $j = f_A(k)$ , tenim que  $A_{k,j} \leq A_{k,l}$ . I com que  $l = f_A(i)$ , tenim que  $A_{i,l} \leq A_{i,j}$ ; de fet, com que  $j < l$ , ha de ser  $A_{i,l} < A_{i,j}$ . Per tant, sumant tenim que  $A_{i,l} + A_{k,j} < A_{i,j} + A_{k,l}$ . Això contradiu que  $A$  sigui una matriu de Monge.

- (b) Per cada  $i$  parell, es pot calcular la columna on apareix el mínim més a l'esquerra de la fila  $i$  de  $A$  de la manera següent. A partir de la crida recursiva sobre  $B_1$ , tenim la columna  $j_1$  on apareix el mínim més a l'esquerra de la fila  $i$  de  $A$  entre les columnes  $0$  i  $\frac{n}{2} - 1$ . Similarment, a partir de la crida recursiva sobre  $B_2$ , tenim la columna  $j_2$  on apareix el mínim més a l'esquerra de la fila  $i$  de  $A$  entre les columnes  $\frac{n}{2}$  i  $n - 1$ . Per tant,  $f_A(i) = j_1$  si  $A_{i,j_1} \leq A_{i,j_2}$ , i  $f_A(i) = j_2$  altrament.

Per cada  $i$  senar, es determina  $f_A(i)$  recorrent les columnes entre  $f_A(i - 1)$  i  $f_A(i + 1)$  (definim  $f_A(n) = n - 1$  per conveniència notacional), i escollint l'índex on apareix el mínim més a l'esquerra. Això té cost  $\Theta(f_A(i + 1) - f_A(i - 1) + 1)$ . En total:

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

ja que  $0 \leq f_A(0) < n$ .

- (c) Sigui  $C(n)$  el cost de l'algorisme proposat per calcular la funció  $f_A$  per a totes les files d'una matriu de Monge  $A$  de mida  $n \times n$ .

A banda de les dues crides recursives del pas (2) sobre matrius de mida  $\frac{n}{2} \times \frac{n}{2}$ , els passos (1) i (3) requereixen temps  $\Theta(n)$  en total. Per tant, la recurrència que descriu el cost és

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

Usant el teorema mestre de recurrències divisores, es té que la solució de la recurrència és  $C(n) = \Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

07/11/2016

## Proposta de solució al problema 1

	<i>Cas millor</i>	<i>Cas mig</i>	<i>Cas pitjor</i>
(a) Quicksort (amb partició de Hoare)	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Inserció	$\Theta(n)$	<b>No ompliu</b>	$\Theta(n^2)$

(b)  $\Theta(\sqrt{n})$ (c)  $\Theta(\sqrt{n} \log n)$ (d)  $\Theta(n)$ (e) L'algorisme de Karatsuba calcula el producte de dos nombres naturals de  $n$  bits en temps  $\Theta(n^{\log_2 3})$ .(f) L'algorisme de Strassen calcula el producte de dues matrius de mida  $n \times n$  en temps  $\Theta(n^{\log_2 7})$ .

## Proposta de solució al problema 2

(a) Una possible solució:

**#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) Sigui  $C(n)$  el cost de tractar un vector de mida  $n$  (sigui amb la funció *search* o amb la funció *dic\_search*) en el cas pitjor (per exemple, quan l'element  $x$  no apareix a la seqüència). A banda d'operacions de cost constant (càlculs aritmètics, comparacions i assignacions entre enters, accessos a vectors), es fa exactament una crida sobre un vector de mida la meitat de la de l'entrada. Així doncs, el cost ve determinat per la recurrència:

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

que, pel teorema mestre de recurrències divisores, té solució  $C(n) = \Theta(\log(n))$ .

### Proposta de solució al problema 3

- (a) Calcula  $m^n$ .
- (b) El cost de la funció ve determinat pel cost del bucle. Cada iteració requereix temps  $\Theta(1)$ , ja que només s'hi duen a terme operacions aritmètiques i assignacions d'enters. Per tant, el cost és proporcional al nombre d'iteracions. Observem que si  $y$  és parell, aleshores  $y$  es redueix a la meitat. I si  $y$  és un senar amb  $y > 1$ , a la següent iteració es considera  $y - 1$ , que és parell, i llavors es redueix a la meitat. Així doncs, si  $n \geq 1$  el nombre de voltes està entre  $1 + \lfloor \log(n) \rfloor$  i  $1 + 2\lfloor \log(n) \rfloor$ . En conclusió, el cost és  $\Theta(\log(n))$ .

### Proposta de solució al problema 4

- (a) Tenim que  $\phi - 1 = \frac{\sqrt{5}+1}{2} - 1 = \frac{\sqrt{5}-1}{2}$ , i aleshores

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

De forma que efectivament  $\phi^{-1} = \phi - 1$ .

- (b) Per inducció sobre  $n$ :

- **Cas base  $n = 0$ :** per una banda  $F(0) = 0$ , i per una altra

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

- **Cas base  $n = 1$ :** per una banda  $F(1) = 1$ , i per una altra

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

- **Cas inductiu  $n > 1$ :** per hipòtesi d'inducció,

$$\begin{aligned} F(n) &= F(n-2) + F(n-1) = \frac{\phi^{n-2} - (-\phi)^{-(n-2)}}{\sqrt{5}} + \frac{\phi^{n-1} - (-\phi)^{-(n-1)}}{\sqrt{5}} = \\ &= \frac{\phi^{n-1}(\phi^{-1} + 1) - (-\phi)^{-(n-1)}(-\phi + 1)}{\sqrt{5}} = \frac{\phi^{n-1} \cdot \phi - (-\phi)^{-(n-1)}(-\phi^{-1})}{\sqrt{5}} = \\ &= \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} \end{aligned}$$

(c) Tenim que

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

donat que  $\phi > 1$ ,  $\phi^2 > 1$  i  $\lim_{n \rightarrow +\infty} (\frac{-1}{\phi^2})^n = 0$ . Per tant  $F(n) = \Theta(\phi^n)$ .

## Solució de l'Examen Parcial EDA

20/04/2017

## Proposta de solució al problema 1

- (a) L'algorisme d'ordenació per inserció té cost  $\Theta(n)$  quan el vector està ordenat, i cost  $\Theta(n^2)$  quan està ordenat del revés. Així doncs, el cost mig és:

$$\left(1 - \frac{\log n}{n}\right)\Theta(n) + \frac{\log n}{n}\Theta(n^2) = \Theta\left(n - \log n + n \log n\right) = \Theta(n \log n)$$

- (b) La funció retorna si  $n$  és un nombre primer. El cost està determinat pel bucle, que fa  $O(\sqrt{n})$  iteracions, cadascuna de les quals de cost constant. Així doncs, el cost és  $O(\sqrt{n})$ .
- (c) Cadascun dels nombres que es multipliquen a  $n!$  són menors o iguals que  $n$ . Com que se'n multipliquen  $n$ , es té  $n! \leq n^n$  per tot  $n \geq 1$ . D'aquí  $n! = O(n^n)$ , prenent  $n_0 = 1$  i  $C = 1$ .
- (d) Llevat de 1, tots els nombres que es multipliquen a  $n!$  són majors o iguals que 2. Com que se'n multipliquen  $n - 1$ , es té  $n! \geq 2^{n-1} = \frac{1}{2} \cdot 2^n$  per tot  $n \geq 1$ . D'aquí  $n! = \Omega(2^n)$ , prenent  $n_0 = 1$  i  $C = \frac{1}{2}$ .

## Proposta de solució al problema 2

- (a) Una possible solució:

```

int top_rec (const vector<int>& a, int l, int r) {
    if (l + 1 ≥ r) {
        if (a[l] < a[r]) return r;
        else return l;
    }
    int m = (l+r)/2;
    if (a[m-1] > a[m]) return top_rec(a, l, m-1);
    if (a[m+1] > a[m]) return top_rec(a, m+1, r);
    return m;
}

int top(const vector<int>& a) {
    return top_rec(a, 0, a.size()-1);
}

```

Sigui  $C(n)$  el cost en temps en el cas pitjor de la funció `top_rec` sobre un vector de mida  $n$ . En el cas pitjor (per exemple, quan el cim és en un extrem del vector) es farà una crida recursiva sobre un subvector de mida  $n/2$ , a més d'operacions de cost constant (càlculs aritmètics, comparacions i assignacions entre enters, accessos a vectors). Així doncs tenim la recurrència  $C(n) = C(n/2) + \Theta(1)$ , que pel teorema mestre de recurrències divisores té solució  $C(n) = \Theta(\log n)$ .

- (b) Una possible solució que usa la funció `int top(const vector<int>& a)` de l'apartat anterior i la funció `binary_search` de la STL:

```

bool search(const vector<int>& a, int x) {
    int t = top(a);

```

```

int n = a.size ();
if ( binary_search (a.begin (), a.begin () + t, x)) return true;
if ( binary_search (a.rbegin (), a.rbegin () + n - t, x)) return true;
return false;
}

```

En una altra possible solució, la funció *search* anterior es pot reemplaçar per:

```

bool bin_search_inc (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 bin_search_inc (a, m+1, r, x);
    if (a[m] > x) return bin_search_inc (a, l, m-1, x);
    return true;
}

bool bin_search_dec (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 bin_search_dec (a, l, m-1, x);
    if (a[m] > x) return bin_search_dec (a, m+1, r, x);
    return true;
}

bool search (const vector<int>& a, int x) {
    int t = top(a);
    int n = a.size ();
    if ( bin_search_inc (a, 0, t-1, x)) return true;
    if ( bin_search_dec (a, t, n-1, x)) return true;
    return false;
}

```

Quan la funció *search* busca en un vector de mida  $n$ , a més de cridar la funció *top*, també es fan una o dues cerques dicotòmiques, cadascuna de les quals té cost  $O(\log n)$ , i operacions de cost constant. Per tant, el cost de *search* és  $O(\log n) + O(\log n) + O(1) = O(\log n)$ . A més, si per exemple el cim de la seqüència és en un extrem del vector, aleshores el cost és  $\Theta(\log n) + O(\log n) + O(1) = \Theta(\log n)$ . Per tant, el cost en el cas pitjor és  $\Theta(\log n)$ .

### Proposta de solució al problema 3

(a) Després de  $m$  crides a la funció *reserve* sobre un vector inicialment buit, la capacitat del vector és de  $C(m) = Am$  elements. Per tant, el nombre de crides fetes a *reserve* després de  $n$  crides a *push\_back* és el menor  $m$  tal que  $Am \geq n$ , és a dir,  $\lceil \frac{n}{A} \rceil$ . Com que  $A$  és constant,  $m$  és  $\Theta(n)$ .

(b) Per inducció.

- **Cas base:**  $m = 0$ . Tenim que  $\frac{BA^m - B}{A - 1} \Big|_{m=0} = \frac{B - B}{A - 1} = 0 = C(0)$ .

- **Cas inductiu:** suposant que és cert per a  $m$ , vegem que és cert per a  $m + 1$ . Per hipòtesi d'inducció es té  $C(m) = \frac{BA^m - B}{A - 1}$ . Aleshores:

$$C(m + 1) = AC(m) + B = A \frac{BA^m - B}{A - 1} + B = \frac{BA^{m+1} - AB + AB - B}{A - 1} = \frac{BA^{m+1} - B}{A - 1}$$

- (c) Després de  $m$  crides a la funció *reserve* sobre un vector inicialment buit, usant l'apartat anterior tenim que la capacitat del vector és de  $C(m) = \frac{BA^m - B}{A - 1}$  elements. Per tant, el nombre de crides fetes a *reserve* després de  $n$  crides a *push\_back* és el menor  $m$  tal que  $\frac{BA^m - B}{A - 1} \geq n$ , és a dir,  $\lceil \log_A(\frac{n(A-1)+B}{B}) \rceil$ . Com que  $A$  i  $B$  són constants,  $m$  és  $\Theta(\log n)$ .

#### Proposta de solució al problema 4

- (a) Una possible solució:

```
int stable_partition (int x, vector<int>& a) {
    int n = a.size ();
    vector<int> w(n);
    int i = 0;
    for (int y : a)
        if (y ≤ x) {
            w[i] = y;
            ++i;
        }
    int r = i-1;
    for (int y : a)
        if (y > x) {
            w[i] = y;
            ++i;
        }
    for (int k = 0; k < n; ++k)
        a[k] = w[k];

    return r;
}
```

El cost en temps és  $\Theta(n)$ , ja que es fan 3 recorreguts sobre el vector, cada iteració dels quals només requereix temps constant. El cost en espai de la memòria auxiliar està dominat pel vector  $w$ , que té mida  $n$ . Així, el cost en espai és  $\Theta(n)$ .

- (b) Transposa els dos subvectors de  $a$  de  $l$  a  $p$  i de  $p + 1$  a  $r$ : si abans de la crida  $a[l..r]$  és  $A_l, \dots, A_p, A_{p+1}, \dots, A_r$ , després de la crida  $a[l..r]$  és  $A_{p+1}, \dots, A_r, A_l, \dots, A_p$ .

- (c) Solució:

```
int stable_partition (int x, vector<int>& a) {
    return stable_partition_rec (x, a, 0, a.size ()-1);
}

int stable_partition_rec (int x, vector<int>& a, int l, int r) {
    if (l == r) {
        if (a[l] ≤ x) return l;
        else return l-1;
    }
```

```
}  
int m = (l+r)/2;  
int p = stable_partition_rec (x, a, l, m);  
int q = stable_partition_rec (x, a, m+1, r);  
mystery(a, p+1, m, q);  
return p+q-m;  
}
```

Sigui  $C(n)$  el cost de la funció *stable\_partition\_rec* sobre un vector de mida  $n = r - l + 1$  en el cas pitjor. Per una banda es fan dues crides recursives sobre vectors de mida  $n/2$ . Per una altra, el cost del treball no recursiu està dominat per la funció *mystery*, que triga temps lineal en la mida del vector que transposa. Usant la hipòtesi de l'enunciat (que es dóna, per exemple, en un vector en què es van alternant successivament elements més petits i més grans que  $x$ ), aquest vector té mida  $\Theta(n)$ . Així doncs tenim la recurrència  $C(n) = 2C(n/2) + \Theta(n)$ , que pel teorema mestre de recurrències divisores té solució  $\Theta(n \log n)$ . En conclusió, el cost de la funció *stable\_partition* sobre un vector de mida  $n$  en el cas pitjor és  $\Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

06/11/2017

## Proposta de solució al problema 1

- (a) El cost és  $\Theta(n\sqrt{n})$ .
- (b) El cost tant en el cas pitjor com en el cas mig és  $\Theta(n)$ .
- (c)  $\Theta(\sqrt{n})$ .
- (d)  $\Theta(n^2 \log n)$ .
- (e)  $\Theta(2^n)$ .

## Proposta de solució al problema 2

- (a) El valor  $b[y]$  compta el nombre d'elements de la seqüència  $a$  que són més petits o iguals que  $y$ .
- (b) Una possible solució:

```
int median(int n, const vector<int>& b) {
    int l = 0;
    int r = b.size() - 1;
    while (l < r) {
        int q = (l+r)/2;
        if (b[q] < b[(l+r)/2]) l = q;
        else r = q;
    }
    return r;
}
```

- (c) A cada volta del bucle, el valor  $r - l$  es redueix per la meitat. Com que inicialment  $l = 0$  i  $r = m$ , es fan  $\Theta(\log m)$  voltes. I com que cada volta té cost constant (operacions aritmètiques d'enters, accessos a vectors, etc.), concloem que el cost de la funció és  $\Theta(\log m)$ .

## Proposta de solució al problema 3

- (a) Una possible solució:

```
complex operator*(const complex& a, const complex& b) {
    return {a.real * b.real - a.imag * b.imag, a.real * b.imag + a.imag * b.real};
}

complex exp(complex z, int n) {
    if (n == 0) return {1, 0};
    complex x = exp(z, n/2);
    complex y = x*x;
    if (n % 2 == 1) y = y*z;
    return y;
}
```

- (b) El cost  $C(n)$  de la funció recursiva *exp* en funció de  $n$  ve determinat per la recurrència  $C(n) = C(n/2) + \Theta(1)$ : es fa una sola crida recursiva sobre un problema de mida  $n/2$ , i la resta d'operacions tenen cost constant. Aplicant el teorema mestre de recurrències divisores, tenim que la solució és  $C(n) = \Theta(\log n)$  tal com es demanava.

**Proposta de solució al problema 4**

- (a) Tenim que

$$U(m) = T(2^m) = T(2^m/2) + \log(2^m) = T(2^{m-1}) + \log(2^m) = U(m-1) + m$$

- (b) Aplicant el teorema mestre de recurrències substractores per a resoldre la recurrència  $U(m) = U(m-1) + m$ , tenim que  $U(m) = \Theta(m^2)$ .
- (c) Com que  $U(m) = \Theta(m^2)$  i  $U(m) = T(2^m)$ , tenim que  $T(n) = T(2^{\log n}) = U(\log n) = \Theta((\log n)^2)$ .



**Solució de l'Examen Parcial EDA****19/04/2018****Proposta de solució al problema 1**(a) La funció  $g$  creix més de pressa:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n^{n^2}}{2^{2^n}} = \lim_{n \rightarrow \infty} 2^{\log(\frac{n^{n^2}}{2^{2^n}})} = 2^{\lim_{n \rightarrow \infty} \log(\frac{n^{n^2}}{2^{2^n}})} = 0$$

perquè

$$\lim_{n \rightarrow \infty} \log\left(\frac{n^{n^2}}{2^{2^n}}\right) = \lim_{n \rightarrow \infty} (\log(n^{n^2}) - \log(2^{2^n})) = \lim_{n \rightarrow \infty} (n^2 \log n - 2^n) = -\infty$$

(b) El cost de l'algorisme en el cas mig és  $\Theta(n^2)$ :

$$\frac{1}{n^3} \cdot \Theta(n^4) + \frac{1}{n} \cdot \Theta(n^3) + \left(1 - \frac{1}{n^3} - \frac{1}{n}\right) \cdot \Theta(n) = \Theta(n) + \Theta(n^2) + \Theta(n) = \Theta(n^2)$$

(c) Usant el teorema mestre de recurrències divisores tenim que  $a = 2$ ,  $b = 4$ ,  $\alpha = \log_4 2 = \frac{1}{2}$  i  $k = \frac{1}{2}$ , de forma que  $T(n) = \Theta(\sqrt{n} \cdot \log n)$ .**Proposta de solució al problema 2**(a) El cas base per a  $k = 0$  és cert:  $A^0 \cdot x_0 = x_0 = x(0)$ . Pel cas inductiu, quan  $k > 0$ , tenim per hipòtesi d'inducció que  $x(k-1) = A^{k-1} \cdot x_0$ . Per tant,  $x(k) = A \cdot x(k-1) = A \cdot (A^{k-1} \cdot x_0) = (A \cdot A^{k-1}) \cdot x_0 = A^k \cdot x_0$ .(b) Es calcula en primer lloc  $A^k$  usant l'algorisme d'exponenciació ràpida en temps  $\Theta(\log k)$  (ja que la  $n$  és constant). Després es retorna el resultat de multiplicar  $A^k$  per  $x_0$ , que es pot fer en temps  $\Theta(1)$  (de nou, perquè la  $n$  és constant). El cost en temps de l'algorisme és doncs  $\Theta(\log k)$ .**Proposta de solució al problema 3**(a)  $x = x_2 \cdot 3^{2n/3} + x_1 \cdot 3^{n/3} + x_0$ .(b)  $x \cdot y = x_2 y_2 \cdot 3^{4n/3} + (x_1 y_2 + x_2 y_1) \cdot 3^{3n/3} + (x_0 y_2 + x_1 y_1 + x_2 y_0) \cdot 3^{2n/3} + (x_1 y_0 + x_0 y_1) \cdot 3^{n/3} + x_0 y_0$ .(c)  $x \cdot y =$ 

$$x_2 y_2 \cdot 3^{4n/3}$$

$$+ (x_1 y_2 + x_2 y_1) \cdot 3^{3n/3}$$

$$+ ((x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2) - x_2 y_2 - (x_1 y_2 + x_2 y_1) - (x_1 y_0 + x_0 y_1) - x_0 y_0) \cdot 3^{2n/3}$$

$$+ (x_1 y_0 + x_0 y_1) \cdot 3^{n/3}$$

$$+ x_0 y_0$$

Es fan servir 7 productes.

- (d) Per a calcular el producte de  $x$  i  $y$  de  $n$  dígit, l'algorisme calcula recursivament  $(x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2)$ ,  $x_2y_2$ ,  $x_1y_2 + x_2y_1$ ,  $x_1y_0 + x_0y_1$  i  $x_0y_0$ , que són productes de nombres de  $n/3$  dígit. Llavors es calcula  $x \cdot y$  usant l'equació de l'apartat anterior. Com que els nombres es representen en base 3, multiplicar per una potència de 3 és afegir zeros a la dreta i es pot fer en temps  $\Theta(n)$ . Les sumes involucrades també es poden fer en temps  $\Theta(n)$ . Així doncs el cost  $T(n)$  satisfà la recurrència  $T(n) = 7T(n/3) + \Theta(n)$ , que té solució  $\Theta(n^{\log_3 7})$ .

#### Proposta de solució al problema 4

- (a) Definim la funció  $U(m) = T(b^m)$ . Llavors  $T(n) = U(\log_b(n))$ . A més, tenim:

$$\begin{aligned} U(m) = T(b^m) &= T(b^m/b) + \Theta(\log^k(b^m)) = T(b^{m-1}) + \Theta(m^k \log^k(b)) = \\ &= T(b^{m-1}) + \Theta(m^k) = U(m-1) + \Theta(m^k) \end{aligned}$$

El teorema mestre de recurrències substractores afirma que si tenim una recurrència de la forma  $U(m) = U(m-c) + \Theta(m^k)$  amb  $c > 0$  i  $k \geq 0$ , llavors  $U(m) = \Theta(m^{k+1})$ . Per tant la solució a la recurrència de l'enunciat és  $T(n) = \Theta((\log_b(n))^{k+1}) = \Theta(\log^{k+1} n)$ .

- (b) Una possible solució:

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

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

- (c) El cas pitjor es dona per exemple quan  $x$  no apareix a  $a$ . En aquesta situació el cost  $T(n)$  ve descrit per la recurrència  $T(n) = T(n/2) + \Theta(\log n)$ , perquè es fa una crida recursiva sobre un vector de mida  $\frac{n}{2}$  i el cost del treball no recursiu està dominat per la cerca binària, que té cost  $\Theta(\log(\frac{n}{2})) = \Theta(\log(n))$ . Aplicant el primer apartat tenim que la solució és  $T(n) = \Theta(\log^2(n))$ .

## Solució de l'Examen Parcial EDA

05/11/2018

## Proposta de solució al problema 1

(a) Les respostes:

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
CERT	X		X	X		X			X	X
FALS		X			X		X	X		

## Proposta de solució al problema 2

(a) El cost en el cas pitjor és  $\Theta(n^3)$ . El cas pitjor es dona sempre.(b) El cost en el cas millor és  $\Theta(n^2)$ . El cas millor es dona, per exemple, quan les dues matrius només tenen *true* en els seus coeficients.

El cost en el cas pitjor és  $\Theta(n^3)$ . El cas pitjor es dona, per exemple, quan les dues matrius només tenen *false* en els seus coeficients.

(c) La funció considera les matrius booleanes com a matrius d'enters, en què *false* s'interpreta com 0 i *true* com 1. Aleshores s'aplica l'algorisme de producte de matrius d'Strassen. Sigui  $M$  la matriu resultat. El coeficient de la fila  $i$ -èsima i columna  $j$ -èsima de  $M$  és

$$m_{ij} = \sum_{k=0}^{n-1} (a_{ik} \cdot b_{kj}).$$

Com que els coeficients de les matrius d'entrada són 0 o 1, el producte com a nombres enters és el mateix que l'operació  $\wedge$  lògica. De forma que  $m_{ij}$  compta el nombre de parells  $(a_{ik}, b_{kj})$  on ambdós  $a_{ik}$  i  $b_{kj}$  són certs a la vegada. Així doncs, per obtenir el producte lògic només cal definir  $p_{ij}$  com 1 si  $m_{ij} > 0$ , i 0 altrament. El cost està dominat per l'algorisme d'Strassen, que té cost  $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$ .

## Proposta de solució al problema 3

(a) Donat un enter  $n \geq 0$ , la funció *mystery* calcula  $\lfloor \sqrt{n} \rfloor$ .

(b) En primer lloc trobem el cost  $C(N)$  de la funció *mystery\_rec* en termes de la mida  $N = r - l + 1$  de l'interval  $[l, r]$ . Aquest cost segueix la recurrència  $C(N) = C(N/2) + \Theta(1)$ , ja que a cada crida es fa una crida recursiva sobre un interval amb la meitat d'elements, i treball addicional no recursiu de cost constant. D'acord amb el teorema mestre de recurrències divisores, la solució d'aquesta recurrència és  $\Theta(\log N)$ . Com que *mystery*( $n$ ) consisteix en cridar *mystery\_rec*( $n, 0, n+1$ ), podem concloure que el cost de *mystery*( $n$ ) és  $\Theta(\log n)$ .

## Proposta de solució al problema 4

(a)  $\Theta(n)$ (b)  $\Theta(n^2)$ (c)  $\Theta(n \log n)$ (d)  $\Theta(n \log n)$

(e) Una possible solució:

```
void my_sort(vector<int>& v) {
    int n = v.size ();
    double lim = n * log(n);
    int c = 0;
    for (int i = 1; i < n; ++i) {
        int x = v[i];
        int j;
        for (j = i; j > 0 and v[j - 1] > x; --j) {
            v[j] = v[j - 1];
            ++c;
        }
        v[j] = x;
        if (c > lim) {
            merge_sort(v);
            return;
        }
    }
}
```

(f) En primer lloc observem que, com que el bucle **for** extern fa  $n - 1$  voltes, cadascuna de les quals té cost  $\Omega(1)$ , el cost de tota execució és  $\Omega(n)$ .

Si per exemple el vector està ja ordenat de forma creixent, aleshores *my\_sort* es comporta com l'ordenació per inserció: no s'entra mai al bucle **for** intern,  $c$  és sempre 0, i no es crida *merge\_sort*. Com que cada volta del bucle **for** extern té cost constant, el cost en aquest cas és  $\Theta(n)$ . De forma que el cost de *my\_sort* en el cas millor és  $\Theta(n)$ .

Per veure el cost en el cas pitjor, distingim dos casos:

- Suposem que no s'arriba a cridar *merge\_sort*. Aleshores el cost és proporcional al valor final de la variable  $c$ . Com que no es crida *merge\_sort*, tenim que  $c \leq n \ln n$ , i el cost és  $O(n \ln n) = O(n \log n)$ .
- Suposem que es crida *merge\_sort*. Si es crida al final de la primera volta del bucle **for** extern, aleshores el cost és  $O(n)$  del bucle **for** intern més  $\Theta(n \log n)$  del *merge\_sort*. En total, el cost és  $\Theta(n \log n)$ .

Si *merge\_sort* es crida al final de la segona, o tercera, etc. volta del bucle **for** extern, aleshores a la iteració anterior del bucle **for** extern no s'ha cridat *merge\_sort*. A més, en la darrera iteració  $c$  com a molt pot haver augmentat en  $i$ , de forma que en el moment de cridar *merge\_sort*, tenim que  $n \ln n < c \leq i + n \ln n \leq n + n \ln n \leq n \ln n + n \ln n = 2n \ln n$  (per  $n$  prou gran), i per tant  $c = \Theta(n \log n)$ . Com que el cost de *merge\_sort* és  $\Theta(n \log n)$ , en total el cost és  $\Theta(n \log n)$ .

El cas pitjor es dona per exemple quan el vector està ordenat al revés, és a dir, de forma decreixent. Com que en aquest cas l'ordenació per inserció té cost  $\Theta(n^2)$ , en algun moment de l'execució de *my\_sort* es cridarà *merge\_sort*, i pel raonament anterior el cost serà  $\Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

25/04/2019

## Proposta de solució al problema 1

(a) Les respostes:

(1)  $f = O(g) : \text{si } \alpha > 1.$

(2)  $f = \Omega(g) : \text{si } \alpha \leq 1.$

(3)  $g = O(f) : \text{si } \alpha \leq 1.$

(4)  $g = \Omega(f) : \text{si } \alpha > 1.$

(b)  $T(n) = \Theta(2^{\frac{n}{2}}) = \Theta(\sqrt{2}^n)$

(c)  $T(n) = \Theta(n \log n)$

(d)  $\Theta(n \log n)$

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

## Proposta de solució al problema 2

(a) Una crida  $mystery(v, 0, v.size() - 1, m)$  permuta els elements de  $v$  de forma que (almenys) les  $m$  primeres posicions de  $v$  estiguin ocupades pels  $m$  elements més petits, ordenats de forma creixent.

(b) Quan es crida  $mystery(v, 0, n-1, n)$ , primer s'executa  $\text{int } q = \text{partition}(v, l, r)$  amb  $l = 0$  i  $r = n-1$ , i es triga  $\Theta(n)$ . A més, com que el vector és d'enters diferents ordenats creixentment i la funció  $\text{partition}$  pren com a pivot l'element de més a l'esquerra, tindrem  $q = 0$ . De forma que la crida  $mystery(v, l, q, m)$  es fa amb  $q = l = 0$  i per tant triga temps constant. A més, llavors  $p = 1$ . Si  $n > 1$  finalment es fa una crida recursiva  $mystery(v, q+1, r, m-p)$ , on  $q+1 = 1, r = n-1$  i  $m-p = n-1$ .

A la seva vegada, en executar aquesta crida el cost de  $\text{int } q = \text{partition}(v, l, r)$  és  $\Theta(n-1)$ , de nou el cost de  $mystery(v, l, q, m)$  és  $\Theta(1)$ , i si  $n-1 > 1$  es fa de nou una crida recursiva  $mystery(v, q+1, r, m-p)$ , on  $q+1 = 2, r = n-1$  i  $m-p = n-2$ .

Repetint l'argument, veiem que el cost acumulat és

$$\Theta(n) + \Theta(n-1) + \dots + \Theta(1) = \Theta\left(\sum_{i=1}^n i\right) = \Theta(n^2).$$

## Proposta de solució al problema 3

(a) Una possible solució:

```
vector<bool> prod(const vector<bool>& x, const vector<bool>& y) {
    if (x.size() == 0 or y.size() == 0) return vector<bool>();
    vector<bool> z = twice(twice(prod(half(x), half(y))));
```

```

vector<bool> one = vector<bool>(1, 1);

if      (x.back() == 0 and y.back() == 0) return z;
else if (x.back() == 1 and y.back() == 0) return sum(z, y);
else if (y.back() == 1 and x.back() == 0) return sum(z, x);
else {
    vector<bool> x2 = twice(half(x));
    vector<bool> y2 = twice(half(y));
    return sum(sum(sum(z, x2), y2), one);
}
}

```

Segui  $T(n)$  el cost de  $prod(x, y)$  si  $n = x.size() = y.size()$ . Es fa una única crida recursiva sobre vectors de mida  $n - 1$ . El treball no recursiu té cost  $\Theta(n)$ . Per tant el cost segueix la recurrència

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

D'acord amb el teorema mestre de recurrències subtractives, la solució d'aquesta recurrència es comporta asimptòticament com  $\Theta(n^2)$ . Per tant, el cost és  $\Theta(n^2)$ .

(b) L'algorisme de Karatsuba, que té cost  $\Theta(n^{\log 3})$ .

#### Proposta de solució al problema 4

(a) Una possible forma de completar el codi:

```

bool search1(int x, const vector<vector<int>>& A, int i, int j, int n) {
    if (n == 1) return A[i][j] == x;
    int mi = i + n/2 - 1;
    int mj = j + n/2 - 1;

    if (A[mi][mj] < x) return
        search1(x, A, mi+1, j, n/2) or
        search1(x, A, mi+1, mj+1, n/2) or
        search1(x, A, i, mj+1, n/2);

    if (A[mi][mj] > x) return
        search1(x, A, mi+1, j, n/2) or
        search1(x, A, i, j, n/2) or
        search1(x, A, i, mj+1, n/2);

    return true;
}

```

Observem que el resultat de cridar  $search1(x, A, 0, 0, N)$  és **true** si i només si  $x$  apareix a  $A$ .

Per analitzar el cost d'aquesta crida, en primer lloc estudiem el cas general de cridar  $search1(x, A, i, j, n)$  en funció de  $n$ . Segui  $T(n)$  el cost en el cas pitjor d'aquesta crida. Com que en el cas pitjor es fan 3 crides amb darrer paràmetre  $n/2$  i el cost del treball no recursiu és constant, tenim que se segueix la recurrència:

$$T(n) = 3T(n/2) + \Theta(1)$$

Aplicant el teorema mestre de recurrències divisores, tenim  $T(n) = \Theta(n^{\log_2 3})$ .

Així doncs, el cost de cridar  $search1(x, A, 0, 0, N)$  és  $\Theta(N^{\log_2 3})$ .

(b) És correcta. Vegem que el bucle manté el següent invariant: si  $x$  apareix a  $A$ , llavors ho fa entre la fila 0 i la  $i$  (incloses), i entre la columna  $j$  i la  $N - 1$  (incloses). En començar el bucle, l'invariant és cert. I a cada volta es manté:

- Si  $A[i][j] > x$  aleshores  $x$  no pot aparèixer a la fila  $i$ : de l'invariant tenim que si  $x$  apareix ha de ser en una columna de la  $j$  a la  $N - 1$ . Però si  $j \leq k < N$  llavors  $A[i][k] \geq A[i][j] > x$ .
- Si  $A[i][j] < x$  aleshores  $x$  no pot aparèixer a la columna  $j$ : de l'invariant tenim que si  $x$  apareix ha de ser en una fila de la 0 a la  $i$ . Però si  $0 \leq k \leq i$  llavors  $A[k][j] \leq A[i][j] < x$ .

Per últim, si el programa retorna **true** pel **return** de dins del bucle, la resposta és correcta perquè  $A[i][j] = x$ . Si retorna **false** pel **return** de fora, llavors  $i < 0$  o  $j \geq N$ . En qualsevol cas, de l'invariant deduïm que  $x$  no apareix a  $A$ .

(c) A cada volta o bé  $i$  es decrementa en 1, o bé  $j$  s'incrementa en 1, o bé es retorna. A més, inicialment  $i$  val  $N - 1$ , i com a mínim val 0 abans de sortir del bucle. Similarment, al principi  $j$  val 0 i com a molt val  $N - 1$  abans de sortir del bucle. Com que en el cas pitjor es poden fer  $2N - 1$  voltes i cadascuna té cost  $\Theta(1)$ , el cost és  $\Theta(N)$ .

## Solució de l'Examen Parcial EDA

11/11/2019

## Proposta de solució al problema 1

(a)  $\Theta(\sqrt{n} \log n)$ 

(b) Calculem:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log(\log n^2)}{\log n} &= \lim_{n \rightarrow \infty} \frac{\log(2 \log n)}{\log n} = \lim_{n \rightarrow \infty} \frac{\log 2 + \log(\log n)}{\log n} = \\ &= \lim_{n \rightarrow \infty} \frac{\log 2}{\log n} + \lim_{n \rightarrow \infty} \frac{\log(\log n)}{\log n} = \lim_{n \rightarrow \infty} \frac{\log(\log n)}{\log n} \end{aligned}$$

Fem el canvi de variable  $n = 2^m$  i obtenim que l'anterior és igual a

$$\lim_{m \rightarrow \infty} \frac{\log(\log 2^m)}{\log 2^m} = \lim_{m \rightarrow \infty} \frac{\log m}{m} = 0$$

Per tant únicament és cert que  $\log(n) \in \Omega(\log(\log(n^2)))$ 

## Proposta de solució al problema 2

(a) Retorna  $f \circ g$ , la composició de  $f$  amb  $g$ . El cost de *misteri* és el de la funció auxiliar *misteri\_aux*, que ve descrit per la recurrència  $T(n) = T(n-1) + \Theta(1)$ , que té com a solució asimptòtica  $T(n) \in \Theta(n)$ .

(b) Retorna  $f^k$ . És a dir, una funció tal que  $f^k(x) = \underbrace{f(f(\dots(f(x))))}_k$ . En funció de  $k$ , el seu cost ve donat per la recurrència  $T(k) = T(k-1) + \Theta(1)$ , que té com a solució  $\Theta(k)$ .

(c)

```
vector<int> misteri_2_quick(const vector<int>&f, int k) {
    if (k == 0) {
        vector<int> r(f.size());
        for (uint i = 0; i < f.size(); ++i) r[i] = i;
        return r;
    }
    else if (k%2 == 0) {
        vector<int> aux = misteri_2_quick(f, k/2);
        return misteri(aux, aux);
    }
    else {
        vector<int> aux = misteri_2_quick(f, k/2);
        return misteri(f, misteri(aux, aux));
    }
}
```

La recurrència que descriu el cost en temps d'aquesta funció és  $T(k) = T(k/2) + \Theta(1)$ , que té com a solució asimptòtica  $\Theta(\log k)$ .



**Proposta de solució al problema 3**

- (a) És fàcil veure que la funció *max\_suma* essencialment implementa una ordenació per selecció, que sabem que té cost en cas pitjor de  $\Theta(m^2)$ . L'única diferència és la línia on actualitzem *suma*, que triga temps constant i només s'executa  $m$  vegades. Per tant el cost total és  $\Theta(m^2) + \Theta(m) = \Theta(m^2)$ .
- (b) Si entenem el codi anterior ens podem adonar que ordena el vector de major a menor i agrupa els enters consecutivament de dos en dos seguint aquest ordre. Per millorar l'eficiència, només cal ordenar el vector amb un *merge sort*, de manera que el cost sigui  $\Theta(m \log m)$ , i agrupar els enters consecutivament de dos en dos. El cost asimptòtic en temps seria de  $\Theta(m \log m)$ .
- (c) Assumim que  $x_0$  i  $x_1$  són els dos nombres més grans de  $S$  i considerem una expressió que conté els productes  $x_0 * y$  i  $x_1 * z$ , per certs  $y, z \in S$ . El que farem és reemplaçar aquests dos productes per  $x_0 * x_1$  i  $y * z$ . Observem ara el següent:  $(x_0 * x_1 + y * z) - (x_0 * y + x_1 * z) = x_0(x_1 - y) + (y - x_1)z = x_0(x_1 - y) - (x_1 - y)z = (x_0 - z)(x_1 - y) > 0$ . L'últim pas és degut a que  $x_0 > z$  i  $x_1 > y$  ja que  $x_0$  i  $x_1$  són els elements majors de  $S$ , i són tots diferents. Per tant l'expressió original no era màxima ja que l'expressió resultant és major.

Anem a demostrar el resultat per inducció sobre  $m$ :

- *Cas base* ( $m = 0$ ). L'algorisme és correcte ja que retorna una expressió que suma zero i per tant és òptima.
- *Pas d'inducció*. Sigui  $m > 0$  i assumim la hipòtesi d'inducció: l'expressió màxima per un conjunt de  $< m$  elements es pot obtenir ordenant els elements de major a menor i agrupant-los de dos en dos consecutivament. Si ordenem els  $m$  elements  $x_0 > x_1 > x_2 > x_3 > \dots > x_{m-1}$ , sabem gràcies al resultat anterior que l'expressió òptima conté el producte  $x_0 * x_1$  seguit d'una expressió formada amb els nombres  $\{x_2, x_3, \dots, x_{m-1}\}$ . Aquesta expressió serà òbviament la major que puguem formar amb  $\{x_2, x_3, \dots, x_{m-1}\}$  i aplicant la hipòtesi d'inducció sabem que tindrà la forma  $x_2 * x_3 + \dots + x_{m-2} * x_{m-1}$ . Per tant, l'expressió òptima és  $x_0 * x_1 + x_2 * x_3 + \dots + x_{m-2} * x_{m-1}$ , com volíem demostrar.

**Proposta de solució al problema 4**

(a)

```
int f(const vector<int>& p, int l, int r){
    if (l + 1 ≥ r) return (p[l] ≤ p[r] ? l : r);
    else {
        int m = (l+r)/2;
        if (p[m] > p[m+1]) return f(p, m+1, r);
        else if (p[m-1] < p[m]) return f(p, l, m-1);
        else return m;
    }
}

pair<int,int> max_guany(const vector<int>& p) {
    return {f(p, 0, p.size()-1), p.size()-1};
}
```

El cost de *max\_guany* coincidirà amb el cost de la funció *f*. Per analitzar aquesta última, cal fixar-se que el seu cost ve donat per la recurrència  $T(n) = T(n/2) + \Theta(1)$ , d'on s'obté el cost de  $\Theta(\log n)$ .

(b)

```
int max_guany (const vector<int>& p, int k) {
    int m = p[k];
    for (int i = k - 1; i ≥ 0; --i)
        m = min(m, p[i]);

    int M = p[k];
    for (int i = k + 1; i < p.size (); ++i)
        M = max(M, p[i]);

    return M - m;
}
```

- (c) Podem utilitzar un algorisme de dividir i vèncer. Donat un vector *p*, el partim en dues meitats, separades pel punt mig *m*. Recursivament, calculem el màxim guany possible si comprem i venem a la part esquerra del vector, i a continuació, també recursivament, calculem el màxim guany possible si comprem i venem a la part dreta del vector. Finalment, utilitzant la funció de l'apartat anterior, calculem el màxim guany d'un període que inclou el punt mig *m* (és a dir, comprem a la part esquerra i venem a la part dreta). El resultat final és el màxim dels tres guanys calculats.

Hem desenvolupat un esquema de dividir i vèncer on fem dues crides recursives de mida la meitat, i a continuació fem un treball lineal per calcular el màxim guany que inclogui el punt *m*. Per tant, la recurrència que determina el cost de la funció és:  $T(n) = 2T(n/2) + \Theta(n)$ , que té com a solució asimptòtica  $\Theta(n \log n)$ .

*Nota:* hi ha solucions més eficients no basades en dividir i vèncer.

## Solució de l'Examen Parcial EDA

23/04/2020

## Proposta de solució al problema 2

Creuers, X35804:

```

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

struct Info {
    string code;
    int price ;
};

int main() {
    map<int, Info> M;
    char c;
    while (cin >> c) {
        if (c == 'n') {
            cout << "num: " << M.size() << endl;
        }
        else if (c == 'u') {
            string code;
            int length, price;
            cin >> code >> length >> price;
            Info inf; inf.price = price; inf.code = code;
            M[length]={code,price};
        }
        else if (c == 'q') {
            int length;
            cin >> length;
            if (M.count(length) == 1) cout << M[length].price << endl;
            else cout << -1 << endl;
        }
        else if (c == 'p') {
            cout << string(10, '-') << endl;
            for (auto& p : M)
                cout << p.second.code << " " << p.first << " " << p.second.price << endl;
            cout << string(10, '*') << endl;
        }
        else {
            if (M.size() < 2) cout << "no" << endl;
            else {
                auto it = M.begin(); ++it;
                cout << it->second.code << " " << it->first << " " << it->second.price << endl;
            }
        }
    }
}

```

Donacions, X22314:

```

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

int main() {
    map<string, int> M;
    char c;
    while (cin >> c) {
        if (c == 'N') {
            cout << "number: " << M.size() << endl;
        }
        else if (c == 'D') {
            string nif;
            int money;
            cin >> nif >> money;
            M[nif] += money;
        }
        else if (c == 'Q') {
            string nif;
            cin >> nif;
            if (M.count(nif) != 0) cout << M[nif] << endl;
            else cout << -1 << endl;
        }
        else if (c == 'P') {
            bool primer = true;
            for (auto& p : M) {
                if ((p.first[p.first.length()-2] - '0')%2 == 0) {
                    cout << (primer?" ":" ") << p.first;
                    if (primer) primer = false;
                }
            }
            cout << endl;
        }
        else { // c == 'L'
            if (M.size() == 0) cout << "NO LAST NIF" << endl;
            else {
                auto it = M.end(); --it;
                cout << it->first << " " << it->second << endl;
            }
        }
    }
}

```

Efemèrides, X79163:

```
#include <iostream>
#include <map>
using namespace std;

struct Data {
    string event;
    int relevance;
};

int main() {
    int maximum_relevance = 0;
    map<string, Data> M;
    char c;
    while (cin >> c) {
        if (c == 'n') {
            cout << "number events: " << M.size() << endl;
        }
        else if (c == 's') {
            string date, event;
            int relevance;
            cin >> date >> event >> relevance;
            if (M.count(date)) cout << "ERROR: repeated date" << endl;
            else {
                M[date] = {event, relevance};
                if (relevance > maximum_relevance) maximum_relevance = relevance;
            }
        }
        else if (c == 'a') {
            string date;
            cin >> date;
            cout << M[date].event << endl;
        }
        else if (c == 'm') {
            cout << "maximum relevance: " << maximum_relevance << endl;
        }
        else { // c == 'e'
            if (M.size() < 2) cout << "ERROR: at least two events needed" << endl;
            else {
                int total = 0;
                auto it = M.end();
                --it;
                total += it->second.relevance;
                it = M.begin();
                total += it->second.relevance;
                cout << total << endl;
            }
        }
    }
}
```

```

    }
  }
}

```

### Proposta de solució al problema 3

Vector xulo, X30043:

```

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

int position (const vector<int>& v, int e, int d) {
    if (e+1 == d) return e;
    int m = (e+d)/2;
    if (v[e] ≥ v[m]) return position (v, m, d);
    else return position (v, e, m);
}

int search (int x, const vector<int>& v, int e, int d) {
    if (e > d) return -1;
    if (e == d) return (v[e] == x ? e : -1);
    int m = (e + d)/2;
    if (x < v[m]) return search (x, v, m + 1, d);
    else return search (x, v, e, m);
}

int search (int x, const vector<int>& v) {
    int n = v.size ();
    int j = position (v, 0, n-1);
    int p = search (x, v, 0, j);
    if (p ≠ -1) return p;
    return search (x, v, j+1, n-1);
}

```

Vector xulo, X74873:

```

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

int position (const vector<int>& v, int e, int d) {
    if (e+1 == d) return e;
    int m = (e+d)/2;
    if (v[e] ≥ v[m]) return position (v, m, d);
    else return position (v, e, m);
}

int search (int x, const vector<int>& v, int e, int d) {
    if (e > d) return -1;

```

```

    if (e == d) return (v[e] == x ? e : -1);
    int m = (e + d + 1)/2;
    if (x > v[m]) return search(x, v, e, m - 1);
    else return search(x, v, m, d);
}

```

```

int search(int x, const vector<int>& v) {
    int n = v.size ();
    int j = position (v, 0, n-1);
    int p = search(x, v, 0, j);
    if (p != -1) return p;
    return search(x, v, j+1, n-1);
}

```

Vector xulo, X83303:

```

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

int position (const vector<int>& v, int e, int d) {
    if (e+1 == d) return e;
    int m = (e+d)/2;
    if (v[e] ≤ v[m]) return position (v, m, d);
    else return position (v, e, m);
}

int search(int x, const vector<int>& v, int e, int d) {
    if (e > d) return -1;
    if (e == d) return (v[e] == x ? e : -1);
    int m = (e + d)/2;
    if (x > v[m]) return search(x, v, m + 1, d);
    else return search(x, v, e, m);
}

int search(int x, const vector<int>& v) {
    int n = v.size ();
    int j = position (v, 0, n-1);
    int p = search(x, v, 0, j);
    if (p != -1) return p;
    return search(x, v, j+1, n-1);
}

```

Vector xulo, X90362:

```

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

int position (const vector<int>& v, int e, int d) {
    if (e+1 == d) return e;

```

```
int m = (e+d)/2;
if (v[e] ≤ v[m]) return position(v, m, d);
else return position(v, e, m);
}

int search(int x, const vector<int>& v, int e, int d) {
    if (e > d) return -1;
    if (e == d) return (v[e] == x ? e : -1);
    int m = (e + d + 1)/2;
    if (x < v[m]) return search(x, v, e, m - 1);
    else return search(x, v, m, d);
}

int search(int x, const vector<int>& v) {
    int n = v.size();
    int j = position(v, 0, n-1);
    int p = search(x, v, 0, j);
    if (p ≠ -1) return p;
    return search(x, v, j+1, n-1);
}
```



## Solució de l'Examen Parcial EDA

06/11/2020

## Proposta de solució al problema 1

```

(a)  bool tri_search (const vector<int>& v, int l, int r, int x) {
    if (l > r) return false;
    else {
        int n_elems = (r-l+1);
        int f = l + n_elems/3;
        int s = r - n_elems/3;
        if (v[f] == x or v[s] == x) return true;
        if (x < v[f]) return tri_search (v, l, f-1, x);
        if (x < v[s]) return tri_search (v, f+1, s-1, x);
        else return tri_search (v, s+1, r, x);
    }
}

```

En el cas pitjor es fan totes les crides recursives fins que  $l > r$ . La recurrència que expressa el cost del programa en aquest cas és:

$$T(n) = T(n/3) + \Theta(1)$$

que té solució  $T(n) \in \Theta(\log n)$ .

(b) Sigui  $f = n(\log n)^{1/2}$  i  $g = n(\log n)^{1/3}$ .

Per veure que són  $\Omega(n)$  i no  $\Theta(n)$  només cal veure que els límits següents són infinit:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/2}}{n} = \lim_{x \rightarrow \infty} (\log n)^{1/2} = \infty$$

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/3}}{n} = \lim_{x \rightarrow \infty} (\log n)^{1/3} = \infty$$

Per veure que són  $O(n \log n)$  i no  $\Theta(n \log n)$  només cal veure que els límits següents són zero:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/2}}{n \log n} = \lim_{x \rightarrow \infty} \frac{(\log n)^{1/2}}{\log n} = \lim_{x \rightarrow \infty} \frac{1}{(\log n)^{1/2}} = 0$$

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/3}}{n \log n} = \lim_{x \rightarrow \infty} \frac{(\log n)^{1/3}}{\log n} = \lim_{x \rightarrow \infty} \frac{1}{(\log n)^{2/3}} = 0$$

Finalment per veure que  $f \notin \Theta(g)$ , vegem que el límit següent no és una constant major estricta que zero:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/3}}{n(\log n)^{1/2}} = \lim_{x \rightarrow \infty} \frac{(\log n)^{1/3}}{(\log n)^{1/2}} = \lim_{x \rightarrow \infty} \frac{1}{(\log n)^{1/6}} = 0$$

### Proposta de solució al problema 2

- (a) La idea d'aquest algorisme és que, cada vegada que trobem un natural es compten les aparicions posteriors d'aquest en el vector i es marquen amb un  $-1$  per a no considerar-les més en el futur. Per tant, quan visitem un element marcat amb un  $-1$  ens estalviem el bucle més intern.

Si construïm un vector on tots els nombres són diferents, aleshores aquesta optimització no serveix per a res. A més, si tots els nombres són diferents no tenim cap element dominant (a no ser que  $n = 1$ ) i els dos bucles s'executen el màxim nombre de vegades. El cos del bucle més intern és clarament constant, pel que només hem de comptar quantes vegades s'executa. Donat una  $i$  concreta, el bucle intern s'executa  $n - i$  vegades. Com que  $i$  va des de 0 fins a  $n - 1$ , el cost total és  $n + (n - 1) + (n - 2) + \dots + 1 = \Theta(n^2)$ .

El cost en cas millor es dona, per exemple, quan tenim un vector amb un únic element repetit  $n$  vegades. En aquest cas, quan  $i = 0$  visitarem tots els elements del vector marcant-los amb un  $-1$ . Per a totes les altres  $i$ , el bucle més intern no s'executarà. Per tant, el cost en cas millor és  $\Theta(n)$ .

Si ens asseguren que tenim com a molt 100 naturals diferents, aleshores el bucle intern s'executarà com a molt 100 vegades. És a dir, hi haurà com a molt 100  $i$ s per les quals el bucle intern s'executarà. Aquestes  $i$ s contribuiran en el cas pitjor un cost de  $\Theta(100n) = \Theta(n)$ . Per la resta de les  $i$ s (en tenim com a màxim  $n$ ) el bucle intern no s'executarà i per tant, contribuiran amb un cost de  $\Theta(n)$ . Així doncs, el cost en cas pitjor ha canviat i ha passat a ser  $\Theta(n)$ .

- (b) Per aquest exercici primer recordem que la ordenació per inserció té cost  $\Theta(n^2)$  en cas pitjor i  $\Theta(n)$  en cas millor. Pel que fa al *quicksort*, tenim un cost de  $\Theta(n^2)$  en cas pitjor i  $\Theta(n \log n)$  en cas millor. Per analitzar el cost de *dominant\_sort*, oblidem-nos de moment de la crida a *own\_sort*. La resta del bucle veiem que com a màxim visita cada element del vector una vegada, fent-hi un treball constant. Cal remarcar que a vegades no visita tots els elements, ja que s'atura quan detecta l'element dominant. El cas pitjor el tenim quan visita tots els elements i no troba cap dominant (això triga  $\Theta(n)$ ). El cas millor es dona quan el primer element que visita és el dominant, però podem observar que per a detectar que és dominant ha de visitar almenys  $n/2$  elements, pel que el cost també és  $\Theta(n)$ . Per tant, el codi sempre triga  $\Theta(n)$  (obviant la crida a *own\_sort*).

Si *own\_sort* és una ordenació per inserció, el cost en cas millor és  $\Theta(n) + \Theta(n) = \Theta(n)$ , i en cas pitjor és  $\Theta(n) + \Theta(n^2) = \Theta(n^2)$ .

Si *own\_sort* és un *quicksort*, el cost en cas millor és  $\Theta(n) + \Theta(n \log n) = \Theta(n \log n)$ , i en cas pitjor és  $\Theta(n) + \Theta(n^2) = \Theta(n^2)$ .

- (c) El codi complet és:

```
int dominant_divide (const vector<int>& v, int l, int r) {
    if (l == r) return v[l];
    int n_elems = (r-l+1);
    int m = (l+r)/2;
    int maj_left = dominant_divide(v, l, m);
    if (maj_left != -1 and times(v, l, r, maj_left) > n_elems/2) return maj_left;
    int maj_right = dominant_divide(v, m+1, r);
    if (maj_right != -1 and times(v, l, r, maj_right) > n_elems/2) return maj_right;
    return -1; }
```

Per analitzar el seu cost ens adonem que en el cas pitjor es fan dues crides recursives de tamany la meitat i dues crides a *times*. La resta del codi té cost constant. Observem ara que la funció *times* rep *v* per referència i per tant el seu cost es pot descriure per  $T(n) = T(n - 1) + \Theta(1)$ , que té solució  $T(n) \in \Theta(n)$ . Així doncs, la recurrència que descriu els cost en cas pitjor d'aquest programa és

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

que té solució  $T(n) \in \Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

15/04/2021

## Proposta de solució al problema 1

(a) Calculem primer el límit

$$\lim_{x \rightarrow \infty} \frac{2^{2n}}{2^n} = \lim_{x \rightarrow \infty} 2^n = \infty$$

.

Per tant, l'única afirmació certa és que  $2^{2n} \in \Omega(2^n)$ .

A continuació, calculem el límit

$$\lim_{x \rightarrow \infty} \frac{\log(2n)}{\log(n)} = \lim_{x \rightarrow \infty} \frac{\log(2) + \log(n)}{\log(n)} = \lim_{x \rightarrow \infty} \frac{\log(2)}{\log(n)} + \lim_{x \rightarrow \infty} \frac{\log(n)}{\log(n)} = 0 + 1 = 1$$

.

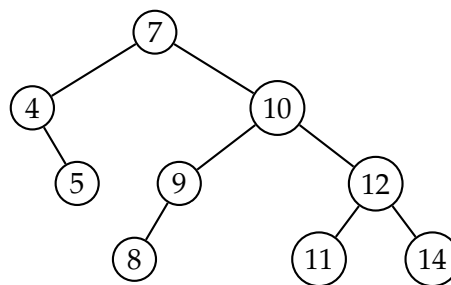
Així doncs,  $\log(2n) \in \Theta(\log(n))$ . Per tant també  $\log(2n) \in O(\log(n))$  i  $\log(2n) \in \Omega(\log(n))$ .

(b) El cos de cada iteració del bucle és  $\Theta(1)$  i per tant només cal comptar quantes iteracions es fan. Si  $y_t$  denota el valor de  $y$  al final de la  $t$ -èsima iteració, el que busquem és el mínim  $t \geq 0$  tal que  $y_t > n$ . Sabem que  $y_t = 1 + 2 + 3 + 4 + \dots + (t + 1) = \Theta(t^2)$  i per tant el mínim  $t$  és  $\Theta(\sqrt{n})$ .

(c) El codi llegeix dues seqüències d' $n$  enters i calcula si les dues seqüències tenen intersecció no buida.

El cas pitjor es dona quan tots els elements de la primera seqüència són diferents. Això farà que hi hagi  $n$  elements al *map*. Sabem que en C++ buscar un element en un *map* té cost logarítmic en el nombre d'elements. Per tant, en el cas pitjor es donen  $n$  voltes al bucle, i cada volta té cost logarítmic, pel que el cost total en cas pitjor és  $\Theta(n \log n)$ .

(d) L'arbre AVL resultant és:



## Proposta de solució al problema 2

(a) Ho demostrarem per inducció sobre  $n$ .

*Cas base:* ( $n = 0$ ). En aquest cas, la graella té mida  $1 \times 1$  i per tant només té una casella, que ha de ser necessàriament la casella bloquejada. Així doncs, no hi ha caselles restants per omplir i el resultat es compleix trivialment.

*Pas d'induccció:* sigui  $n > 0$  i assumim que el resultat és cert per graelles de mida  $2^{n-1} \times 2^{n-1}$ . Aleshores podem partir la graella en 4 parts iguals, que seran subgraelles de mida  $2^{n-1} \times 2^{n-1}$ . Si, tal com es mostra a la figura de l'enunciat, la casella bloquejada cau a la subgraella de baix a l'esquerra, aleshores podem col·locar una peça a la part central de manera que bloqueja exactament una casella a les altres 3 subgraelles. Si la casella bloquejada cau a una altra subgraella, sempre podrem escollir una peça que bloquegi exactament una casella a les altres 3 subgraelles. Per tant, després d'haver col·locat aquest peça central sempre tindrem 4 subgraelles de mida  $2^{n-1} \times 2^{n-1}$  amb exactament una casella bloquejada a cadascuna d'elles. Gràcies a la hipòtesi d'inducció sabem que totes elles es poden omplir amb les peces disponibles, i per tant la graella original de mida  $2^n \times 2^n$  també.

(b)

```

int quadrant(Coord pos, int i_l, int i_r, int j_l, int j_r) {
    int size = j_r - j_l + 1;
    int i_m = i_l + size / 2;
    int j_m = j_l + size / 2;
    if (pos.first < i_m and pos.second < j_m) return 0;
    if (pos.first < i_m) return 1;
    if (pos.second < j_m) return 2;
    return 3;
}

void fill (vector<vector<int>>& M, int i_l, int i_r, int j_l, int j_r,
           Coord c_blocked, int& num){
    if (i_l == i_r) return; // 1x1
    int size = j_r - j_l + 1;
    int i_m = i_l + size / 2; // Midpoints
    int j_m = j_l + size / 2;

    vector<Coord> coords_blocked(4); // Blocked cell in each quadrant
    coords_blocked [0] = {i_m - 1, j_m - 1};
    coords_blocked [1] = {i_m - 1, j_m};
    coords_blocked [2] = {i_m, j_m - 1};
    coords_blocked [3] = {i_m, j_m};

    int q = quadrant(c_blocked, i_l, i_r, j_l, j_r);
    coords_blocked [q] = c_blocked;

    for (int k = 0; k < 4; ++k)
        if (M[coords_blocked[k].first][coords_blocked[k].second] == -1)
            M[coords_blocked[k].first][coords_blocked[k].second] = num;

    ++num;

    fill (M, i_l, i_m - 1, j_l, j_m - 1, coords_blocked [0], num); // Q0
    fill (M, i_l, i_m - 1, j_m, j_r,      coords_blocked [1], num); // Q1
    fill (M, i_m, i_r,      j_l, j_m - 1, coords_blocked [2], num); // Q2
    fill (M, i_m, i_r,      j_m, j_r,      coords_blocked [3], num); // Q3
}

```

- (c) Per analitzar el cost del codi, ens hem de fixar en la funció *fill*. Fixem-nos que el seu codi essencialment no té bucles. Només n'hi ha un, però sempre dóna 4 voltes (independentment de la mida del problema). Per tant, fa un nombre constant de voltes. Com que totes les altres instruccions triguen temps  $\Theta(1)$ , si no consideréssim les crides recursives la funció trigaria temps constant. No obstant, per un problema de mida  $2^n \times 2^n$  es fan 4 crides recursives amb mida  $2^{n-1} \times 2^{n-1}$ . Per tant, el cost en funció de  $n$  ve donat per la recurrència:

$$T(n) = 4T(n-1) + \Theta(1)$$

que té solució  $T(n) \in \Theta(4^n)$ .

Com que el nombre de caselles és  $2^n \cdot 2^n = 4^n$ , podem afirmar que el codi és lineal en el nombre de caselles.

## Solució de l'Examen Parcial EDA

08/11/2021

## Proposta de solució al problema 1

- (a) La funció *mystery* determina si  $n$  és un nombre primer.

Pel que fa al seu cost, fixem-nos que les tres primeres línies del codi tenen cost constant perquè només fan operacions aritmètiques i comparacions entre enters. La part interessant del codi és el bucle. En el cas pitjor es fan totes les voltes al bucle. Per facilitar el raonament podem assumir que el bucle comença a  $i = 1$  (afegir dues voltes al bucle no canvia el cost asimptòtic). Com que parem quan  $i^2 > n$ , és a dir  $i > \sqrt{n}$ , es farien  $\sqrt{n}$  voltes si  $i$  s'incrementés en una unitat a cada volta. Com que s'incrementa en dos, se'n fan la meitat:  $\sqrt{n}/2$  voltes. Si considerem que cada volta només fa un treball constant (assignacions, operacions aritmètiques i comparacions entre enters), el cost total en cas pitjor és  $\Theta(\sqrt{n})$ .

- (b) El seu cost és  $\Theta(n\sqrt{n}) = \Theta(n^{3/2})$ .

Les dues primeres línies tenen cost constant. Anem a estudiar el bucle. El primer fet a observar és que el bucle més intern (el que varia  $k$ ) té cost  $\Theta(n)$ . El bucle més extern fa  $n$  voltes. Per algunes d'elles s'executarà el bucle intern i per les altres es farà un treball constant. Anem a comptar quantes vegades s'executa el bucle intern. Aquest només s'executa quan  $i = j^2$ . La variable  $j$  inicialment val zero, i cada vegada que s'executa el bucle intern s'incrementa en una unitat. Ens podem fixar que el bucle intern s'executarà per primera vegada quan  $i = j^2 = 0$ , i llavors  $j$  passarà a valer 1. La segona vegada serà quan  $i = j^2 = 1$  i llavors  $j$  passarà a valer 2. La tercera vegada serà quan  $i = j^2 = 4$ , i així successivament. Per tant, el bucle intern s'executarà tantes vegades com quadrats hi hagi entre 0 i  $n - 1$ , és a dir,  $\lceil \sqrt{n} \rceil$ . La resta de vegades, és a dir,  $(n - \lceil \sqrt{n} \rceil)$  vegades, el bucle intern no s'executa. Així doncs el cost total és  $(n - \lceil \sqrt{n} \rceil) \cdot \Theta(1) + \lceil \sqrt{n} \rceil \cdot \Theta(n) = \Theta(n \cdot \sqrt{n})$ .

- (c) Ens fixem que *pairs* és una funció recursiva on a cada crida recursiva decreix el nombre d'elements en  $v[l \dots r]$ . Inicialment aquest nombre és  $n$ . Si ens centrem en el cas recursiu, podem veure que hi ha dues crides recursives on el nombre d'elements és la meitat, una sèrie d'instruccions de cost constant i dos bucles enniestrats. El bucle extern tracta la part esquerra del vector, entre  $l$  i  $m$ , on hi ha essencialment  $n/2$  elements. Per cadascun d'aquests elements, el bucle intern recorre la part dreta del vector, entre  $m + 1$  i  $r$ , on també tenim aproximadament  $n/2$  elements. Tot plegat, el condicional de dins el bucle s'executa bàsicament  $n^2/4$  vegades, que equival a un treball  $\Theta(n^2)$ . Per tant, la recurrència que descriu el cost d'aquesta funció és

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

que té solució  $T(n) \in \Theta(n^2)$ .

- (d) Com que  $K$  no depèn d' $n$ , la primera línia del codi té cost constant, així com la segona. El primer bucle té cost  $\Theta(n)$ . Només ens falta analitzar el segon bucle, que repeteix  $K$  vegades un treball constant. Com que  $K$  també és constant, el bucle triga  $\Theta(1)$ . Tot plegat el cost de la funció és  $\Theta(n)$ .

La correcció del codi es basa en la següent observació. El primer bucle calcula, per cada nombre  $k$  entre 0 i  $K - 1$ , quantes vegades apareix  $k$  a  $v$ . Aquest valor es guarda a *times*[ $k$ ]. Si un nombre  $k$  apareix  $p$  vegades, aleshores hi haurà exactament  $p \cdot (p - 1)/2$

parelles  $(i, j)$  amb  $0 \leq i < j < n$  tals que  $v[i] = v[j] = k$ . Com que suma sobre totes les  $k$  possibles, el segon bucle ens calcula la quantitat desitjada.

### Proposta de solució al problema 2

```
(a)  int first_occurrence (int x, const vector<int>& v, int l, int r) {
        if (l > r) return -1;
        else {
            int m = (l+r)/2;
            if (v[m] < x) return first_occurrence (x, v, m+1, r);
            else if (v[m] > x) return first_occurrence (x, v, l, m-1);
            else if (m == l or v[m-1] != x) return m;
            else return first_occurrence (x, v, l, m-1);
        }
    }
```

(b) El codi a omplir és  $res = n - q - p$ ;

Per analitzar el seu cost, veiem que el codi comença amb una crida a *first\_occurrence* de cost, en cas pitjor,  $O(\log n)$ . A continuació la propera part interessant és el primer bucle, de cost  $\Theta(n)$ . El segon bucle, tot i que només visita la meitat dels elements de  $v$ , té el mateix cost  $\Theta(n)$ , perquè sabem que un *swap* triga temps  $\Theta(1)$ . Finalment, tenim una altra crida a *first\_occurrence*, altra vegada de cost, en cas pitjor,  $O(\log n)$ . Tot plegat el cost del codi és  $\Theta(n)$ .

Per tal d'entendre per què el codi és correcte, hem d'entendre que essencialment aquest intenta calcular la primera i la darrera aparició d' $x$  a  $v$ , i calcular quants elements hi ha entre aquestes dues posicions. El punt clau és adonar-se que si invertim el vector i neguem tots els seus elements, obtenim un vector ordenat creixentment tal que l'última aparició d' $x$  en el  $v$  original coincideix amb la primera aparició de  $-x$  en el nou  $v$ . A més, si la primera aparició de  $-x$  en el nou vector  $v$  és a la posició  $q$ , aleshores la darrera aparició d' $x$  en el vector  $v$  original és  $n - 1 - q$ . Per tant, el nombre d'elements entre  $p$  (primera aparició) i  $n - 1 - q$  (última aparició) és  $(n - 1 - q) - p + 1 = n - q - p$ .

(c) Podem aconseguir fàcilment un algorisme de cost, en cas pitjor,  $O(\log n)$  si calculem millor la darrera aparició d' $x$  a  $v$ . Per tal de fer-ho, podem modificar el codi de *first\_occurrence* lleugerament:

```
int last_occurrence (int x, const vector<int>& v, int l, int r) {
    if (l > r) return -1;
    else {
        int m = (l+r)/2;
        if (v[m] < x) return last_occurrence (x, v, m+1, r);
        else if (v[m] > x) return last_occurrence (x, v, l, m-1);
        else if (m == r or v[m+1] != x) return m;
        else return last_occurrence (x, v, m+1, r);
    }
}
```

Una crida a *last\_occurrence* té cost en cas pitjor  $O(\log n)$ . Aleshores, només ens caldrà trobar la primera i la darrera aparició d' $x$ , diguem-ne  $p$  i  $q$ , i retornar  $q - p + 1$  o bé 0 si no hi ha cap aparició.



## Solució de l'Examen Parcial EDA

31/03/2022

## Proposta de solució al problema 1

- (a) El bucle que inicialitza el vector  $v$  triga temps  $\Theta(n)$ . El mateix passa amb la crida a *random\_shuffle*.

Pel que fa als bucles ennierats, fixem-nos primer que dins el vector  $v$  hi posem tots els nombres entre 1 i  $n$ , i a continuació els reordenem de manera aleatòria. Si no els haguessim ordenat, per cada valor de  $i$ , el bucle més intern faria  $v[i] = i + 1$  voltes (i per tant incrementaria  $s$  en una unitat  $i + 1$  vegades). Com que  $i$  es mou des de 0 fins a  $n - 1$ , el nombre de voltes totals del bucle intern seria  $1 + 2 + 3 + \dots + n = n(n + 1)/2$ . Així doncs, podem afirmar que el codi sense l'ordenació calcula  $n(n + 1)/2$  i té cost  $\Theta(n(n + 1)/2) = \Theta(n^2)$ .

L'única cosa que canvia degut a l'ordenació de  $v$  és que deixa ser cert que per  $i = 0$  el bucle intern faci 1 volta, que per  $i = 1$  en faci 2, que per  $i = 2$  en faci 3, etc. En el nostre codi, per cada valor de  $i$  el bucle intern farà  $v[i]$  voltes. Però com que  $v$  és una permutació de  $\{1, 2, \dots, n\}$ , hi haurà una iteració on es farà 1 volta, una altra on se'n faran 2, una altra on se'n faran 3, etc. Per tant, el codi calcula el mateix amb el mateix cost asimptòtic.

- (b) El seu cost és  $\Theta(n \log \log n)$ . Anem a veure per què.

El bucle extern dona  $n$  voltes, i el cost del bucle intern és independent del valor de  $j$ . Per tant, el cost total serà  $n$  vegades el cost del bucle intern.

Pel que fa al bucle intern, aquest s'atura quan  $k \geq n$ . En entrar a la primera iteració  $k$  val 2, a la segona val 4, a la tercera val 16, a la quarta val 256, etc. Podem afirmar que a la iteració  $i$ -èsima  $k$  val  $2^{2^i}$ . Es donaran voltes, per tant, mentre  $2^{2^i} < n$ , el que equival a que  $2^i < \log n$ , i que  $i < \log \log n$ . Per tant, el bucle intern fa  $\Theta(\log \log n)$  iteracions i el cost total del codi és  $\Theta(n \log \log n)$ .

- (c) Per analitzar el cost de l'algorisme d'inserció, podem comptar el nombre d'intercanvis que s'han de fer. Sabem que el nombre d'intercanvis que es faran quan s'hagi de col·locar un nombre a la posició que li pertoca es correspon al nombre d'elements a la seva esquerra que són estrictament majors que ell en la configuració inicial.

Els dos primers nombres no tenen cap element major a la seva esquerra. Pel 2 i pel  $2n - 1$  en tenim 1 per a cadascun. Pel 3 i pel  $2n - 2$  en tenim 2 per a cadascun. Pel 4 i pel  $2n - 3$  en tenim 3 per a cadascun, i així successivament, fins a la parella  $n, n + 1$  pels que en tenim  $n - 1$  per cadascun. Així doncs, el nombre d'intercanvis serà  $1 + 1 + 2 + 2 + 3 + 3 + \dots + (n - 1) + (n - 1) = 2 \cdot n(n - 1)/2 = n(n - 1)$ , que és  $\Theta(n^2)$ . Per tant, aquest és el cost de l'algorisme d'ordenació per inserció per aquesta entrada.

## Proposta de solució al problema 2

- (a) En el cas pitjor, realitzarem totes les iteracions ( $n$ ) al bucle que hi ha dins *inefficient* i acabarem retornant  $n$ . A cada bucle es crida a la funció *find*. Com que es passa el vector per referència, el pas de paràmetres és  $\Theta(1)$ . Totes les altres operacions dins *inefficient* tenen cost  $\Theta(1)$ , pel que només ens hem de centrar en les crides a *find*.

Podem veure que *find* és una funció recursiva, on el que decreix és el valor de *pos*, que inicialment és  $n - 1$ . Com que totes les operacions són constants, el seu cost ve descrit per la recurrència  $T(n) = T(n - 1) + \Theta(1)$ , que té solució  $T(n) \in \Theta(n)$ .

Per tant, com que cada crida a *find* té cost  $\Theta(n)$  i es fan  $n$  crides, el cost total és  $\Theta(n^2)$ .

(b) Una possible solució és:

```
int efficient (const vector<int>& v, int l, int r) {  
    if (l > r) return v.size ();  
    int m = (l+r)/2;  
    if (v[m] > m) {  
        if (m == l or v[m-1] == m-1) return m;  
        else return efficient (v, l, m-1);  
    }  
    else return efficient (v, m+1, r); // we know v[m] == m  
}
```

## Solució de l'Examen Parcial EDA

03/11/2022

## Proposta de solució al problema 1

- (a) El programa anterior escriu el valor
- $N^N$
- per pantalla.

Només cal demostrar que  $f(x, n)$  retorna  $x^n$  per  $n \geq 1$ . Es pot demostrar fàcilment per inducció. Per  $n = 1$ , és obvi que retorna  $x = x^1$  gràcies a la primera línia de la funció. Per  $n > 1$ , si assumim per hipòtesi d'inducció que  $f(x, n - 1) = x^{n-1}$ , aleshores la variable *tmp* conté  $x^{n-1}$ . El bucle suma  $x$  vegades el valor de *tmp*, és a dir  $x^{n-1}$ , i el guarda dins *res*. Per tant *res* conté  $x \cdot x^{n-1} = x^n$ .

Pel que fa al cost, ens n'adonem que la part recursiva de la funció fa tot d'operacions constants més (1) una crida recursiva de mida  $n - 1$ , i (2) un bucle de cost  $\Theta(x)$ . Seria incorrecte dir que el cost del bucle és  $\Theta(n)$  perquè en les successives crides a la funció no es compleix que  $x = n$ , sinó que  $x$  es manté constant i  $n$  va decreixent.

Per a solucionar-ho, calcularem d'una banda el cost de la funció sense considerar el bucle i d'una altra banda el cost de totes les execucions del bucle. Si no considerem el bucle, el cost de la funció ve donat per  $C(n) = C(n - 1) + \Theta(1)$ , que té solució  $C(n) \in \Theta(n)$ . Pel que fa al bucle, tenim en compte que per una crida inicial  $f(N, N)$  amb  $N > 1$  es faran  $N - 1$  execucions del bucle, cadascuna d'elles amb cost  $\Theta(N)$ . Tot plegat, el bucle ens dona un cost  $\Theta((N - 1)N) = \Theta(N^2)$ . Per tant, el cost del main és  $\Theta(N) + \Theta(N^2) = \Theta(N^2)$ .

- (b) Calculem el límit:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\ln(\ln(n^2))}{\ln(\ln n)} = \lim_{n \rightarrow \infty} \frac{\ln(2 \ln n)}{\ln(\ln n)} = \lim_{n \rightarrow \infty} \frac{\ln(2) + \ln(\ln n)}{\ln(\ln n)} =$$

$$\lim_{n \rightarrow \infty} \frac{\ln(2)}{\ln(\ln n)} + \lim_{n \rightarrow \infty} \frac{\ln(\ln n)}{\ln(\ln n)} = 0 + 1 = 1$$

Per tant, podem afirmar que  $f(n) \in \Theta(g(n))$ .

Pel que fa a  $F(n)$  i  $G(n)$ , podríem calcular també  $\lim_{n \rightarrow \infty} \frac{F(n)}{G(n)}$ . Com que quan calculem el límit quan  $n \rightarrow \infty$  no ens importen els valors que  $F$  o  $G$  puguin prendre per  $n \leq 10$ , el límit també serà 1 i per tant  $F(n) \in \Theta(G(n))$ .

## Proposta de solució al problema 2

- (a) Recordem que  $n = h.size() = s.size()$ . La funció proporcionada només fa operacions amb cost  $\Theta(1)$  (operacions aritmètiques bàsiques, accessos a vectors, càlcul de mínim i màxim) però té dos bucles ennierrats. El bucle intern té cost  $\Theta(n)$ , i aquest cos és independent de la iteració del bucle extern on estiguem. El bucle extern s'executa  $n$  vegades, i per tant el cost total és  $\Theta(n \cdot n) = \Theta(n^2)$ .

(b) El codi omplert és:

```
int radium_v2 (const vector<int>& h, const vector<int>& s) {
    int r = 0, j = 0;
    for (int i = 0; i < s.size (); ++i){
        while (j < h.size () and h[j] < s[i]) ++j;
        int rad;
        if (j == h.size ()) rad = s[i] - h[h.size ()-1];
        else if (j == 0) rad = h[0] - s[i];
        else rad = min(s[i] - h[j - 1], h[j] - s[i]);
        r = max(r, rad);
    }
    return r;
}
```

Pel que fa al cost, ens fixem que la funció fa tot d'operacions de cost  $\Theta(1)$ , però té dos bucles ennyerats. El que hem de tenir en compte és que el valor de  $j$ , que controla el bucle intern, no s'inicialitza en cada volta del bucle extern, sinó que  $j$  val 0 a l'inici de la funció i es va incrementant al llarg de tota l'execució, fins a valer com a molt  $n$ . Per tant, durant tota l'execució de la funció es fan com a molt  $n$  voltes al bucle **while** i, per tant, té cost  $\Theta(n)$ . Remarquem que aquest cost no és per a cada volta del bucle extern, sinó que ja considera totes les voltes. Finalment només ens queda analitzar el bucle extern, que s'executa  $n$  vegades i, per tant, té cost  $\Theta(n)$  si obviem el cost del **while**, que ja l'hem comptat a part. Així doncs, el cost total és  $\Theta(n + n) = \Theta(n)$ .

(c) Una possible solució és:

```
int find (const vector<int>& h, int l, int r, int p) {
    if (l > r) return h.size ();
    else {
        int m = (l+r)/2;
        if (h[m] < p) return find(h, m+1, r, p);
        else if (m > l and h[m-1] ≥ p) return find(h, l, m-1, p);
        else return m;
    }
}
```

Per analitzar el seu cost ens n'adonem que totes les operacions tenen cost  $\Theta(1)$ , excepte la crida recursiva que sempre té mida la meitat de la mida original. Per tant, el cost de la funció ve donat per la recurrència  $C(n) = C(n/2) + \Theta(1)$ , que té solució  $C(n) = \Theta(\log n)$ .

(d) La línia que acabem d'introduir sempre té cost en cas pitjor  $\Theta(\log n)$ . Per tant, com que el bucle **for** dona  $n$  voltes, i a cada volta fa un treball  $\Theta(\log n)$  més altres operacions de temps  $\Theta(1)$ , tenim que el cost total és  $\Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

21/04/2023

## Proposta de solució al problema 1

- (a) El cas pitjor es dona quan s'efectuen totes les iteracions al bucle més extern sense trobar una solució. Centrem-nos, doncs, en aquest cas.

Sabem que el bucle extern farà  $n$  voltes. Per cadascuna d'aquestes voltes, el bucle intern en farà  $n$ . Per tant, el cos de bucle més intern s'executarà exactament  $n^2$  vegades.

Aquest cos executarà dues instruccions *swap*, una comparació entre enters i dues crides a *suma*. Si no fos per aquestes dues crides, el cos tindria cost  $\Theta(1)$ . Una crida a *suma* involucra passar un vector per referència, inicialitzar la variable  $s$  a zero, retornar un enter (tot plegat cost  $\Theta(1)$ ), així com un bucle que itera  $n$  vegades acumulant el valor dels elements de  $v$  a la variable  $s$ . Així doncs, el cost d'una crida a *suma* és  $\Theta(n)$ . Resumint, el cost del cos del bucle més intern és  $\Theta(1) + 2\Theta(n) = \Theta(n)$ .

Per tant, com que aquest s'executa  $n^2$  vegades, tenim un cost total de  $n^2 \cdot \Theta(n) = \Theta(n^3)$ .

- (b) La part del codi a completar és:

```
int x = v1[i] + dif/2;
```

Passem a analitzar el cost en cas pitjor d'una crida a *sol*. Com abans, el cas pitjor tindrà lloc quan executem totes les iteracions del bucle més extern.

La primera línia de *sol* efectua dues crides a *suma* i una resta, pel que té cost  $\Theta(n)$ . La segona línia té cost  $\Theta(1)$ , doncs només es fan operacions aritmètiques bàsiques i una comparació amb zero. En el cas pitjor, el bucle farà  $n$  voltes, i en cadascuna d'elles s'efectuarà una crida a *cerca* i altres operacions  $\Theta(1)$ . Així doncs, ja podem concloure que el cost serà  $n$  vegades el cost d'una crida a *cerca*.

La funció *cerca* és una funció recursiva que, en cas pitjor, efectua tot d'operacions aritmètiques de cost  $\Theta(1)$  i dues crides recursives. Com que  $m$  és el punt mig del vector, ens n'adonem que les crides recursives són de mida la meitat. Així doncs, la recurrència que descriu el cost d'aquesta funció és  $C(n) = 2 \cdot C(n/2) + \Theta(1)$ . Si apliquem el teorema mestre per a recurrències divisores del tipus  $C(n) = a \cdot C(n/b) + \Theta(n^k)$ , podem identificar que  $a = b = 2$ ,  $k = 0$  i, per tant  $\alpha = \log_2 2 = 1$ . Com que  $\alpha > k$ , tenim que la recurrència té solució  $C(n) \in \Theta(n^\alpha) = \Theta(n)$ .

Per tant, el cost total serà  $n \cdot \Theta(n) = \Theta(n^2)$ .

- (c) Per tal de millorar el cost, farem que la crida a *cerca* sigui més eficient. En concret, a la funció *sol*, just abans de l'inici del bucle, ordenarem el vector  $v_2$  amb un algorisme d'ordenació que ens garanteixi cas pitjor  $n \log n$ , com pot ser el *mergesort*. Una vegada ordenat, enlloc de la funció *cerca* podem utilitzar una cerca dicotòmica, que tindrà cost  $\Theta(\log n)$ .

Si ho comparem amb l'anàlisi de cost anterior, veiem que haurem d'afegir el cost de l'ordenació, i reemplaçar el cost de *cerca* pel cost de la cerca dicotòmica. Per tant, el cost total serà  $\Theta(n \log n) + n \cdot \Theta(\log n) = \Theta(n \log n)$ .

**Proposta de solució al problema 2**

(a)  $n = 10, k = 2 \Rightarrow 10 = 2^3 + 2^1$

$n = 10, k = 3 \Rightarrow 10 = 2^2 + 2^2 + 2^1$

$n = 10, k = 4 \Rightarrow 10 = 2^2 + 2^1 + 2^1 + 2^1$

$n = 10, k = 5 \Rightarrow 10 = 2^1 + 2^1 + 2^1 + 2^1 + 2^1$

- (b) Com que sabem que la representació en binari d' $n$  és la representació com a potències de 2 amb el menor nombre de sumands, si  $k$  és menor que el nombre d'uns de la representació en binari no hi ha solució possible.

D'altra banda, és fàcil adonar-se que la representació d' $n$  en potències de dos amb el major nombre de sumands és la suma d' $n$  sumands  $2^0$ . Així doncs, si  $k > n$  tampoc hi haurà solució.

- (c) Una possible solució és:

```
vector<int> pos_uns (int n) {
    vector<int> v;
    int pos = 0;
    while (n != 0) {
        if (n%2 == 1) v.push_back(pos);
        ++pos;
        n = n/2;
    }
    return v;
}
```

Veiem que totes les operacions que fa la funció tenen cost  $\Theta(1)$  (assumint que el *push\_back* té cost  $\Theta(1)$ ). Així doncs, el cost vindrà donat pel nombre de voltes que faci el bucle. Si  $n_0$  és el valor inicial de la variable  $n$ , en acabar la primera iteració,  $n$  val com a molt  $n_0/2$  (recordem que la divisió entera en C++ trunca cap a baix). En acabar la segona, val com a molt  $n_0/2^2$ , en acabar la tercera com a molt  $n_0/2^3$  i, en general després de la iteració  $k$ -èsima valdrà com a molt  $n_0/2^k$ . Per tant, podem garantir que quan  $n_0 < 2^k$  el bucle s'aturarà perquè  $n$  valdrà zero. Això passa quan  $\log n_0 < k$ , i per tant, podem garantir que el bucle donarà com a molt  $\Theta(\log n_0)$  voltes. Com que  $n_0$  era el valor inicial d' $n$ , podem concloure que el cost és  $\Theta(\log n)$ .

- (d) Una possible solució és:

```
void escriu_suma_potencies (int n, int k) {
    vector<int> uns = pos_uns(n);
    if (uns.size() > k or k > n)
        cout << "No hi ha solucio" << endl;
    else {
        priority_queue<int> Q;
        for (auto x : uns) Q.push(x);
        while (Q.size() < k) {
            int m = Q.top();
            Q.pop();
            Q.push(m-1);
        }
    }
}
```

```
        Q.push(m-1);
    }
    bool primer = true;
    while (not Q.empty()){
        if (not primer) cout << " + ";
        else primer = false;
        cout << "2^" << Q.top();
        Q.pop();
    }
    cout << endl;
}
}
```