

notebook

GOOD MONDAYS PAPER

Sections

1 *Anàlisis d'algorithmes*

2 *Dividir i vèncer*

3 *Diccionaris*

4

5

6

7

8

9

10

11

12

T

1

2

3

4

5

6

7

8

9

10

11

12

•
•
•
T
1
2
3
4
5
6
7
8
9
10
11
12

Section One

Analisis d'algorithmes

[BACK TO INDEX](#)

MIDA DE L'ENTRADA I COST

o Cas pitjor: $T_{\text{pitjor}}(n) = \max \{ T(x) | x \in E \wedge |x|=n \} \Rightarrow$ El màxim que triga un algorisme en tractar dades de mida n .

o Cas mitj: $T_{\text{mitj}}(n) = \sum_{x \in E, |x|=n} \Pr(x) T(x)$

o Cas millor: $T_{\text{millor}}(n) = \min \{ T(x) | x \in E \wedge |x|=n \}$

★ Si $T_p(n) = T_{\text{millor}}(n) \Rightarrow$ sabem amb certesa que el cost serà el mateix per tot $|x|=n | x \in E$
ex: vector $\prec T \succ v(n)$ busquem un element x a v .

Cas millor: que x sigui a $v[0]$

Cas pitjor: que x no hi sigui

Cas mitj: que x sigui per la meitat

NOTACIÓ "O" GRAN

o $O(f)$ representa la classe de funcions que creixen asymptòticament com f ($g \in O(f) \Rightarrow g < f$)

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

ex: $f(n) = 3n^3 + 5n^2 - 7n + 41$. $f = O(n^3)$?

$$3n^3 + 5n^2 - 7n + 41 \leq n^3 \Rightarrow 3n^3 + 5n^2 + 41 \leq 8n^3 + 41 \Rightarrow 8n^3 + 41 \leq 9n^3 \Rightarrow 41 \leq n^3$$

$$C = 9 \quad ; \quad n_0 = ? \quad 1^3 = 1 ; \quad 2^3 = 8 ; \quad 3^3 = 27 ; \quad 4^3 = 64 \quad ! \Rightarrow n_0 = 4$$

NOTACIÓ " Ω "

o $\Omega(f)$ representa la classe de funcions tq $g \in \Omega(f) \Rightarrow g > f$

o $\Omega(f) = \{ g : \mathbb{N} \rightarrow \mathbb{R}^+ | \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 : g(n) \geq c \cdot f(n) \}$

NOTACIÓ " Θ "

o $\Theta(f)$ representa el conjunt de funcions tq $g \in \Theta(f) \Rightarrow g = f$

o $\Theta(f) = \{ g : \mathbb{N} \rightarrow \mathbb{R}^+ | \exists c_1, c_2 \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 : c_1 f(n) \leq g(n) \leq c_2 f(n) \}$

LIMITS

Si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f \in O(g) \wedge f \notin \Omega(g)$

Si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f \in \Omega(g) \wedge f \notin O(g)$

Si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = C$ on $0 < C < \infty \Rightarrow f \in \Theta(g) \wedge g \in \Theta(f)$

NOTACIÓ ASIMPTÒTICA

POLINOMIS sigui $p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_0$ Amb $a_k > 0 \Rightarrow p(n) \in \Theta(n^k)$

LOGARITMES $a, b > 1 \Rightarrow \log_a(n) \in \Theta(\log_b(n))$

LOG + POL + EXP $a, b, c > 0$

o $(\ln n)^a \in O(n^b)$ però $(\ln n)^a \notin \Omega(n^b)$

o $n^b \in O(c^n)$ però $n^b \notin \Omega(c^n)$

FORMES DE CREIXEMENT

COSTOS FREQUENTS

o Constant $\Theta(1)$

o Logarítmic $\Theta(\log n) \Rightarrow$ cerca binària

o Radical $\Theta(\sqrt{n})$

o Lineal $\Theta(n)$

o Quasilineal $\Theta(n \log n)$

o Quadràtic $\Theta(n^2)$

o Cúbic $\Theta(n^3)$

o Polinòmic $\Theta(n^k)$

o Exponencial $\Theta(k^n)$

o Altres funcions $\Theta(n!)$, $\Theta(n^n)$

ALGORISMES ITERATIUS

- Comparacions, operacions aritmètiques, lect./escripc., accessos, declaracions $\Rightarrow \Theta(1)$
- Assignacions $x = E \Rightarrow$ cost evaluar $E +$ cost copiar E en v

o condicionals :

* $\text{if}(E) A \rightarrow$ cas pitjor: E cert \Rightarrow cost if = $\Theta(E) + \Theta(A)$

cas més petit: E fals \Rightarrow cost if = $\Theta(E)$

* $\text{if}(E) A \rightarrow$ cas pitjor: cost if = $\Theta(E) + \Theta(\max\{A, B\})$

$\text{else } B \rightarrow$ cas més petit: cost if = $\Theta(E) + \Theta(\min\{A, B\})$

- Bucle \Rightarrow Cost bucle = cost(A_0) + $\sum_{i=1}^n (\Theta(E_i) + \Theta(A_i))$

* $\text{while}(A) \rightarrow$ cas pitjor: cost bucle = $\Theta(n^{\text{iterations}} \cdot (A+B))$

\downarrow cas més petit: cost bucle = $\Theta(A) \Rightarrow$ no entra

ALGORISMES RECURSIVUS

El cost d'un algorisme recursiu s'expressa sovint en forma de recurrència

- Recurrència \Rightarrow equació o inequació que descriu una funció expressada en termes del seu valor per entrades més petites. Per trobar una recurrència cal:

1. Paràmetre de recursió n (normalment, mida de l'entrada)

2. Cost cas inductiu (n^{crides} , valors paràmetre recursiu crida, cost càlculs extra)

- Tipus de recursivitat \rightarrow Substractives $T(n) = a \cdot T(n-c) + O(n^k)$

\nwarrow n° trucades recursives
 \nearrow El cost de lo NO recursiu
 \searrow El que decreix

Divisors $T(n) = a \cdot T(n/b) + O(n^k)$

TEOREMA MESTRE SUSTRACTORES

$$T(n) = \begin{cases} \Theta(n^k) & a < 1 \\ \Theta(n^{k+1}) & a = 1 \\ \Theta(n^{k/c}) & a > 1 \end{cases}$$

TEOREMA MESTRE DIVISORES

$$\alpha = \log_b a$$

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

NOMBRES DE FIBONACCI

$$F(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ F(n-1) + F(n-2) & n \geq 2 \end{cases}$$

* Els nostres teoremes mestres NO serveixen per resoldre recurrències d'aquest tipus

$$F(n-1) \leq F(n) \leq F(n+1) \Rightarrow 2F(n-2) \leq F(n) \leq 2F(n-1) \Rightarrow F(n) = O(2^n) \wedge F(n) = \Omega(2^{n/2})$$

o Observació:

```
int fib(int k) {
    matrix f = {{1, 1}, {1, 0}};
    matrix p = pow(f, k);
    return p[1][0] + p[1][1];
}
```

$$\begin{pmatrix} F(2) \\ F(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F(1) \\ F(0) \end{pmatrix} = \begin{pmatrix} F(1) + F(0) \\ F(1) \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} F(n+1) \\ F(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F(1) \\ F(0) \end{pmatrix} \text{ per tot } k \geq 0.$$

Té cost $T(n) = \Theta(\log n)$

ALGORISMES DE SORTING

SELECTION SORT

```
template <typename elem>
void sel_sort (vector<elem>& v) {
    int n = v.size();
    for (int i = 0; i < n - 1; ++i) { O(n)
        int p = pos_min(v, i, n-1);
        swap(v[i], v[p]); ↘ O(n)
    }
    cost = O(n) · O(n) = O(n²)
}

template <typename elem>
int pos_min (vector<elem>& v, int l, int r) {
    int p = l;
    for (int j = l + 1; j <= r; ++j) { O(n)
        if (v[j] < v[p]) {
            p = j;
        }
    }
    return p;
}
```

 $O(n^2)$ operacions per ordenar n elements

Funcionament:

1. Busca el mínim element de la llista

2. L'intercanvia amb la primera posició

...

En general busca el mínim element entre "i" i "n-1" i intercanvia el mínim trobat amb el de la posició "i".

ex:

8 5 2 9 6 | 1 5 2 9 6 | 1 2 5 9 6 | 1 2 5 9 | 1 2 5 6 9

INSERTION SORT

```
template <typename elem>
void ins_sort_1 (vector<elem>& v) {
    int n = v.size();
    for (int i = 1; i < n; ++i) {
        for (int j = i; j > 0 and v[j - 1] > v[j]; --j) {
            swap(v[j - 1], v[j]);
        }
    }
}

void ins_sort_2 (vector<elem>& v) {
    int n = v.size(); * Evita swap
    for (int i = 1; i < n; ++i) { desplaçant els elem.
        elem x = v[i];
        int j;
        for (j = i; j > 0 and v[j - 1] > x; --j) {
            v[j] = v[j - 1];
        }
        v[j] = x;
    }
}

void ins_sort_3 (vector<elem>& v) {
    int n = v.size(); * situa l'elem
    swap(v[0], v[pos_min(v, 0, n-1)]); més petit al
    for (int i = 2; i < n; ++i) { principi
        elem x = v[i];
        int j;
        for (j = i; v[j - 1] > x; --j) {
            v[j] = v[j - 1];
        }
        v[j] = x;
    }
}
```

 $O(n^2)$ operacions per ordenar n elementsEn el millor dels casos $O(n)$.

Funcionament:

1. Comparem el segon element amb el primer, s'insereix si és necessari

2. Passem al tercer element i comparem amb els anteriors, s'insereix si és necessari

...

En general comparem l'element a la posició "i" fins "0" inserint l'element "i" on correspongui.

ex:

12 3 5 10 8 | 3 12 5 10 8 | 3 5 12 10 8 | 3 5 10 12 8 | 3 5 8 10 12

BUBBLE SORT

```
template <typename elem>
void bubble_sort (vector<elem>& v) {
    int n = v.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = n - 1; j > i; --j) {
            if (v[j - 1] > v[j]) {
                swap(v[j - 1], v[j]);
            }
        }
    }
}
```

 $O(n^2)$ operacions per ordenar n elements

Funcionament:

1. Compara el primer element amb el segon. Si es necessari fa un swap.

2. Compara el segon element amb el tercer. Si es necessari fa un swap.

...

En general es fan comparacions de l'element "i" i "i+1" fent swaps si es necessari fins que no calguin més passades.

ex:

6 5 3 1 8 | 5 6 3 1 8 | 5 3 6 1 8 | 5 3 1 6 8 | 5 3 1 6 8 | 3 5 1 6 8 | 3 1 5 6 8 | 3 1 5 6 8 | 1 3 5 6 8

MERGE SORT

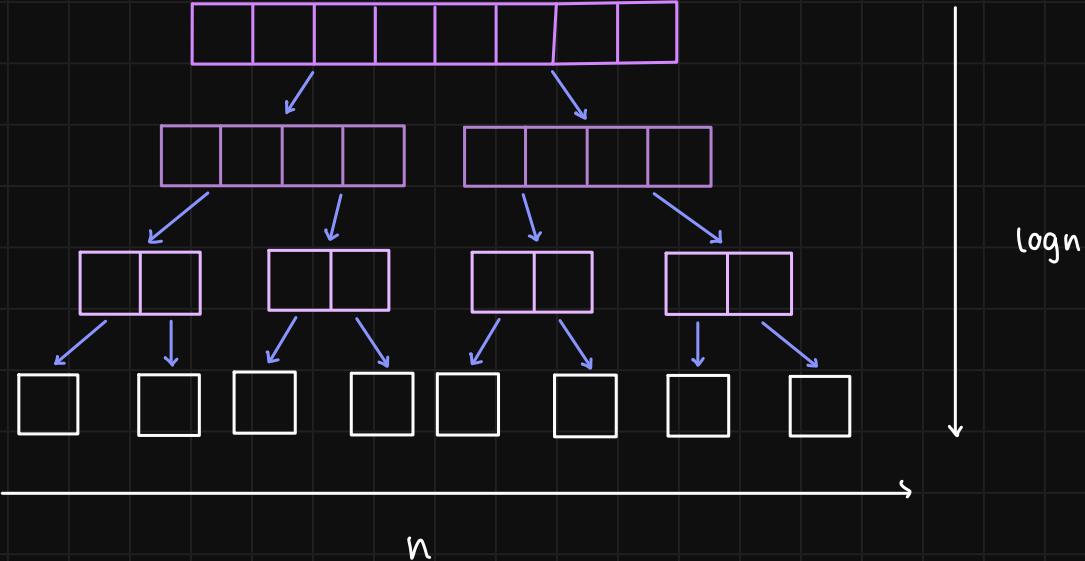
```

template <typename elem>
void merge_sort_1 (vector<elem>& v) {
    merge_sort_1(v, 0, v.size() - 1);
}
void merge_sort_1 (vector<elem>& v, int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        merge_sort_1(v, l, m);
        merge_sort_1(v, m + 1, r);
        merge(v, l, m, r);
    }
}
void merge (vector<elem>& v, int l, int m, int r) {
    vector<elem> b(r - l + 1);
    int i = l, j = m + 1, k = 0;
    while (i <= m and j <= r) {
        if (v[i] <= v[j]) b[k++] = v[i++];
        else b[k++] = v[j++];
    }
    while (i <= m) b[k++] = v[i++];
    while (j <= r) b[k++] = v[j++];
    for (k = 0; k <= r - l; ++k) v[l + k] = b[k];
}

template <typename elem>
void merge_sort_2 (vector<elem>& v) {
    merge_sort_2(v, 0, v.size() - 1);
}
void merge_sort_2 (vector<elem>& v, int l, int r) {
    const int critical_size = 50; * Para la
    if (r - l < critical_size) { recursivitat
        ins_sort(v, l, r); quan v és suficientment
    } else { petit. Llavors implementa
        int m = (l + r) / 2;
        merge_sort_2(v, l, m);
        merge_sort_2(v, m + 1, r);
        merge(v, l, m, r);
    }
}

template <typename elem>
void merge_sort_bu (vector<elem>& v) {
    int n = v.size();
    for (int m = 1; m < n; m *= 2) {
        for (int i = 0; i < n - m; i += 2*m) {
            merge(v, i, i + m - 1, min(i + 2 * m - 1, n - 1));
        }
    }
}

```

 $O(n \log n)$ operacions per ordenar n elements $\log n$ n

Funcionament:

1. Divideix una seqüència en n subseqüències d'un element
2. Va fusionant subseqüències ordenades fins a tenir la seqüència ordenada

Recursivitat:

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ 2T(n/2) + \Theta(n) & n \geq 2 \end{cases} \Rightarrow \alpha = 2, b = 2, \alpha = \log_2(2) = 1 \rightarrow K = 1$$

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

* Bottom-up (iteratiu):



•
•
•

T

1

2

3

4

5

6

7

8

9

10

11

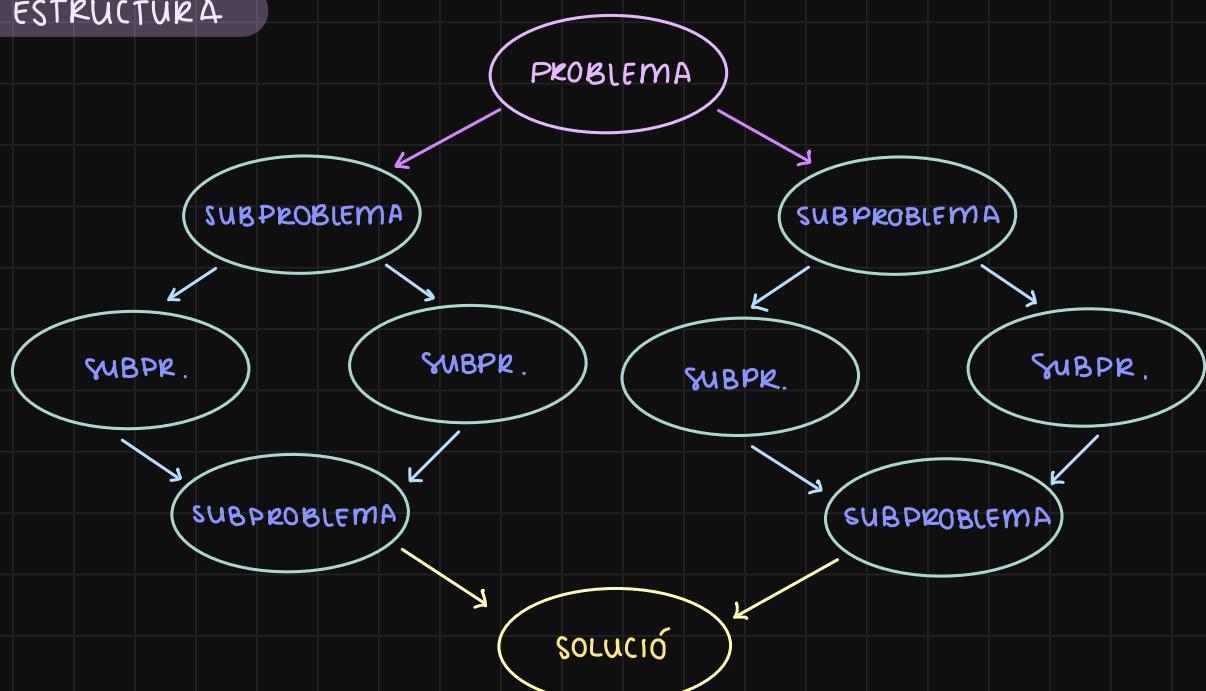
12

Section Two

Dividir i vencer

[BACK TO INDEX](#)

ESTRUCTURA



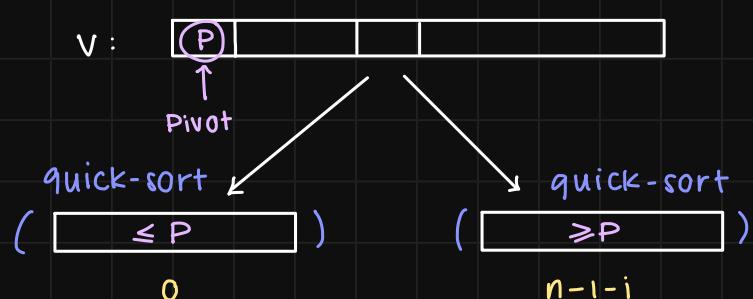
- Alguns algorismes que segueixen aquesta estructura són

→ merge - sort
→ Binary search
→ Quick exp.
...

QUICK-SORT

- Ordenació ràpida
- C.A.R. Hoare 1960

ESTRUCTURA



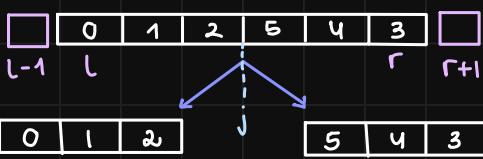
donat un vector T d'almenys 2 elements :

- Tria un element x de T
- Divideix T en dos grups disjunts T_1 i T_2
 - T_1 conté elements $\leq x$
 - T_2 conté elements $\geq x$
- Ordena T_1 i T_2 recursivament
- Retorna T_1 seguit de T_2

IMPLEMENTACIÓ

```

int partition ( vector<int> &v , int l , int r ) { // partició Hoare
    int q = v[l];
    int i = l-1;
    int j = r+1;
    for (;;) {
        while ( q > v[++i] );
        while ( q < v[--j] );
        if ( i >= j ) return j;
        swap ( v[i] , v[j] );
    }
}
void quicksort ( vector<int> &v ) {
    quicksort ( v, 0, v.size()-1 );
}
void quicksort ( vector<int> v , int l , int r ) {
    if ( l < r ) {
        int p = partition ( v, l, r );
        quicksort ( v, l, p );
        quicksort ( v, p+1, r );
    }
}
  
```

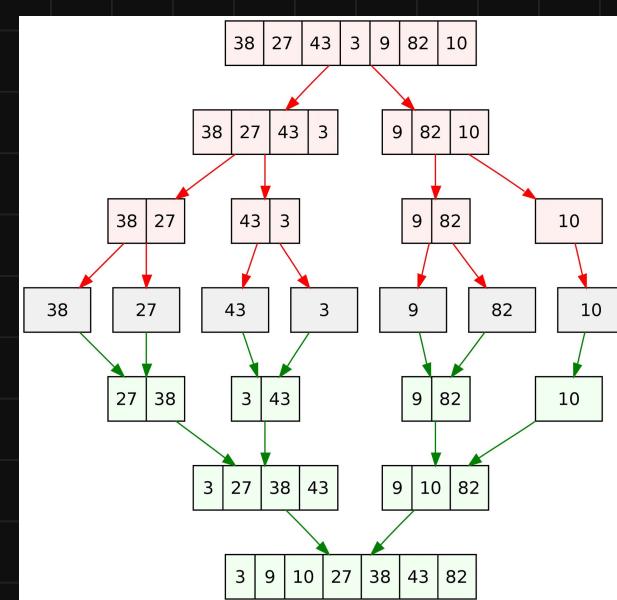


o Precondició " $p = \text{partition}(v, l, r)$ " :

$$0 \leq l \leq r \leq v.size() - 1$$

o Postcondició " $p = \text{partition}(v, l, r)$ " .

$$\exists x \forall i \text{ si } l \leq i \leq p \Rightarrow v[i] \leq x \\ \text{si } p < i \leq r \Rightarrow v[i] \geq x$$



COST QUICK-SORT

- o Cas millor (tenim un bon pivot)

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

$$\alpha = 2$$

$$b = 2 \Rightarrow \alpha = \log_b \alpha = \log_2 2 = 1 \Rightarrow \alpha = k \Rightarrow T(n) = n^k \log n = \Theta(n \log n)$$

$$k = 1$$

- o Cas pitjor (mal pivot, tenim un vector amb 1 elem. i l'altre amb $n-1$ elem.)

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

$$\alpha = 1 \Rightarrow \alpha = k \Rightarrow T(n) = n^{k+1} = \Theta(n^2)$$

$$k = 1$$

- o cas mitjà

$$T(n) = \sum_{\forall n \in E_n} P(n) T'(n)$$

Necessitem fer unes suposicions de les probabilitats

1. Elements diferents

2. Qualsevol permutació dels elements a ordenar són equiparables, això implica

3. que qualsevol element té la mateixa probabilitat $\frac{1}{n}$ d'apareixer a la primera posició (de ser el pivot) $\Rightarrow T(n) = \Theta(n \log n)$

4. Càlculs:

$$5. \begin{aligned} T(n) &= n+1 + \frac{1}{n} \sum_{i=0}^{n-1} T(i) + T(n-1-i) \\ &\quad \text{són els mateixos} \\ &\quad \text{comparacions per veure que } i \geq j \end{aligned}$$

$$6. \begin{aligned} nT(n) &= n(n+1) + 2 \sum_{i=0}^{n-1} T(i) \\ (n-1)T(n-1) &= n(n-1) + 2 \sum_{i=0}^{n-2} T(i) \end{aligned} \quad \left. \begin{aligned} nT(n) - (n-1)T(n-1) &= 2n + 2T(n-1) \Rightarrow \frac{T(n)}{n+1} = \frac{2}{n+1} + \frac{T(n-1)}{n} \\ &\quad \Theta(\log n) \end{aligned} \right.$$

$$7. \Rightarrow \frac{T(n)}{n+1} = \frac{2}{n+1} + \frac{2}{n} + \frac{T(n-2)}{n-1} \Rightarrow \frac{T(n)}{n+1} = 2 \left(\frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{2} \right) \Rightarrow T(n) = \Theta(n \log n)$$

BINARY-SEARCH

- o Busca un element x en un vector ordenat creixentment

ESTRUCTURA

- o Divideix V en dos subvectors $V[i \dots m-1]$ i $V[m \dots j]$ on $m = (i+j)/2$
- o x es busca recursivament en $V[i \dots m-1]$ si $V[m] > x$ o a $V[m \dots j]$ si $V[m] < x$

IMPLEMENTACIÓ

```
int bsearch (const vector<int> &V, int x, int i, int j) {
    if (i > j) return -1;
    int m = (i+j)/2;
    if (V[m] == x) return m;
    else if (x < V[m]) return bsearch (V, x, i, m-1);
    else return bsearch (V, x, m+1, j);
}
```

RECURRÈNCIA

$$T_{bs}(n) = \begin{cases} \Theta(1) & \text{si } n=2 \\ T_{bs}\left(\frac{n}{2}\right) + \Theta(1) & \text{si } n > 2 \end{cases}$$

$$T_{bs}(n) = \Theta(\log n)$$

EXPONENTIACIÓ RÀPIDA

- o Calcula x^n

ESTRUCTURA

- o Mira la paritat de n i divideix en dos casos
- o Cas parell: $x^n = x^{n/2} \cdot x^{n/2}$
- o Cas imparell: $x^n = x^{n/2} \cdot x^{n/2} \cdot x$

IMPLEMENTACIÓ

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

RECURRÈNCIA

$$T(n) = \begin{cases} \Theta(1) & \text{si } n=1 \\ T\left(\frac{n}{2}\right) + \Theta(1) & \text{si } n>0 \end{cases}$$

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

KARATSUBA

- o Calcula la multiplicació de 2 números molt grans

ESTRUCTURA

- o Divideix els números de n xifres en dues mitades

$$x = a \cdot 10^{n/2} + b \Rightarrow x = \boxed{a \quad b}$$

$$y = c \cdot 10^{n/2} + d \Rightarrow y = \boxed{c \quad d}$$

- o Fa les multiplicacions dels números de $n/2$ xifres

- o combina els resultats, no resol amb 3 multiplicacions i fent 3 trucades recursives amb la mitad de díigits.

$$xy = ((a+b)(c+d) - ac - bd) \cdot 10^{n/2} + ac \cdot 10^n + bd$$

$$\rightarrow \text{amb } n \text{ bits : } xy = ((a+b)(c+d) - ac - bd) \cdot 2^{n/2} + ac \cdot 2^n + bd$$

RECURRÈNCIA

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

$$T_k(n) = \Theta(n^{\log_2 3})$$

STRASSEN

- o multiplica dues matrius de $n \times n$

ESTRUCTURA

- o Divideix les matrius x, y en 4 quadrants $x = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ $y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$

- o Resol el problema fent 7 multiplicacions de matrius de $n/2 \times n/2$

$$P_1 = A(F-H) ; P_2 = (A+B)H ; P_3 = (C+D)E ; P_4 = D(G-E) ; P_5 = (A+D)(E+H) ; P_6 = (B-D)(G+H) ; P_7 = (A-C)(E+F)$$

- o Redefinim xy

$$xy = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

RECURRÈNCIA

$$T_s(n) = \begin{cases} \Theta(1) & \text{si } n \leq 1 \\ 7T_s(\frac{n}{2}) + \Theta(n^2) & \text{si } n > 1 \end{cases}$$

$$T_s(n) = \Theta(n^{\log_2 7})$$

•
•
•

T

1

2

3

4

5

6

7

8

9

10

11

12

Section Three

Diccionaris

[BACK TO INDEX](#)

DICCIÓNARIS

- o Es caracteritzen per ser parells de <key, valor> on la key és única
- o Tenen les següents operacions:
 - 1) Asignar afageix un nou elem <clau, info>, si ja existeix la clau reescriu la info d'aquesta
 - 2) Borrar borra un elem, si no existeix no fa res
 - 3) consultar donada una clau retorna un bool que diu si està, donada una info retorna una referència a l'elem amb aquesta info, donada una clau retorna una referència a la info.
 - 4) Tamany retorna la mida del diccionari

TAULES D'ACCÉS DIRECTE

- o Si sabem el tam. màx d'un diccionari podem fer una implementació de taula amb tam. fixe.
 - o Cada posició correspon a una clau (suposant claus d'ints)
 - o El val. d'una posició es la info associada
 - o El cost de les op. es constant en el cas pitjor

TAULES DE HASH

- o S'utilitzen quan el num de claus utilitzades es petit amb el nombre possible de claus
 - o Es molt gran.
 - o Útil per cerques aleatòries d'elem
 - o Funciona transformant la clau amb la funció hash en un número que la taula utilitza per localitzar el valor desitjat.
- cost cas pitjor $\Theta(n)$ cost cas millor $\Theta(1)$

$$\text{HASH} : K \rightarrow \{0, 1, 2, \dots, m-2, m-1\}$$

↑ conjunt claus ↓ mida

- o A les taules de HASH es poden produir colissons (dos claus assignades a la mateixa pos) que es solucionen amb encadenaments

TAULES DE HASH AMB LÍSTES ENCADENADES

- o En cas de colisió es crea una llista encadenada associada a esa posició
- o Funció de hash($clau_k$) retorna posició $\Theta(1) \Rightarrow k \bmod m$
- o Op. crear reserva un espai per la taula de m màx $\Theta(1)$
- o Les op. buscar, insertar, borrar en cas pitjor tenen cost $\Theta(n)$. En cas mitjà tenim $\Theta(\alpha)$ on $\alpha = n/m$. En cas millor $\Theta(1)$.

0	1	2	3	4	5	6	7
1	2	12	13	6	7		

| |
10 15

$$m = 8 \quad K = \{7, 6, 2, 10, 15, 1, 13, 12\} \quad \alpha = \frac{8}{8} = 1$$

TAULES DE HASH AMB DIRECCIONAMENT OBERT

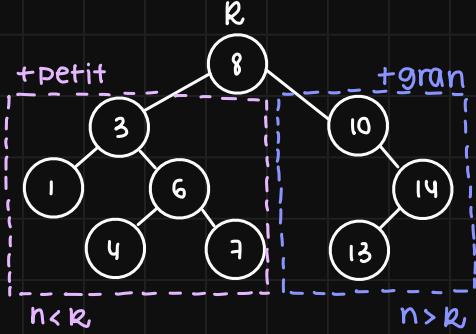
- o En cas de colisió l'element a insertar es guarda a la següent posició lliure de la taula
- o Op. crear reserva un espai per la taula de m màx $\Theta(1)$
- o Les op. buscar, insertar, borrar en cas pitjor tenen cost $\Theta(n)$. En cas mitjà tenim $\Theta(\alpha)$ on $\alpha = n/m$. En cas millor $\Theta(1)$.

15	1	2	10	12	13	6	7

$$m = 8 \quad K = \{7, 6, 2, 10, 15, 1, 13, 12\} \quad \alpha = \frac{8}{8} = 1$$

ARBRE BINARI DE SERCA

(ABB, ABS, BST)



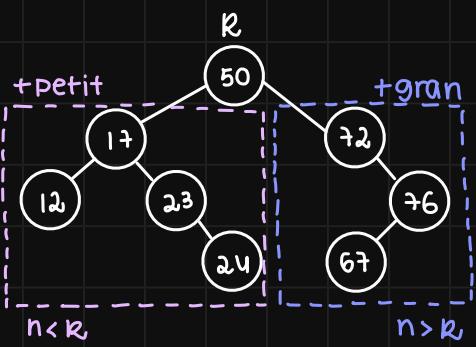
- o Un ABB es una manera d'implementar un diccionari
- o Características:
 - o Els sub. arbres esq. i dre han de ser ABB.
 - o R és més gran que totes les claus del sub. arbre esq i més petita que totes les claus del sub. arbre dre.

Operacions:

- o crear, crea un ABB buit $\Theta(1)$
- o Buscar, segons si $x >$ clau o $x <$ clau busquem pel fill esq. o dre. fins trobar x o un node nul
- o Insertar, segons si $x >$ clau o $x <$ clau insertem pel fill esq. o dre. fins trobar un node nul on insertar x.
- o Borrar un element T :
 - o Si T és fulla: s'elimina T
 - o Si T té un fill: s'elimina T i el fill puja
 - o Si T té dos fills: s'elimina T i pujem el fill més petit de l'arbre dre. o el més gran del fill esq.
- o Cost de buscar, insertar, borrar \rightarrow cas pitjor $\Theta(n)$ \rightarrow si ABB és una uista
 \rightarrow cas mitjà $\Theta(h)$ depen de l'altura de l'arbre
 \rightarrow cas millor $\Theta(1)$

ARBRE BINARI DE SERCA EQUILIBRADA

(AVL)



- o Un AVL és un ABB 100% equilibrat (la dif. d'altura entre el fill esq. i dre. es 0 o 1)

Operacions:

- o crear, crea un ABB buit $\Theta(1)$
- o Buscar, segons si $x >$ clau o $x <$ clau busquem pel fill esq. o dre. fins trobar x o un node nul
- o Insertar, segons si $x >$ clau o $x <$ clau insertem pel fill esq. o dre. fins trobar un node nul on insertar x.
- o Borrar un element T :
 - o Si T és fulla: s'elimina T. Es rota si es necessari, en cas de desequilibri.
 - o Si T té un fill: s'elimina T i el fill puja. Es rota si es necessari, en cas de desequilibri.
 - o Si T té dos fills: s'elimina T i pujem el fill més petit de l'arbre dre. o el més gran del fill esq. Es rota si es necessari, en cas de desequilibri.
- o Cost de buscar, insertar, borrar \rightarrow cas pitjor i mitjà $\Theta(\log n)$
 \rightarrow cas millor $\Theta(1)$

Rotacions en cas de desequilibri

ROTACIÓ SIMPLE

- o Esq - Esq .
- o Dre - Dre .

ROTACIÓ DOBLE

- o Dre - Esq .
- o Esq - Dre .

•
•
•

T

1

2

3

4

5

6

7

8

9

10

11

12

Section Eight

Problemas

[BACK TO INDEX](#)

$$1. \text{ a) } \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n-1 + n$$

$$\sum_{i=n}^1 i = n + n-1 + n-2 + \dots + 2 + 1$$

$$\text{b) } \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$2i = (n+1) + (n+1) + \dots + (n+1) = n(n+1) \Rightarrow \frac{2i}{2} = \frac{n(n+1)}{2} \Rightarrow \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\text{Cas base } n=1 \Rightarrow \sum_{i=1}^1 1 = \frac{1(2)(3)}{6} = 1 \text{ cert.}$$

Pas inductiu

$$\sum_{i=1}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6} = \frac{n^2 - n \cdot (2n-1)}{6} = \frac{2n^3 - n^2 - 2n^2 + n}{6} = \frac{2n^3 - 3n^2 + n}{6}$$

$$\sum_{i=1}^{n-1} i^2 + n^2 = \sum_{i=1}^n i^2 \Rightarrow \frac{2n^3 - 3n^2 + n}{6} + n^2 = \frac{2n^3 + 3n^2 + n}{6} = \frac{n(n+1)(2n+1)}{6} = \sum_{i=1}^n i^2$$

$$c) \sum_{i=0}^n 2^i = 2^{n+1} - 1$$

$$\text{Cas base } n=0 \Rightarrow \sum_{i=0}^0 1 = 2^0 - 1 = 1$$

Cas inductiu

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

$$\sum_{i=0}^{n-1} 2^i + 2^n = \sum_{i=0}^n 2^i \Rightarrow 2^n - 1 + 2^n = 2 \cdot 2^n - 1 = 2^{n+1} - 1 = \sum_{i=0}^n 2^i$$

$$7. \text{ a) } p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 n^0 \text{ i } a_k > 0 \text{ es } \theta(n^k)$$

$$\lim_{n \rightarrow \infty} \frac{p(n)}{n^k} = c \text{ i } c > 0?$$

$$\lim_{n \rightarrow \infty} \frac{a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 n^0}{n^k} = \lim_{n \rightarrow \infty} a_k + \underbrace{\frac{a_{k-1}}{n} + \dots + \frac{a_0}{n^k}}_{\text{per } n \rightarrow \infty \text{ tendeix a } 0} = a_k \text{ amb } a_k > 0.$$

$$b) a, b > 1 \Rightarrow \log_a n \text{ es } \theta(\log_b n)$$

$$\log_a n = \log_a (b^{\log_b n}) \Rightarrow \log_a n = \frac{\log_b n}{\log_a b} \Rightarrow \log_b n = \log_a n \cdot \underbrace{\log_a b}_{c>0}$$

$$\Rightarrow \log_b n = \log_a n$$

$$8. \frac{n}{\ln n} = O(n/\log n)$$

$$\log_n(n^n) = n \cdot \log_n(n) = O(n \log n)$$

$$4n\sqrt{n} = O(n\sqrt{n})$$

$$\frac{n}{\log_2 n} = O(n/\log n)$$

$$3 \ln(7^n) = 3n \ln 7 = O(n)$$

$$5n \cdot \ln n = O(n \log n)$$

12. //Fragment 1

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

$$\Theta(1) \rightarrow T_{\text{for}}(n) = \sum_{i=0}^n \Theta(1) = \Theta\left(\sum_{i=0}^n 1\right) = \Theta(n)$$

// Fragment 2

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

$$\Theta(1) \rightarrow T_{\text{for}}(n) = \sum_{i=0}^n \Theta(1) = \Theta\left(\sum_{i=0}^n 1\right) = \Theta\left(\frac{n}{2}\right) = \Theta(n)$$

// Fragment 3

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

$$\begin{aligned} \Theta(1) \rightarrow T_{\text{for}}(n) &= \sum_{i=0}^n \Theta(1) = \Theta\left(\sum_{i=0}^n 1\right) = \Theta(n) \\ \Theta(1) \rightarrow T_{\text{for}}(n) &= \sum_{i=0}^n \Theta(1) = \Theta\left(\sum_{i=0}^n 1\right) = \Theta(n) \end{aligned} \quad \left. \begin{array}{l} \Theta(n) + \Theta(n) \rightarrow \Theta(n) \end{array} \right\}$$

//Fragment 4

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

$$\Theta(1) \rightarrow T_{\text{for}}(n) = \sum_{i=0}^n \Theta\left(\sum_{j=0}^n 1\right) = \sum_{i=0}^n \Theta(\Theta(n)) = \Theta\left(\sum_{i=0}^n \Theta(n)\right) = \Theta(n^2)$$

// Fragment 5

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

$$\Theta(1) \rightarrow T_{\text{for}}(n) = \sum_{i=0}^n \Theta\left(\sum_{j=0}^i 1\right) = \sum_{i=0}^n \Theta(i) = \Theta\left(\sum_{i=0}^n i\right) = \Theta(n^2)$$

//Fragment 6

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

$$\Theta(1) \rightarrow T_{\text{for}}(n) = \sum_{i=0}^n \Theta\left(\sum_{j=i}^n 1\right) = \sum_{i=0}^n \Theta(n-i) = \Theta\left(\sum_{i=0}^n n\right) = \Theta(n^2)$$

//Fragment 7

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

$$\Theta(1) \rightarrow T_{\text{for}}(n) = \sum_{i=0}^n \Theta\left(\sum_{j=0}^n \Theta\left(\sum_{k=0}^n 1\right)\right) = \sum_{i=0}^n \Theta\left(\sum_{j=0}^n \Theta(n)\right) = \sum_{i=0}^n \Theta(n^3) = \Theta(n^3)$$

// Fragment 8

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

$$\begin{aligned} \Theta(1) \\ \rightarrow T_{\text{for}}(n) &= \sum_{i=0}^n \theta\left(\sum_{j=0}^i \theta\left(\sum_{k=0}^j \theta(1)\right)\right) = \sum_{i=0}^n \theta\left(\sum_{j=0}^i \theta(j)\right) = \sum_{i=0}^n \theta\left(\theta\sum_{j=0}^i (j-i)\right) = \sum_{i=0}^n \theta(i^2) \\ \theta(1) &= \sum_{i=0}^n \theta(i^2) = \theta\left(\sum_{i=0}^n i^2\right) = \theta(n^3) \end{aligned}$$

// Fragment 9

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

$$\begin{aligned} \theta(1) \\ \rightarrow T_{\text{for}}(n) &= \sum_{i=1}^n \theta(1) = \theta\sum_{i=1}^n 1 = \theta(\log n) \\ \theta(1) \end{aligned}$$

// Fragment 10

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

$$\begin{aligned} \theta(1) \\ \rightarrow T_{\text{for}}(n) &= \sum_{i=0}^n \theta\left(\sum_{j=0}^{i^2} \theta\left(\sum_{k=0}^n \theta(1)\right)\right) = \sum_{i=0}^n \theta\left(\sum_{j=0}^{i^2} \theta(n)\right) = \sum_{i=0}^n \theta(n \cdot i^2) = \\ \theta(1) &= \theta\left(n \sum_{i=0}^n i^2\right) = \theta(n^4) \end{aligned}$$

// Fragment 11

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

$$\begin{aligned} \theta(1) \\ \rightarrow T_{\text{for}}(n) &= \sum_{i=0}^n \left(\sum_{j=0}^{i^2} \theta(1) + \sum_{j=0}^i \theta(n) \right) = \sum_{i=0}^n \theta(i^2 + i \cdot n) = \\ \theta(1) &= \theta\left(\sum_{i=0}^n i^2 + n \sum_{i=0}^n i\right) = \theta(n^3) + \theta(n^3) = \theta(n^3) \end{aligned}$$