

Problema 1: Calcular el coste de Fibonacci:
$$F(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ F_{n-1} + F_{n-2} & \text{si } n \geq 2 \end{cases}$$

```
int fib (int n) {
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fib(n-1) + fib(n-2);
}
```

~~se saben que $F(n+1) \leq F(n)$ y $F(n) \leq F(n+1)$, así $F(n+1) \leq F(n+1) + F(n) \leq 2F(n+1)$~~

1) Sabem que podem calcular els nombres de Fibonacci amb matrius tal que $\begin{pmatrix} F(n+1) \\ F(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F(1) \\ F(0) \end{pmatrix}$ per tot $n \geq 0$. Per tant calculem M^n (on $M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$) amb l'algorisme d'exponenciació ràpida que utilitz la recurrència $T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \Theta(1)$ per tant tenim $\Theta(\log n)$

2) $F(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ F(n-1) + F(n-2) & n \geq 2 \end{cases}$ * no podem resoldre la recurrència amb teoremes mestres però podem observar el següent $F(n+1) \leq F(n) \leq F(n+1)$
 $\Rightarrow 2F(n-2) \leq F(n) \leq 2F(n-1) \Rightarrow F(n) = O(2^n) \wedge F(n) = \Omega(2^{\frac{n}{2}})$

Parcial 06/11/2017

Problema 4: (2,5 puntos)

Considereu la següent recurrència:

$$\Theta(\log n)$$

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

~~b = 2~~ k =
~~a = 1~~

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

$$\begin{aligned} U(m) &= T(2^m) = T(2^m/2) + \log(2^m) = T(2^m/2) + \log(2^m) = \\ &= T(2^{m-1}) + m = U(m-1) + m \end{aligned}$$

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

$$U(m) = U(m-1) + m \quad a = 1, b = 1, k = 1$$

$$\Theta(m^{k+1}) = \Theta(m^2)$$

c) (1 pt.) Resoleu asymptòticament la recurrència per a T.

$$U(m) = T(2^m) \rightarrow T(n) = T(2^{\log n}) = U(\log n) = \Theta((\log n)^2)$$

$n = 2^m$
 $\log n = m$

Parcial 11/11/2019

Problema 1: (1 punto)

$$\sqrt{n} = n^{\frac{1}{2}}$$

- (a) (0.5 pts.) La solució de la recurrència $T(n) = 2T(n/4) + \Theta(\sqrt{n})$ és asimptòticament $T(n) = \Theta(n^{1/2} \cdot \log n)$.
No cal que justifiqueu la resposta.

$$T(n) = 2T(n/4) + \Theta(n^{1/2}) \Rightarrow \text{TEOREMA DE RECURRÈNCIES}$$

~~SUMATORES~~
~~PRODUCTES~~

$a=2, b=4, k=\frac{1}{2} \Rightarrow \text{calculem } \alpha = \log_b a = \log_4 2 = \frac{1}{2}$

$\Rightarrow \alpha = k = \frac{1}{2} \text{ per tant } T(n) = \Theta(n^k \log n) = \Theta(n^{1/2} \log n)$

- (b) (0.5 pts.) Per a quines $X \in \{O, \Omega, \Theta\}$ es compleix que $\log_2(n) \in X(\log_2(\log_2(n^2)))$?

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log(\log n^2)}{\log n} &= \lim_{n \rightarrow \infty} \frac{\log(2\log n)}{\log n} = \lim_{n \rightarrow \infty} \frac{(\log 2 + \log(\log n))}{\log n} = \\ &= \lim_{n \rightarrow \infty} \frac{\cancel{\log 2}^0}{\log n} + \lim_{n \rightarrow \infty} \frac{\log(\log n)}{\log n} = \lim_{n \rightarrow \infty} \frac{\log(\log n)}{\log n} \end{aligned}$$

Fem canvi de variable $m = \log n \Rightarrow \lim_{n \rightarrow \infty} \frac{\log m}{m} = 0$

Per tant es compleix únicament $\log_2(n) \in \Omega(\log(\log n^2))$

Parcial 18/10/2010

Problema 5 (turno 2): (1 puntos)

Considereu un procediment amb els principals:

```
int k = f(n); O(n) / O(n^2) / O(1)
int s = 0; O(1)
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 $n!$. for $\Theta(n!) > O(n)$ per tant $O(n!)$
2. El temps de $f(n)$ és $O(n)$ i el valor de $f(n)$ és n . for $\Theta(n)$ per tant $O(n)$
3. El temps de $f(n)$ és $O(n^2)$ i el valor de $f(n)$ és $n!$. for $\Theta(n!) > O(n^2)$ per tant $O(n!)$
4. El temps de $f(n)$ és $O(1)$ i el valor de $f(n)$ és 0. for $\Theta(0) < O(1)$ per tant $O(1)$

Parcial 07/11/2016

Problema 3: (2 puntos)

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;
}

```

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$+ 1 \log n$

$\Theta(1) +$

$$\begin{aligned}
 m &= \frac{x}{n} = 12 \\
 P &= 1, x = 10, y = 12 \quad u \\
 12! &= 0 \rightarrow 12 \% 2 \rightarrow \cancel{1000000} = x; y = 6 \\
 6! &= 0 \rightarrow 6 \% 2 \rightarrow \cancel{1000000} = x; y = 3 \\
 3! &= 0 \rightarrow y = 2; P = \cancel{100000} 16 \\
 2! &= 0 \rightarrow 2 \% 2 \rightarrow x = 256; y = 1 \\
 1! &= 0 \rightarrow y = 0 \quad P = \cancel{256} 4096 \\
 + 1 \log n &= 2^{12} \\
 \Theta(1) + 2 \log n &
 \end{aligned}$$

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

$$\begin{aligned}
 m &= 3 \quad n = 5 \\
 P &= 1; \quad x = 3; \quad y = 5 \\
 5! &= 0 \rightarrow y = 4; \quad P = 3 \\
 4! &= 0 \rightarrow 4 \% 2 \rightarrow x = 9 \quad y = 2 \\
 2! &= 0 \rightarrow 2 \% 2 \rightarrow x = 81 \quad y = 1 \\
 1! &= 0 \rightarrow y = 0; \quad P = 81 \cdot 3 = 243 \\
 &= 3^5
 \end{aligned}$$

L'algorisme mysteri(m, n) calcula donats dos enters $m, n \geq 0 \Rightarrow m^n$

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

```

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

```

* El cost de l'algorisme mystery depen de les vegades que s'executa el while. Si y es parell reduim el valor de y a la mitat. Si no reduim d'un en un, que torna a ser parell. Per tant el numero de voltas està entre $1 + \log n$ i $1 + 2\log n$ que es $\Theta(\log n)$.

Parcial 20/04/2017

Problema 1: (2 puntos)

- a) (0.5 pts.) Calculeu el cost mig de l'algorithm 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.

El cas millor es que ja estigui ordenat per tant té $\Theta(n)$.
 El cas pitjor que estigui ordenat del revés que valors tenim $\Theta(n^2)$.

Per tant el cas mitjà es el $\sum \Pr(n) \cdot T(n)$.

$$T(n)_{\text{mitg}} = \frac{\log n}{n} \cdot \Theta(n^2) + 1 - \frac{\log n}{n} \cdot \Theta(n) = \frac{n^2 \log n}{n} + n - \frac{n \log n}{n}$$

$$= n \log n + n - \log n = \Theta(\max(n \log n, n - \log n)) = \Theta(n \log n)$$

- 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;
}
```

$$\begin{aligned}
 & n=7 \\
 & i=3; i \cdot i \leq n; i+=2 \\
 & 7 \% 3 \neq 0 \\
 & \text{if } i \text{ primo} \\
 & i^2 \leq n \rightarrow i \leq \sqrt{n}
 \end{aligned}$$

Completeu: la funció *mystery* calcula si el nombre n és un nombre primer

i el seu cost és $O(\boxed{\sqrt{n}})$. No cal justificar la resposta.

Parcial 20/04/2017

Problema 1: (2 puntos)

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

$$O(g) = \{ f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c > 0, n_0 \in \mathbb{N} \text{ s.t. } \forall n \geq n_0 \quad f(n) \leq c \cdot g(n) \}$$

~~Definició d'O(g)~~

~~Definició d'Omega(g)~~

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 1$$

$$n^n = n \cdot n \cdot n \cdot n$$

$$\text{per tant } n! < n^n \Rightarrow n! \in O(n^n) \quad \forall n \geq 1 \quad n_0 = 1, c = 1$$

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

$$n! \geq 2^n \Rightarrow n! \geq 2^{n-1} \Rightarrow n! \geq \frac{1}{2} \cdot 2^n \quad \forall n \geq 1$$

$$\text{per tant } n! = \Omega(2^n) \quad n_0 = 1 \text{ i } c = \frac{1}{2}$$

Dejando el 1, tots els nombres que es multipliquen a $n!$ són majors o iguals a 2.

$$n! = \underbrace{n \cdot (n-1) \cdot (n-2) \cdots 1}_{n \text{ veces}} \stackrel{\geq 2}{\geq} \Rightarrow \text{per tant } n! \geq 2^{n-1} \Rightarrow n! \geq \frac{2^n}{2}$$

$$\Rightarrow n! \geq \frac{1}{2} \cdot 2^n$$

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

$i = 0$
 $x = 2 \quad y = 1$
 $y = 3 = \textcircled{2} + 1$
 $x = 3$
 $i = 1$
 $x = 3 \quad y = 3$
 $y = 3 + \textcircled{3}$
 $x = x + 1$

$$\begin{aligned}
 & 1 + 2 + 3 + \dots + n \\
 & = \frac{n \cdot (n+1)}{2} \rightarrow y^2 \leq n \\
 & \sqrt{n} \text{ iteracions} \Rightarrow \Theta(\sqrt{n})
 \end{aligned}$$

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

$i = 0$
 $x = 2 \quad y = 1$
 $y = \textcircled{1} + \textcircled{2}$
 $x = 4$
 $i = 1$
 $x = 4 \quad y = 3$
 $y = 3 + \textcircled{4}$
 $x = 8$

$$\begin{aligned}
 & i = 1 \\
 & x = 8 \quad y = 7 \\
 & y = 7 + \textcircled{8} \\
 & x = 16 \\
 & 1 + 2 + 4 + 16 + \dots \\
 & 2^0 + 2^1 + 2^2 + \dots + 2^i \Rightarrow 2^i \leq n \\
 & i \leq \log n \text{ iteracions} \Rightarrow \Theta(\log n)
 \end{aligned}$$

```
void f3(int n) {
    int x = 2;
    int y = 1;
    while (y <= n) {
        y = y * x;
        x = 2x;
    }
}
```

$i = 0$
 $x = 2 \quad y = 1$
 $y = 2$
 $x = 4$
 $i = 1$
 $x = 4 \quad y = 2$
 $y = 2 \cdot 4$
 $x = 8$

$$\begin{aligned}
 & 1 \cdot 2 \cdot 4 \cdot 8 \cdot 16 \\
 & 2^0 \cdot 2^1 \cdot 2^2 \cdot \dots \cdot 2^i \\
 & = 2^{\sum_{i=0}^i} = 2^{i^2} \leq n \\
 & i^2 \leq \log n \Rightarrow i \leq \sqrt{\log n} \\
 & \Theta(\sqrt{\log n})
 \end{aligned}$$

```
void f4(int n) {
    int x = 2;
    int y = 1;
    while (y <= n) {
        y = y * x;
        x = x * x;
    }
}
```

$i = 0$
 $x = 2 \quad y = 1$
 $y = 1 \cdot \textcircled{2}$
 $x = 2 \cdot 2$
 $i = 1$
 $y = 2 \cdot \textcircled{4}$
 $x = 4 \cdot 4$

$$\begin{aligned}
 & i = 2 \\
 & y = 8 \cdot \textcircled{16} \\
 & x = 16 \cdot 16 \\
 & 1 \cdot 2 \cdot 4 \cdot 16 \dots \\
 & 2^0 \cdot 2^1 \cdot 2^2 \cdot 2^3 \cdot 2^4 \\
 & 2^{2^i} \leq n \Rightarrow 2^i \leq \log n
 \end{aligned}$$

$$i \leq \log \log n \Rightarrow \Theta(\log \log n)$$

Parcial 11/11/2019

Problema 2: (2,5 puntos)

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 mateixa 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.

$$i = 0$$

$$r[0] = f(i) = 1$$

$$i = 1$$

$$r[1] = f(i) = 1$$

$$i = 2$$

$$r[2] = f(2) = 2$$

$$i = 3$$

$$r[3] = f(3) = 4$$

$$i = 4$$

$$r[4] = f(4) = 1$$

$$r[1, 1, 2, 4, 1]$$

Retorna $f \circ g$.

$$f(g(i))$$

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

Fem la recurrència

$$T(n) = 1 * T(n-1) + \Theta(1) \quad a=1, b=1, K=0 \quad a < K$$

$$T(n) = \Theta(n^{k+1}) = \Theta(n)$$

Parcial 11/11/2019

Problema 2: (2,5 puntos)

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

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

$$f[2, 3, 1]$$

$$k = 3$$

f o f o f

- Assumint que $k \geq 0$, què retorna la funció *misteri_2*? No cal que justifiqueu la resposta.

Retorna f^k , es a dir $f \circ f \circ f \dots k$ vegades

- Quin és el cost de *misteri_2* en funció només de k ?

$$T(k) = 1 \cdot T(k-1) + \Theta(1) \quad a=1, b=1, k=0$$

$$T(k) = \Theta(k)$$

- 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;
    }
    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));
    }
}
```

Anàlisi del cost en funció de k .

$$T(k) = 1 \cdot T(K/2) + \Theta(1) \quad a=1, b=2, k=0$$

$$\alpha = \log_b a = \log_2 1 = 0 \quad ; \quad \alpha = k \quad T(k) = \log k$$

Parcial 11/11/2019

Problema 3: (3,25 puntos)

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 els 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; Θ(1)
    for (int j = l + 1; j ≤ r; ++j)      j = 0+1   j ≤ r   ++j
        if (v[j] > v[p]) p = j; Θ(1)      j = 1   j ≤ 5
    return p;                                Θ(m)
}

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

`max_suma` implementa ordenació per selecció, en cas pitjor el seu cost es $\Theta(m^2)$. Per tant el cost es $\Theta(m^2) + \Theta(m) = \Theta(m^2)$.

Parcial 11/11/2019

Problema 3: (3,25 puntos)

- 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 asymptòticament que *max_suma*. Indiqueu clarament quin és el cost resultant.

En comptes d'utilitzar un selection sort podem ordenar el vector amb un mergesort amb cost $\Theta(m \log m)$ i agrupar els enters consecutivament.

- 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 són els dos nombres més grans de S , aleshores una expressió que conté els productes $x_0 \cdot y$ i $x_1 \cdot z$, 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 .

Sabem que $x_0 \cdot y$ i $x_1 \cdot z$ no pot ser màxima.

$$(x_0 \cdot x_1 + y \cdot z) - (x_0 \cdot y + x_1 \cdot z) > 0,$$

$$x_0(x_1 - y) + z(y - x_1) > 0,$$

$$x_0(x_1 - y) - z(x_1 - y) > 0;$$

$$(x_0 - z)(x_1 - y) > 0; \quad x_0 > z \wedge x_1 > y;$$

* Cas base per $m=0$. suma = 0. cert.

* Cas inductiu: per $m > 0$.

HIPOTESI D'INDUCCIÓ: per $m-2$ *max-suma* és màxima.

Donat un conjunt de m nombres S , podem ordenar

S de manera creixent tq $S = \{x_0, x_1, x_2, \dots, x_m\}$

on $x_0 > x_1 > x_2 > \dots > x_m$. Amb això, tal com hem demostrat anteriorment, podem ajuntar el conjunt en parells consecutius els quals multiplicarem entre si per tal d'aconseguir la suma màxima de

S . Si treiem els dos primers elements, $m-2$ elements restants, per hipòtesi d'inducció sabem que aquests

$m-2$ elements formen la suma màxima, de la forma,

$x_2 \cdot x_3 + \dots + x_{m-1} \cdot x_m$. Si afegim 2 elements més

grans com que aquests dos tindran $x_0 \cdot x_1 > x_2 \cdot x_3$

la suma més gran amb m elements serà de la

forma $x_0 \cdot x_1 + x_2 \cdot x_3 + \dots + x_{m-1} \cdot x_m$.

Parcial 25/04/2019

Problema 4: (3 puntos)

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 suposem 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 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);
SEARCH1(X, A, MI+1, MJ+1, N/2);

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);
SEARCH1(X, A, I, MJ+1, N/2);

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; ;
```

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

Parcial 25/04/2019

Problema 4: (3 puntos)

Problema 4: (3 puntos) 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 .

criudem a la funció `search(x, A, 0, 0, N)` retorna true si només si x apareix a A .

$$T(n) = 3T(n/2) + \Theta(1)$$

$$a = 3 \quad \alpha = \log_b a = \log_2 3 > K$$

$$b = 2$$

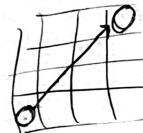
$$k=0 \quad T(n) = \Theta(n^{\log_2})$$

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.

La funció de search2 comença a la primera columna, a la última fila. ~~Si~~ ~~sa~~ Es posiciona al valor més gran de la columna i més petit de la fila. Si $A[i][j] > x$ significa que hem de pujar de fila, si no ens hem de desplaçar de columnes. ~~Si~~ ~~correcte~~ Si no compleix cap dels dos ifs llavors sabem que o $A[i][j] = x$, en aquest cas retorna true. ~~Si~~ ~~incorrecte~~ ~~Si~~ ~~incorrecte~~ Si surt del bucle retorna false.



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

(0.75 pts.) Analitzeu el cost en el cas pitjor de selecció en funció de $n = A[0..n-1]$. El cas pitjor es quan x està a $A[0..n-1]$ que llavors farem $2 \cdot (n-1)$ iteracions. $T(n) = \Theta(n)$.

Parcial 15/04/21

Problema 1: (5 punts)

- (a) (1,25 pts.) Escriviu C o F dins de cada casella indicant si l'affirmació corresponent és certa o falsa, respectivament:

$2^{2n} \in O(2^n)$ F

$\log(2n) \in O(\log(n))$ C

$2^{2n} \in \Omega(2^n)$ C

$\log(2n) \in \Omega(\log(n))$ C

$2^{2n} \in \Theta(2^n)$ F

$\log(2n) \in \Theta(\log(n))$ C

Justifiqueu la vostra resposta:

$$\lim_{n \rightarrow \infty} \frac{2^{2n}}{2^n} = \lim_{n \rightarrow \infty} 2^n = \infty$$

$$\lim_{n \rightarrow \infty} \frac{\log 2n}{\log n} = \lim_{n \rightarrow \infty} \frac{\log 2 + \log n}{\log n} = \lim_{n \rightarrow \infty} \frac{1 + \log n}{\log n} = 1.$$

- (b) (1,25 pts.) Considerieu el codi següent:

```

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

$$\begin{aligned}
x &= 2 & y &= 1 \\
&& i \leq n \\
y &= 1 + \textcircled{2} \\
&& x = 3 \\
&& y = 2 + \textcircled{3} \\
x &= 3 & y &= 3 \\
&& x = 4 \\
&& y = 4 + \textcircled{4} \\
&& x = 5
\end{aligned}$$

$x = 3 \quad y = 3$

$y = 3 + \textcircled{3}$

$x = 4$

$x = 4 \quad y = 6$

$y = 6 + \textcircled{4}$

$\sum_{i=0}^n i \approx i \cdot n^2$

$i^2 \leq n, i \leq \sqrt{n}$

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

→ Es correcte. Observem que el bucle manté la següent invariant:

Si $x \in A$ valors ho fa entre la fila 0 i i (incloses), i entre la columna 0 i j (incloses).

• $A[i][j] > x$ aleshores $x \notin A$ a la fila $i : j \leq k \leq N \Rightarrow A[i][k] \geq A[i][j]$
 > 0 .

• $A[i][j] < x$ aleshores $x \in A$ a la columna $j : 0 \leq k \leq i$

$\Rightarrow A[k][j] \leq A[i][j] < x$.

El programa retorna true si dins del bucle trouem $A[i][j] == x$.
Si retorna false valors $i < 0 \circ j \geq N, x \notin A$.

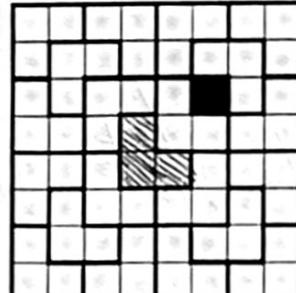
Parcial 15/04/21

Problema 2: (5 punts)

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:



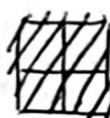
Com és d'esperar, les peces no es poden solapar ni sortir de la graella. Per exemple, per una graella 8×8 i la casella bloquejada en negre, una possible solució és:



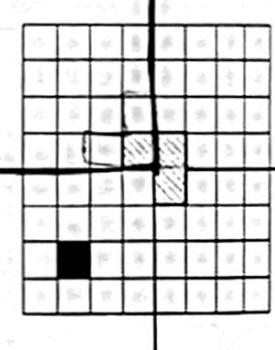
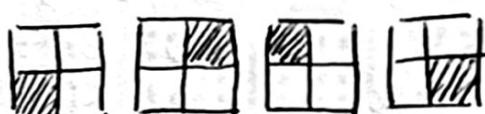
(a) (2 pts.) Demostreu que per a tot $n \geq 0$, sigui quina sigui la posició de la casella bloquejada, sempre podrem omplir la resta de les caselles amb les peces indicades.

Pista: Observeu la següent figura.

* Caso base per $n=0$.



* Cas base per $n=1$.



~~* SUPOSAC~~

~~HIPOTESI~~

* Cas inductiu: Per $n > 0$.

Suposem que per $n-1$ és cert i demostrarem que per n també.

HIPOTESI D'INDUCCIÓ: per $n-1$ cert.

Donada una graella de $2^n \times 2^n$ on $n > 0$ amb una casella bloquejada, podem partir la graella ~~en~~ en 4 subgraelles iguals de $2^{n-1} \times 2^{n-1}$. Les tres graelles que no tenien una peca bloquejada els hi podem afegir una afegint una peca a la graella de $2^n \times 2^n$ tal que al partir la graella tinguin una peca bloquejada cada una. Per hipòtesi d'inducció com sabem que per caselles de $2^{n-1} \times 2^{n-1}$ es compleix per caselles de $2^n \times 2^n$ també.

Parcial 15/04/21

Problema 2: (5 punts)

(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;
    x   y
// Returns quadrant of pos in square [i_1,i_r] x [j_1,j_r]
// Quadrants are:
// 0 1
// 2 3
int quadrant(Coord pos, int i_1, int i_r, int j_1, int j_r)
    int size = j_r - j_1 + 1;
    int i_m = i_1 + size/2;
    int j_m = j_1 + size/2;

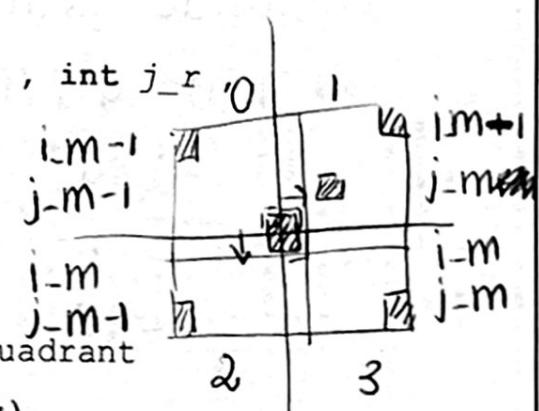
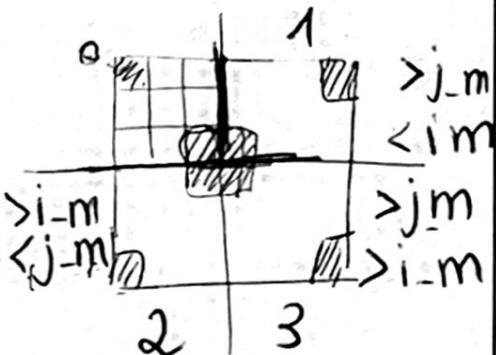
    if (pos.first < i_m and pos.second < j_m) return 0;
    if (pos.first < i_m and pos.second > j_m) return 1;
    if (pos.first > i_m and pos.second < j_m) return 2;
    return 3; // pos.first > i_m and pos.second > j_m
}

void fill (vector<vector<int>>& M, int i_1, int i_r, int j_1, int j_r, Coord c_blocked, int& num){
    if (i_1 == i_r) return; // 1x1 Θ(1)
    int size = j_r - j_1 + 1; Θ(1)
    int i_m = i_1 + size/2; // Midpoints Θ(1)
    int j_m = j_1 + size/2; Θ(1)

    vector<Coord> coords_blocked(4); // Blocked cell in each quadrant
    coords_blocked[0] = {i_m-1, j_m-1}; Θ(1)
    coords_blocked[1] = {i_m-1, j_m}; Θ(1)
    coords_blocked[2] = {i_m, j_m-1}; Θ(1)
    coords_blocked[3] = {i_m, j_m}; Θ(1)

    int q = quadrant(c_blocked, i_1, i_r, j_1, j_r); Θ(1)
    coords_blocked[q] = c_blocked; Θ(1)
    for (int k = 0; k < 4; ++k) Θ(1)
        if (M[coords_blocked[k].first][coords_blocked[k].second] == -1)
            M[coords_blocked[k].first][coords_blocked[k].second] = num; Θ(1)
    ++num;
    fill (M, i_1, i_m-1, j_1, j_m-1, coords_blocked[0], num); // Q0
    fill (M, i_1, i_m-1, j_m, j_r, coords_blocked[1], num); // Q1
    fill (M, i_m, i_r, j_1, j_m-1, coords_blocked[2], num); // Q2
    fill (M, i_m, i_r, j_m, j_r, coords_blocked[3], num); // Q3
}

```



$$T(n) = 4 \cdot T(n-1) + \Theta(1)$$

$$a = 4 \quad a > 1 \quad T(n) = (4^n)$$

$$b = 1$$

$$K = 0$$

Parcial 15/04/21

Problema 2: (5 punts)

```

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); Θ(m²)
}

```

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

En funció de n :

$$T(n) = 4 \cdot T(n-1) + \Theta(1)$$

$$a=4$$

$$b=1$$

$$k=0$$

$$a > 1 \Rightarrow T(n) = \Theta(4^n)$$

En funció de m :

Com que $m = 2^n \times 2^n = 4^n$ podem afirmar que la funció es líneal en nombre de caselles.

Clase 3. Problemas

Parcial 08/11/2021

Problema 1: (5 puntos) Respondeu les preguntes següents:

(a) (1 pt.) Considerieu la funció mystery:

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

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

El cost en el cas pitjor vindrà donat pel for, i serà quan s'hagin de fer totes les iteracions. El for acaba quan $i \cdot i > n$, es a dir quan $i^2 > n$; $i > \sqrt{n}$ per tant fa \sqrt{n} iteracions i $T(n) = \Theta(\sqrt{n})$

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

```
int j = 0;  
int s = 0;  
for (int i = 0; i < n; ++i) Θ(n) → i = j2;  
    if (i == j * j) { Θ(1) →  
        for (int k = 0; k < n; ++k) ++s; Θ(n)  
        ++j;  
    }
```

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

El for exterior s'executa sempre amb cost línearly $\Theta(n)$ i el segon for s'executará sempre que $i = j^2$. Es a dir el segon for s'executará tant vegades com quadrats hi hagin entre 0 i $n-1$, \sqrt{n} vegades i no s'executara $n-\sqrt{n}$ vegades. Per tant el cost del codi $(n-\sqrt{n}) \cdot \Theta(1) + (\sqrt{n}) \cdot \Theta(n)$
 $= \Theta(n\sqrt{n})$

(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) Θ(m) → n/2
            for (int j = m+1; j <= r; ++j) Θ(m) n/2
                if (v[i] == v[j]) ++n_crossed; Θ(1)
        return n_left + n_right + n_crossed ; Θ(1)
    }
}

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

```

$$T(n) = 2 \cdot T(n/2) \cdot \Theta(n^2)$$

$$a = 2 \quad \alpha = \log_2 2 = 1$$

$$b = 2 \quad \alpha < K$$

$$K = 2 \quad 1 < 2 \Rightarrow \Theta(n^2)$$

En funció d' n , el cost d'una crida a $pairs(v)$ és $\Theta(n^2)$. Justifiqueu la vostra resposta:

La funció implementa un binary search amb dues trucades recursives del tipus divisor amb $n/2$. El cost de pàirs vindrà donat per la recursivitat

$T(n) = 2 \cdot T(n/2) \cdot \Theta(n^2)$ on $\Theta(n^2)$ es el cost dels dos bucles. $T(n) = \Theta(n^2)$.

(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]]; Θ(n)

    int res = 0;
    for (int i = 0; i < K; ++i) res += (times[i] * (times[i]-1))/2; Θ(1)
    return res; Θ(1)
}
```

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

La funció `pairs_2` està formada per expressions aritmèticologiques i declaracions de cost constant $\Theta(1)$ per tant el seu cost és dels dos fors. El primer for fa n iteracions d'algo constant per tant té cost $\Theta(n)$ i el segon for depen de la constant K per tant haurà cost constant $\Theta(1)$. En total `pairs_2` té cost $\Theta(n)$.

Expliqueu per què el codi anterior és una solució correcta

`Pairs_2` calcula el nombre de parelles (i, j) tals que $0 \leq i < j \leq n-1$. `Pairs_2` té un vector `times` de tamany K que va guardant per a cada posició quantes vegades es repeteix un valor x a v ($++times[v[i]]$). A continuació mira el vector `times` per determinar el numero de parelles a v que compleixen $0 \leq i < j \leq n-1$ amb la formula $(times[i] \cdot times[i]-1)/2$, que calcula el nombre de combinacions possibles de parelles. $v[i]=v[j]=k$

+10/6/2021

```

int a;
int f( const vector<int>&v, int l, int r ) {
    if ( l > r ) return 0;
    int tot = 0;
    int m = ( l + r ) / 2;
    for ( int i = 0; i < a; ++i ) {
        if ( i % 2 == 0 ) tot += f( v, l, m );
        else tot += f( v, m + 1, r );
    }
    for ( int i = l; i <= r; ++i ) {
        for ( int j = i + 1; j <= r; ++j ) {
            tot += v[i] * v[j];
        }
    }
    return tot;
}

```

$$T(n) = a \cdot T(n/2) + \Theta(n^k)$$

$$\frac{a}{b} = 2 \quad \alpha = \log_b a = \log_2 a$$

$$k = 2$$

Si $\log_2 a < 2$
 $a < 4$. $T(n) = \Theta(n^2)$

Si $\log_2 a = 2$
 $a = 4$. $T(n) = \Theta(n^2 \cdot \log n)$

Si $\log_2 a > 2$; $a > 2^2$; $a > 4$; $T(n) = \Theta(n^{\log_2 a})$

Parcial 08/11/2021

$$v = [x_0, x_1, \dots, x_{n-1}, x_n]$$

$$x_0 < x_1 < \dots < x_{n-1} < x_n$$

Problema 2: (5 puntos)

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

- 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) { //Binary search
    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;
            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 = n - p - q
cout << res << endl;

```

$x = 3$
 $\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 & \cancel{6} \\ \cancel{-5} & \cancel{-4} & \cancel{-3} & \cancel{-2} & \cancel{-1} \end{array}$

$n = 6$
 $\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ -6 & -5 & -4 & -3 & -2 & -1 \\ \downarrow 3 \\ 6 - 3 - 2 = 1 \end{array}$

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

El cost de first occurrence es $\Theta(\log n)$ i els costos dels fors son $\Theta(n)$. En cas pitjor valdrà $\Theta(n^2)$.

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

És correcte. El codi calcula essencialment la primera i última posició de x a v i mira quants elements hi ha entre si. Al invertir el vector tenim v ordenat decreixentment on la primera aparició x a v ~~original~~ invertit coincideix amb la última a v original. Per tant sabem que $n-1-q$ representa la última pos. ~~de x av.~~ de x av. Per tant el número d'elements entre p i $n-1-q$ és $(n-1-q)-p+1 = n-q-p$.

(c) (2 pts.) Existeix un algorisme per calcular el nombre d'aparicions que sigui, en cas pitjor, asymptòticament 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.

Si, si modifiquem `first_occurrence` podem trobar la última posició de x a v amb cost $\Theta(\log n)$ amb `last_occurrence`. Llavors només ens cal una crida $p = \text{first_occurrence}(x, v)$ una segona crida $q = \text{last_occurrence}(x, v)$ i $\text{res} = q - p + 1$, tot amb cost $\Theta(\log n)$

```
int last_occurrence ( int x, const vector<int>& v ) {
    return last_occurrence ( int x, const vector<int>& v, 0, v.size() )
}
int last_occurrence ( int x, const vector<int>& v, int l, int r ) {
    if ( l > r ) return -1;
    int m = ( l + r ) / 2;
    if ( v[m] > x ) return last_occurrence ( x, v, l, m - 1 );
    else if ( v[m] < x ) return last_occurrence ( x, v, m + 1, r );
    else {
        if ( m == r || v[m + 1] != x ) return m;
        return last_occurrence ( x, v, m + 1, r );
    }
}
```

Parcial 06/11/20

Problema 1: (2,5 puntos)

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:

```

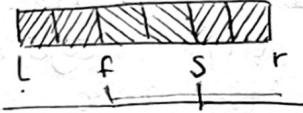
bool tri_search (const vector<int>& v, int l, int r, int x) {
    if (l > r) return false;
    else {
        int n_elems = (r-l+1);
        int f = l + n_elems/3;
        int s = r - n_elems/3;

        if (v[f] == x || v[s] == x) return true;
        if (v[f] > x) return tri_search(v, l, f-1, x);
        if (v[s] > x) return tri_search(v, f+1, s-1, x);

        return tri_search (v, s+1, r, 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.

El cost de `tri-search` vindrà donat per la recursivitat $T(n) = 1 \cdot T(n/3) + \Theta(1)$. Apliquem el teorema mestre per recursivitats sustractores: $a = 1$; $b = 3$; $k = 0$; $\alpha = \log_b a = \log_3 1 = 0$; $\alpha = k$ per tant $T(n) = n^k (\log n) = \log n$. Per tant el cost de `tri-search` es $\Theta(\log n)$.

Parcial 06/11/20

Problema 1: (2,5 puntos)

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

$$f = n(\log n)^{1/2}; \quad g = n(\log n)^{1/3}$$

$$\lim_{x \rightarrow \infty} \frac{x(\log n)^{1/3}}{x} = \lim_{x \rightarrow \infty} (\log n)^{1/3} = \infty. \quad g \in \Omega(n)$$

$$\lim_{x \rightarrow \infty} \frac{x(\log n)^{1/2}}{x} = \lim_{x \rightarrow \infty} (\log n)^{1/2} = \infty \quad f \in \Omega(n)$$

$$\lim_{x \rightarrow \infty} \frac{x(\log n)^{1/3}}{x(\log n)} = \lim_{n \rightarrow \infty} \frac{(\log n)^{1/3}}{\log n} = \lim_{n \rightarrow \infty} \frac{1}{(\log n)^{2/3}} = 0 \quad g \in O(n\log n)$$

$$\lim_{x \rightarrow \infty} \frac{x(\log n)^{1/2}}{x(\log n)} = \lim_{n \rightarrow \infty} \frac{(\log n)^{1/2}}{\log n} = \lim_{n \rightarrow \infty} \frac{1}{(\log n)^{1/2}} = 0 \quad f \in O(n\log n)$$

$g \in \Omega(n) \wedge g \in O(n\log n)$

$f \in \Omega(n) \wedge f \in O(n\log n)$

$$\frac{1}{2} - \frac{1}{3} = \frac{3}{2 \cdot 3} - \frac{2}{2 \cdot 3} = \frac{3}{6} - \frac{2}{6} = \frac{1}{6}$$

$$\lim_{n \rightarrow \infty} \frac{x(\log n)^{1/2}}{x(\log n)^{1/3}} = \lim_{n \rightarrow \infty} \frac{(\log n)^{1/2}}{(\log n)^{1/3}} = \lim_{n \rightarrow \infty} (\log n)^{1/6} = \infty$$

$f \notin \Theta(g)$

Parcial 06/11/20

Problema 2: (7,5 puntos)

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 suggeréix 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; // Visitat
                }
            }
            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.

El cost en cas pitjor sera quan s'executin els dos fors totes les vegades es a dir $\Theta(n^2)$ això passa quan tenim un vector on tots els seus elements són diferents.

- 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.

El cost en cas millor serà quan tinguem un vector on tots els seus elements són iguals. d'aquesta manera quan $i=0$ es passaran tots els elements a -1 i tindrem cost $\Theta(n)$ ja que el segon for no s'executa.

- 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?

Si sabem que v sempre tindrà 100 elements diferents el for intern s'executara 100 vegades en el cas pitjor tenint ara cost $\Theta(100 \cdot n) = \Theta(n)$ per tant el seu cost ha canviat.

Parcial 06/11/20

Problema 2: (7,5 puntos)

- (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()); → depen de l'algorisme d'ordenació
    int i = 0;
    while (i < n) { i < n
        int times = 0, j = i;
        while (j < n and v[j] == v[i]) { → cas pitjor visita tots els
            ++times;
            ++j;
        }
        if (times > n/2) return v[i];
        i = j;
    }
    return -1;
}

```

cas pitjor visita tots els elem de V, $\Theta(n)$

sino visita $n/2$ que segueix tenint $\Theta(n)$.

- 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 ?

Ordenació per inserció té cost $\Theta(n)$ en cas millor i $\Theta(n^2)$ en cas pitjor. L'algorisme `dominant_sort` sense tenir en compte `own-sort`, en cas pitjor mira els n elements de V i en cas millor mirara $n/2$ elements. En els dos casos té cost $\Theta(n)$. Per tant:

- cost cas millor : $\Theta(n) + \Theta(n) = \Theta(n)$
- cost cas pitjor : $\Theta(n^2) + \Theta(n) = \Theta(n^2)$

- Si `own_sort` implementa un quicksort, quin és el cost en cas millor i pitjor de `dominant_sort` en funció de n ?

El quicksort té cost $\Theta(n \log n)$ en cas millor i $\Theta(n^2)$ en cas pitjor. Sense tenir en compte l'ordenació, tal com he explicat anteriorment `dominant_sort` té cost $\Theta(n)$. Per tant:

- cost cas millor : $\Theta(n \log n) + \Theta(n) = \Theta(n \log n)$
- cost cas pitjor : $\Theta(n^2) + \Theta(n) = \Theta(n^2)$

Parcial 06/11/20

Problema 2: (7,5 puntos)

(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, [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;
}

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

```

$$T(n) = 1 \cdot T(n-1) + \Theta(1)$$

$$a=1 \quad k=0$$

$$b=1 \quad a=1 \rightarrow T(n) = \Theta(n)$$

$$T(n) = 2 \cdot T(n/2) + \Theta(n)$$

$$a=2 \quad \alpha = (\log_b a) = 1$$

$$b=2 \quad \text{dels } k \text{ } \alpha = K$$

$$K=1 \quad \alpha = K$$

$$T(n) = \Theta(n \log n)$$

Analitzeu el cost de dominant_divide en cas pitjor en funció de n.

La funció dominant_divide ~~devoldria~~ té cost que ve donat per la següent recurrència $T(n) = 2 \cdot T(n/2) + \Theta(n)$. El cost de l'no recursiu ve donat per la funció times amb cost donat per la recurrència $T(n) = 1 \cdot T(n-1) + \Theta(1)$ on $a=1 \rightarrow T(n) = \Theta(n)$. Resolem la recurrència $a=2, b=2, k=1$, $\alpha = (\log_b a) = (\log_2 2) = 1$, $\alpha = K$, $T(n) = \Theta(n \log n)$.

Parcial 31/03/2022

Problema 1: (6 punts)

Responeu les preguntes següents:

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

```
int n; cin >> n;
vector<int> v(n); // Θ(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;
```

0	1	2	3	4	5
11	2	3	4	5	6

random_shuffle					
0	1	2	3	4	5
15	1	6	3	4	2

$S_0 = 5$	$S_2 = 12$	$\sum_{i=1}^n i = \frac{n(n+1)}{2}$
$S_1 = 6$	$S_3 = 15$	
	$S_4 = 19$	

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 , que calcula el codi anterior i quin és el seu cost asimptòtic?

Si el vector no estuviera ordenado, cada iteración i del bucle interno se ejecuta $v[i]$ veces, es decir $i+1$ veces.

Como que la i va de 0 a $n-1$ el numero de iteraciones totales del bucle interno es $1+2+3+\dots+n = \frac{n(n+1)}{2}$

El cost de l'algorisme sense la permutació es $\Theta(n^2)$.

La diferéncia si permitem el vector es que en la vuelta i no tienen por que hacerse $i+1$ iteraciones.

Como que v es una permutació de $\{1, 2, 3, \dots, n\}$ habrá una vuelta que haga 1 iteración, otra que haga tres, otra que haga 2. Por lo tanto el código calcula lo mismo sin la ordenación con el mismo coste.

Parcial 31/03/2022

Problema 1: (6 punts)

(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;
}
```

$n = 5$

$j = 1 \quad j < 5$

$k = 2$

$2 < 5 \rightarrow k = 2 \cdot 2 = 4$

$4 < 5 \rightarrow k = 4 \cdot 4 = 16$

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

El bucle extern ~~while~~ fa n iteracions. Per tant el cost del bucle serà $n \cdot$ (cost while). Mirem el while a la primera iteració tindrem $i=0 \ k=2$ a la segona $i=1 \ k=4$, a la tercera $i=2 \ k=16$. Observem que $k = 2^{2^i}$. Busquem el valor de tipus $\Theta(2^{2^i}) \geq n$. Pertant podem deduir que el while fa $\log \log n$ iteracions. El cost total sera $T(n) = n \log \log n$

Parcial 31/03/2022

Problema 1: (6 punts)

(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?

$n \geq 1 \rightarrow v = \{1, 2n, 2, 2n-1, 3, 2n-2, 4, 2n-3, \dots, n, n+1\}$

Per exemple $n=5 \rightarrow v = \{1, 10, 2, 9, 3, 8, 4, 7, 5, 6\}$

El cost de l'algorisme d'ordenació per inserció sobre aquest vector ve determinat pel número d'intercanvis que caldrà fer. Si observem el vector veiem que el número d'intercanvis depenen dels números més grans que tinguem a l'esquerra.

- Per 1 i $2n$ tenim 0 elem
- Per 2 i $2n-1$ tenim 1 elem
- Per 3 i $2n-2$ tenim 2 elem

Pertant el número d'it.

$0 + 0 + 1 + 1 + \dots + (n-1) + (n-1)$

que és el mateix que

dir $(0 + 1 + 2 + 3 + \dots + n-1) \cdot 2 = \frac{n(n-1)}{2} \cdot 2 = n \cdot (n-1) = n^2$

Per tant tindrem cost $\Theta(n^2)$.

Parcial 31/03/2022

Problema 2: (4 punts)

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 conegit 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; Θ(1)
    return v[pos] == x or find(x, v, pos-1); → T(n) = 1 · T(n-1) · Θ(1)
}

int inefficient (const vector<int>& v) {
    int n = v.size(); Θ(1)
    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*?

En el cas pitjor la funció farà les n iteracions y retornarà n . Per tant el cost de la funció en cas pitjor sera $n \cdot \text{cost find}$, ja que la resta d'operacions son $\Theta(1)$. El cost de *find* ve donat per la recursivitat $T(n) = 1 \cdot T(n-1) \cdot \Theta(1)$ $a=1, b=1, k=1$ com que $a=1$ apliquem el teorema mestre i tenim que el cost de *find* és $T(n) = \Theta(n)$. Per tant el cost total es $\Theta(n \cdot n) = \Theta(n^2)$.

Parcial 31/03/2022

Problema 2: (4 punts)(b) (2.5 pts.) Doneu vosaltres una funció que solucioni aquest problema i que trigui, en cas pitjor, $\Theta(\log n)$.

si està ordenado

 $v[m] = m$ por lo tantosi $v[m] > m$ miro por la izq.

efficient(v, l, m-1)

```

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) return m;
    else if (v[m] < m) return efficient (v, l, m-1);
    else return efficient (v, m+1, r);
}

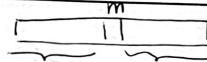
int efficient (const vector<int>& v) {
    return efficient (v, 0, v.size()-1);
}

```

Parcial 31/03/2022

Problema 2: (4 punts)(b) (2.5 pts.) Doneu vostres una funció que soluciò aquest problema i que trigui, en cas pitjor, $\Theta(\log n)$.

si està ordenado

 $V[m] = m$ por lo tantosi $V[m] > m$ miro por la izq., efficient ($v, l, m-1$)

```

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);
}

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

```

Parcial 11/11/2019

Problema 4: (3,25 puntos)

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 $p_0 > p_1 > \dots > p_d < p_{d+1} < \dots < p_{n-1}$.

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 retorne 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) { // Busca el dia d.
```

```
    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, [l], [m-1]);
```

```
        else if ( [P[m]] > P[m+1] ) return f(p, [m+1], [r]);
```

```
        else return m; // P[m-1] > P[m] AND P[m] < P[m+1]
```

```
}
```

C V
pair<int,int> max_guany (const vector<int>& p) {

```
    return { f (p,0,p.size()-1), [P.size()-1] };
```

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:

El cost de la funció ve donat per la recurrencia

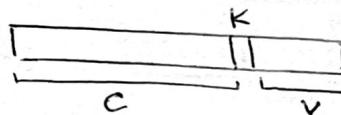
$$T(n) = 1 \cdot T(n/2) \cdot \Theta(1) = \Theta(\log n).$$

Parcial 11/11/2019

Problema 4: (3,25 puntos)

- 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) {
    int c = p[k];
    for (int i = 0; i < k; ++i) {
        if (p[i] < c) c = p[i];
    }
    int v = p[k];
    for (int i = k + 1; i < p.size(); ++i) {
        if (p[i] > v) v = p[i];
    }
    return v - c;
}

```

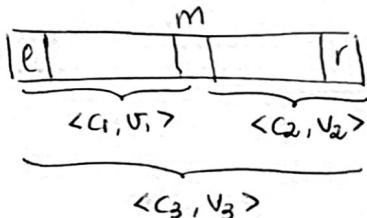
Parcial 11/11/2019

Problema 4: (3,25 puntos)

- c) (1 pt.) Afrontem finalment el problema general, en el que la seqüència p pot tenir qualsevol 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.



- $\langle c_1, v_1 \rangle$ màxima ganància comprant i venent per l'esquerra
- $\langle c_2, v_2 \rangle$ màxima ganància comprant i venent per la dreta

- $\langle c_3, v_3 \rangle$ màxima ganància comprant per l'esquerra i venent per la dreta.

El resultat vindrà donat pel màxim dels tres.

El cost d'aquesta funció serà, sent $n = r - c + 1$

$$T(n) = 2 \cdot T(n/2) + \Theta(n)$$

$$a = 2 \quad \alpha = \log_b a = \log_2 2 = 1$$

$$b = 2$$

$$k = 1$$

$$\alpha = k \quad T(n) = \Theta(n \log n)$$