

Tema 2. Dividir i vèncer

Estructures de Dades i Algorismes

FIB

Antoni Lozano

Q2 2017–2018

Versió de 19 de març de 2018

1 Ordenació per fusió

- Algorisme de fusió bàsic
- Variants

2 Ordenació ràpida

- Algorisme general
- Variants
- Anàlisi

3 Productes i exponents

- Algorisme de Karatsuba
- Exponenciació ràpida
- Algorisme de Strassen

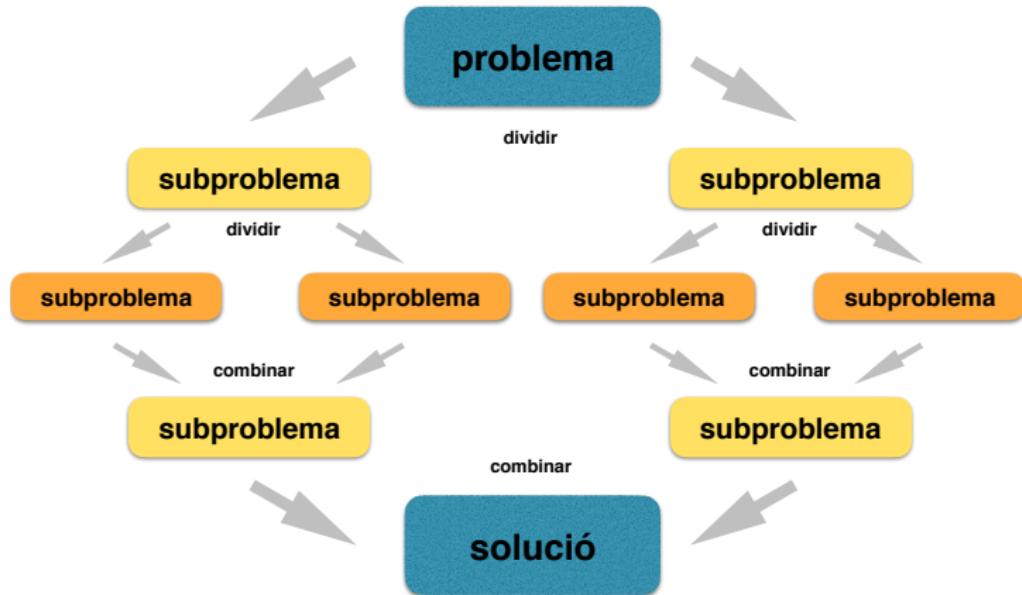
4 Altres algorismes

- Torres de Hanoi
- Mediana

L'estratègia **dividir i vèncer** resol un problema en tres passos:

- ① **dividint-lo** en *subproblems*, exemples més petits del mateix problema
- ② **resolent** els subproblems recursivament
- ③ **combinant** les solucions de manera adequada

Dividir i vèncer



La feina es fa, per tant, en tres parts: (1) en la divisió en subproblems, (2) al final de la recursió i (3) en la combinació de les solucions.

Els algorismes *dividir i vèncer* segueixen sovint una estratègia uniforme.
Ataquen un problema de mida n

- dividint-lo en a subproblemes de mida n/b i
- combinant les respostes en temps n^k ,

on a, b, k són constants naturals. En aquest cas, el seu cost es pot descriure mitjançant la recurrència

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

que es pot resoldre aplicant el **teorema mestre II**.

Cerca dicotòmica

```
int cerca_dicotomica(const vector<int>& a,
                      int i, int j, int v) {
    // retorna la pos. de v entre a[i] i a[j] si hi es; -1 si no
    if (i <= j) {
        int k = (i + j) / 2;
        if (v == a[k])
            return k;
        else if (v < a[k])
            return cerca_dicotomica(a, i, k-1, v);
        else
            return cerca_dicotomica(a, k+1, j, v);
    } return -1;
}
```

El paràmetre de recursió és $n = j - i + 1$ i el cost $T(n) = T(n/2) + \Theta(1)$.
Pel teorema mestre II, $T(n) \in \Theta(\log n)$.

Exercici: Rang

Escriviu un algorisme de cost $\Theta(\log n)$ i basat en *dividir i vèncer* que, donada una taula ordenada T amb n elements diferents i dos elements x i y amb $x \leq y$, retorni el nombre d'elements en T que es troben entre x i y (x i y inclosos).

Tema 2. Dividir i vèncer

1 Ordenació per fusió

- Algorisme de fusió bàsic
- Variants

2 Ordenació ràpida

- Algorisme general
- Variants
- Anàlisi

3 Productes i exponents

- Algorisme de Karatsuba
- Exponenciació ràpida
- Algorisme de Strassen

4 Altres algorismes

- Torres de Hanoi
- Mediana

Algorisme de fusió bàsic

L'**ordenació per fusió**, o *mergesort*, és un bon exemple de l'esquema **dividir i vèncer** que fa servir un nombre de comparacions gairebé òptim. És un algorisme estable en un doble sentit:

- preserva l'ordre entre valors iguals
- es comporta de manera semblant amb independència del grau d'ordenació de l'entrada

Mergesort va ser inventat per John von Neumann l'any 1945.



Donat un vector T de talla ≥ 2 , l'algorisme consisteix a:

- ① Ordenar recursivament la primera meitat de T
- ② Ordenar recursivament la segona meitat de T
- ③ Retornar la fusió de les dues meitats

L'operació principal (punt 3) s'encarrega de **fusionar** dos vectors ordenats en un.

Algorisme de fusió bàsic

Exemple de fusió

entrada	E	X	E	M	P	L	E	F	U	S	I	O
ordenar 1a meitat	E	E	L	M	P	X	E	F	U	S	I	O
ordenar 2a meitat	E	E	L	M	P	X	E	F	I	O	S	U
resultat fusió	E	E	E	F	I	L	M	O	P	S	U	X

Algorisme de fusió bàsic

Exemple de fusió

entrada	E	X	E	M	P	L	E	F	U	S	I	O
ordenar 1a meitat	E	E	L	M	P	X	E	F	U	S	I	O
ordenar 2a meitat	E	E	L	M	P	X	E	F	I	O	S	U
resultat fusió	E	E	E	F	I	L	M	O	P	S	U	X

Exemple de fusió

entrada	E	X	E	M	P	L	E	F	U	S	I	O
ordenar 1a meitat	E	E	L	M	P	X	E	F	U	S	I	O
ordenar 2a meitat	E	E	L	M	P	X	E	F	I	O	S	U
resultat fusió	E	E	E	F	I	L	M	O	P	S	U	X

Algorisme de fusió bàsic

Exemple de fusió

entrada	E	X	E	M	P	L	E	F	U	S	I	O
ordenar 1a meitat	E	E	L	M	P	X	E	F	U	S	I	O
ordenar 2a meitat	E	E	L	M	P	X	E	F	I	O	S	U
resultat fusió	E	E	E	F	I	L	M	O	P	S	U	X

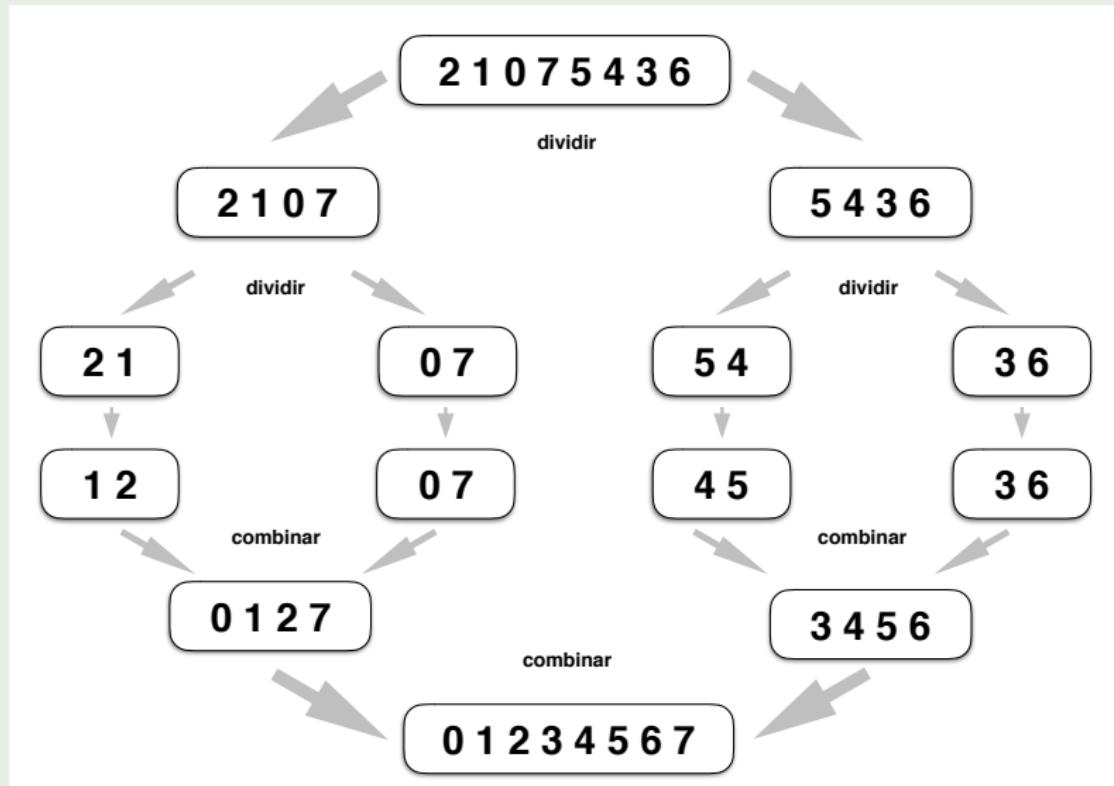
Ordenació per fusió (*Algoritmes en C++, EDA*)

```
template <typename elem>
void mergesort (vector<elem>& T) {
    mergesort(T, 0, T.size() - 1);
}

template <typename elem>
void mergesort (vector<elem>& T, int e, int d) {
    if (e < d) {
        int m = (e + d) / 2;
        mergesort(T, e, m);
        mergesort(T, m + 1, d);
        merge(T, e, m, d);
    }
}
```

Algorisme de fusió bàsic

Exemple (aquí, la recursió acaba per a talla 2, en l'algorisme és per a 1)



Algorisme de fusió bàsic

El cor de l'algorisme rau en fer la fusió de dos vectors ordenats.

Fusió (*Algorismes en C++, EDA*)

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d)
// donats els subvectors ordenats T[e..m] i T[m+1..d],
// deixa T[e..d] ordenat
{
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

0

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

0	1
---	---

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

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

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

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

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

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

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

0	1	2	3	4	5
---	---	---	---	---	---

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

0	1	2	3	4	5	6
---	---	---	---	---	---	---

Algorisme de fusió bàsic

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d-e; ++k) T[e + k] = B[k];
}
```

Exemple

0	1	2	7
---	---	---	---

3	4	5	6
---	---	---	---

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
template <typename elem>
void merge (vector<elem>& T, int e, int m, int d) {
    vector<elem> B(d - e + 1);
    int i = e, j = m + 1, k = 0;
    while (i <= m and j <= d) {
        if (T[i] <= T[j]) B[k++] = T[i++];
        else B[k++] = T[j++];
    }
    while (i <= m) B[k++] = T[i++];
    while (j <= d) B[k++] = T[j++];
    for (k = 0; k <= d - e; ++k) T[e + k] = B[k];
}
```

Observació

Cada comparació afegeix un element a la taula B excepte l'última, que n'afegeix almenys dos.

- Per tant, el nombre de **comparacions** és $< N = d - e + 1$
- El nombre d'**assignacions** és $2N$
- El cost és **lineal**

Algorisme de fusió bàsic

```
template <typename elem>
void mergesort (vector<elem>& T, int e, int d) {
    if (e < d) {
        int m = (e + d) / 2;
        mergesort (T, e, m);
        mergesort (T, m + 1, d);
        merge (T, e, m, d);
    }
}
```

Donat que el procediment `merge` és lineal, el cost de l'ordenació per fusió es pot expressar fàcilment amb la recurrència

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

i, aplicant el teorema mestre II, podem dir que

$$T(n) \in \Theta(n \log n).$$

Variants

Ordenació per fusió amb inserció per a vectors petits (*Alg. en C++, EDA*)

```
template <typename elem>
void mergesort (vector<elem>& T, int e, int d) {
    const int talla_critica = 50;
    if (d - e < talla_critica)
        ordena_insercio(T, e, d);
    else {
        int m = (e + d) / 2;
        mergesort(T, e, m);
        mergesort(T, m + 1, d);
        merge(T, e, m, d);
    }
}
```

Les dues **versions iteratives** que veurem parteixen del fet que les fusions només comencen al final de la recursió, de manera que:

- comencen directament pels elements a ordenar
- arriben al vector ordenat mitjançant fusions

Ordenació per fusió iterativa 1

Versió del llibre *Algorithms* de Dasgupta/Papadimitriou/Vazirani en pseudocodi (pàg. 51). Es fa servir el TAD cua amb operacions

- **inject (Q, e)**: afegir l'element e a la cua Q i
- **eject (Q)**: funció que treu i retorna l'últim element de Q

```
function mergesort_queue(a[1...n])
    Q = [] (cua buida)
    for i=1 to n:
        inject(Q, a[i])
    while |Q| > 1:
        inject(Q, merge(eject(Q), eject(Q)))
    return eject(Q)
```



Ordenació per fusió iterativa 1

Versió del llibre *Algorithms* de Dasgupta/Papadimitriou/Vazirani en pseudocodi (pàg. 51). Es fa servir el TAD cua amb operacions

- `inject(Q, e)`: afegir l'element `e` a la cua `Q` i
- `eject(Q)`: funció que treu i retorna l'últim element de `Q`

```
function mergesort_queue(a[1...n])
    Q = [] (cua buida)
    for i=1 to n:
        inject(Q, a[i])
    while |Q| > 1:
        inject(Q, merge(eject(Q), eject(Q)))
    return eject(Q)
```

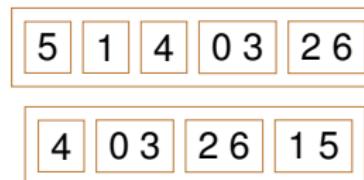


Ordenació per fusió iterativa 1

Versió del llibre *Algorithms* de Dasgupta/Papadimitriou/Vazirani en pseudocodi (pàg. 51). Es fa servir el TAD cua amb operacions

- `inject(Q, e)`: afegir l'element `e` a la cua `Q` i
- `eject(Q)`: funció que treu i retorna l'últim element de `Q`

```
function mergesort_queue(a[1...n])
    Q = [] (cua buida)
    for i=1 to n:
        inject(Q, a[i])
    while |Q| > 1:
        inject(Q, merge(eject(Q), eject(Q)))
    return eject(Q)
```

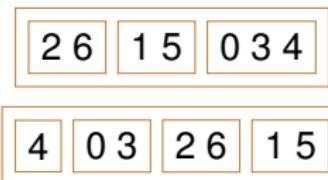


Ordenació per fusió iterativa 1

Versió del llibre *Algorithms* de Dasgupta/Papadimitriou/Vazirani en pseudocodi (pàg. 51). Es fa servir el TAD cua amb operacions

- **inject (Q, e)**: afegir l'element e a la cua Q i
- **eject (Q)**: funció que treu i retorna l'últim element de Q

```
function mergesort_queue(a[1...n])
    Q = [] (cua buida)
    for i=1 to n:
        inject(Q, a[i])
    while |Q| > 1:
        inject(Q, merge(eject(Q), eject(Q)))
    return eject(Q)
```

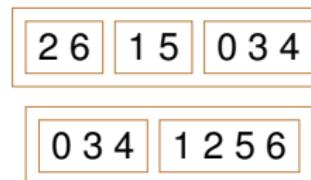


Ordenació per fusió iterativa 1

Versió del llibre *Algorithms* de Dasgupta/Papadimitriou/Vazirani en pseudocodi (pàg. 51). Es fa servir el TAD cua amb operacions

- `inject(Q, e)`: afegir l'element `e` a la cua `Q` i
- `eject(Q)`: funció que treu i retorna l'últim element de `Q`

```
function mergesort_queue(a[1...n])
    Q = [] (cua buida)
    for i=1 to n:
        inject(Q, a[i])
    while |Q| > 1:
        inject(Q, merge(eject(Q), eject(Q)))
    return eject(Q)
```



Ordenació per fusió iterativa 1

Versió del llibre *Algorithms* de Dasgupta/Papadimitriou/Vazirani en pseudocodi (pàg. 51). Es fa servir el TAD cua amb operacions

- `inject(Q, e)`: afegir l'element `e` a la cua `Q` i
- `eject(Q)`: funció que treu i retorna l'últim element de `Q`

```
function mergesort_queue(a[1...n])
    Q = [] (cua buida)
    for i=1 to n:
        inject(Q, a[i])
    while |Q| > 1:
        inject(Q, merge(eject(Q), eject(Q)))
    return eject(Q)
```

0 1 2 3 4 5 6

0 3 4 1 2 5 6

Ordenació per fusió iterativa 2 (Algorismes en C++, EDA)

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



Ordenació per fusió iterativa 2 (Algorismes en C++, EDA)

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



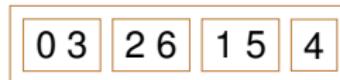
Ordenació per fusió iterativa 2 (Algorismes en C++, EDA)

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



Ordenació per fusió iterativa 2 (Algorismes en C++, EDA)

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



Ordenació per fusió iterativa 2 (Algorismes en C++, EDA)

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



Ordenació per fusió iterativa 2 (Algorismes en C++, EDA)

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

0 1 2 3 4 5 6

0 2 3 6 1 4 5

Exercici: **Fusió amunt, fusió avall**

Ordeneu la taula $\langle 3, 8, 15, 7, 12, 6, 5, 4, 3, 7, 1 \rangle$ amb l'algorisme d'ordenació per fusió recursiu i amb l'algorisme d'ordenació per fusió d'avall cap amunt (fusió iterativa 2).

Exercicis

Recordem la versió recursiva de *mergesort*

```
template <typename elem>
void mergesort (vector<elem>& T) {
    mergesort(T, 0, T.size() - 1);
}

template <typename elem>
void mergesort (vector<elem>& T, int e, int d) {
    if (e < d) {
        int m = (e + d) / 2;
        mergesort(T, e, m);
        mergesort(T, m + 1, d);
        merge(T, e, m, d);
    }
}
```

Exercici: Alcària de pila

Quantes crides recursives cal guardar com a màxim en un instant donat a la pila per ordenar per fusió una taula de n elements?

Exercicis

Recordem la versió recursiva de *mergesort*

```
template <typename elem>
void mergesort (vector<elem>& T) {
    mergesort(T, 0, T.size() - 1);
}

template <typename elem>
void mergesort (vector<elem>& T, int e, int d) {
    if (e < d) {
        int m = (e + d) / 2;
        mergesort(T, e, m);
        mergesort(T, m + 1, d);
        merge(T, e, m, d);
    }
}
```

Exercici: Nombre de vectors

Cada crida a `merge` crea un vector auxiliar. Fins a quants n'hi pot haver en un moment donat per a una taula de n elements?

Tema 2. Dividir i vèncer

1 Ordenació per fusió

- Algorisme de fusió bàsic
- Variants

2 Ordenació ràpida

- Algorisme general
- Variants
- Anàlisi

3 Productes i exponents

- Algorisme de Karatsuba
- Exponenciació ràpida
- Algorisme de Strassen

4 Altres algorismes

- Torres de Hanoi
- Mediana

Algorisme general

Com el seu nom indica, l'**ordenació ràpida**, o **quicksort**, és l'algorisme d'ordenació genèric més ràpid. Tot i que el seu cost en el cas pitjor és $\Theta(n^2)$, el cas mitjà és $\Theta(n \log n)$ i l'eficiència del seu bucle intern el converteix en l'algorisme que es comporta millor a la pràctica.

Quicksort va ser inventat per C. A. R. Hoare l'any 1960. Ha estat molt estudiat des de llavors.

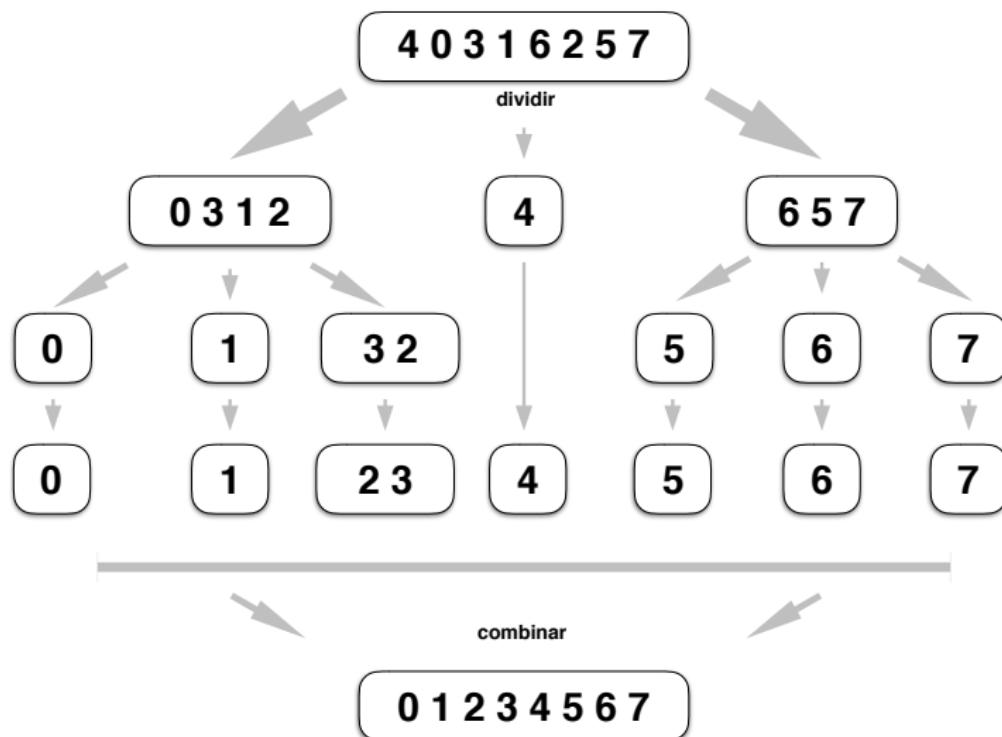


Donat un vector T d'almenys 2 elements, l'**algorisme bàsic** fa el següent:

- 1 Tria un element x de T
- 2 Dividir $T - \{x\}$ en dos grups disjunts:
 - T_1 , que conté elements $\leq x$ de T
 - T_2 , que conté elements $\geq x$ de T
- 3 Ordenar T_1 i T_2 recursivament
- 4 Retornar T_1 , seguit de x , seguit de T_2

Algorisme general

Exemple



Algorisme general

Ordenació ràpida (*Algoritmes en C++, EDA*)

```
void quicksort(vector<elem>& T) {
    quicksort(T, 0, T.size() - 1);
}

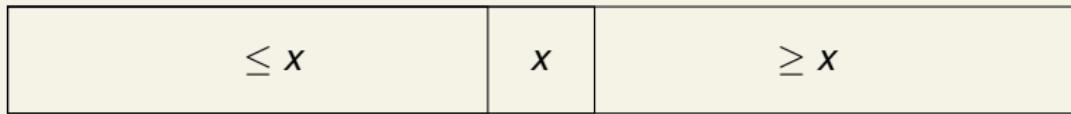
template <typename elem>
void quicksort(vector<elem>& T, int e, int d) {
    if (e < d) {
        int q = partition(T, e, d);
        quicksort(T, e, q);
        quicksort(T, q + 1, d);
    }
}
```

- Precondició de $q = \text{partition}(T, e, d)$: $0 \leq e \leq d \leq T.size() - 1$
- Postcondició de $q = \text{partition}(T, e, d)$: per a tot natural i
 - si $e \leq i \leq q$, tenim que $T[i] \leq T[q]$
 - si $q \leq i \leq d$, tenim que $T[i] \geq T[q]$

Algorisme general

Ordenació ràpida (*Algorismes en C++, EDA*)

```
void quicksort(vector<elem>& T) {  
    quicksort(T, 0, T.size() - 1);  
}  
  
template <typename elem>  
void quicksort(vector<elem>& T, int e, int d) {  
    if (e < d) {  
        int q = partition(T, e, d);  
        quicksort(T, e, q);  
        quicksort(T, q + 1, d);  
    }    }
```



e

q

d

- Paral·lelismes amb l'ordenació per fusió:
 - resol dos subproblemes
 - fa un treball addicional lineal
- Estratègies oposades:
 - Ordenació per fusió:
 - divisió en subproblemes trivial
 - fusió dels vectors feta amb cura
 - Ordenació ràpida:
 - divisió en subproblemes feta amb cura
 - combinació trivial

Particicó de Hoare

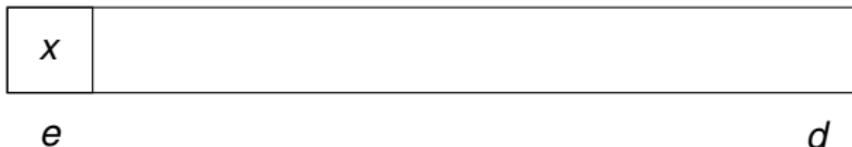
Partició original de Hoare amb el primer element com a pivot.

Partició de Hoare (*Algorismes en C++, EDA*)

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

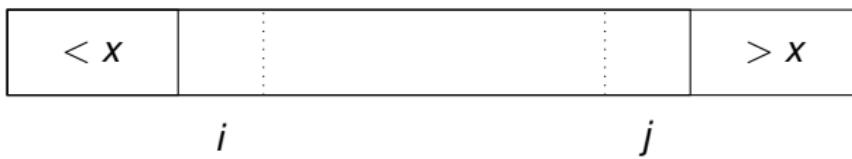
Partició de Hoare

- Inici de la funció `partition(T, e, d)`:

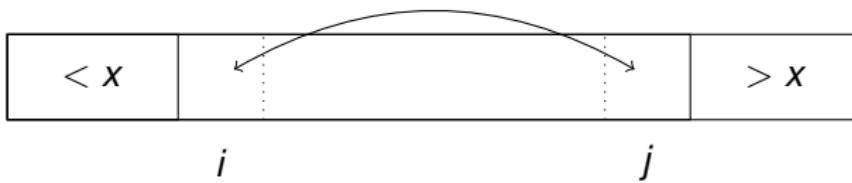


- Bucle principal:

- es troben els valors de i, j més centrals tals que



- s'intercanvien els continguts de les posicions i, j



Exercici

Simuleu `partition(T, 0, 7)` per a la taula `T`: $\langle 4, 2, 1, 6, 4, 0, 2, 6 \rangle$.

- Una altra diferència amb l'ordenació per fusió és que els subproblemes no sempre tenen la mateixa mida
- La mida dels subproblemes depèn de la tria de l'element inicial, anomenat pivot

Considerem tres estratègies per a l'elecció del pivot:

1 Tria el **primer** element:

- és acceptable si l'entrada és aleatòria
- si l'entrada està ordenada (en ordre creixent o decreixent), l'algorisme pren temps $\Theta(n^2)$ per no fer res

2 Tria un element **aleatori**

- és una tria segura per dividir en subproblemes semblants
- no sempre fa l'algorisme més ràpid perquè generar nombres aleatoris pot ser costós

3 Tria la **mediana** de tres elements

- la millor opció seria triar la mediana del vector, però és massa cara
- una bona estimació és fer la mediana de 3, normalment el 1r, l'element mig i l'últim

Considerem tres estratègies per a l'elecció del pivot:

1 Tria el **primer** element:

- és acceptable si l'entrada és aleatòria
- si l'entrada està ordenada (en ordre creixent o decreixent), l'algorisme pren temps $\Theta(n^2)$ per no fer res

2 Tria un element **aleatori**

- és una tria segura per dividir en subproblemes semblants
- no sempre fa l'algorisme més ràpid perquè generar nombres aleatoris pot ser costós

3 Tria la **mediana** de tres elements

- la millor opció seria triar la mediana del vector, però és massa cara
- una bona estimació és fer la mediana de 3, normalment el 1r, l'element mig i l'últim

Considerem tres estratègies per a l'elecció del pivot:

1 Tria el **primer** element:

- és acceptable si l'entrada és aleatòria
- si l'entrada està ordenada (en ordre creixent o decreixent), l'algorisme pren temps $\Theta(n^2)$ per no fer res

2 Tria un element **aleatori**

- és una tria segura per dividir en subproblemes semblants
- no sempre fa l'algorisme més ràpid perquè generar nombres aleatoris pot ser costós

3 Tria la **mediana** de tres elements

- la millor opció seria triar la mediana del vector, però és massa cara
- una bona estimació és fer la mediana de 3, normalment el 1r, l'element mig i l'últim

Ordenació ràpida per a pivot aleatori (*Algorismes en C++, EDA*)

```
template <typename elem>
void quicksort (vector<elem>& T, int e, int d) {
    if (e < d) {
        int p = randint(e, d);
        swap(T[e], T[p]);
        int q = partition(T, e, d);
        quicksort(T, e, q);
        quicksort(T, q + 1, d);
    }
}
```

Ordenació ràpida amb la mediana de tres com a pivot

```
template <typename elem>
void quicksort (vector<elem>& T, int e, int d) {
    if (e < d) {
        int centre = (e + d) / 2;
        if (T[e] < T[centre]) swap(T[centre], T[e]);
        if (T[d] < T[centre]) swap(T[centre], T[d]);
        if (T[d] < T[e]) swap(T[e], T[d]);
        // el pivot es en posicio e
        int q = partition(T, e, d);
        quicksort(T, e, q);
        quicksort(T, q + 1, d);
    }
}
```

Després de les línies 1–3, tenim $T[\text{centre}] \leq T[e]$, $T[d] \leq T[e]$ i $T[e] \leq T[d]$.

Per tant, $T[\text{centre}] \leq T[e] \leq T[d]$ i la mediana és $T[e]$.

Per a vectors molt petits, l'ordenació per inserció es comporta millor que *quicksort*. Una bona solució, doncs, és tallar la recursió quan el vector és més petit que una certa mida (normalment, entre 5 i 20).

Ordenació ràpida amb inserció per a vectors petits

```
template <typename elem>
void quicksort (vector<elem>& T, int e, int d) {
    const int talla_critica = 20;
    if (d - e < talla_critica)
        ordena_insercio(T, e, d);
    else {
        int q = partition(T, e, d);
        quicksort(T, e, q);
        quicksort(T, q + 1, d);
    }
}
```

Recurrència

Sigui $T(n)$ el cost de l'algorithm d'ordenació ràpida amb n elements.
Llavors,

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

on i és el nombre d'elements més petits que el pivot.

Recurrència

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

En el **cas pitjor**, el pivot és sempre l'element més petit o el més gran. Com a resultat, una crida recursiva té cost constant i l'altra $T(n - 1)$.

Cas pitjor

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

Resolent la recurrència directament (o aplicant el teorema mestre I), s'obté:

$$T(n) \in \Theta(n^2).$$

Recurrència

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

En el **cas pitjor**, el pivot és sempre l'element més petit o el més gran. Com a resultat, una crida recursiva té cost constant i l'altra $T(n - 1)$.

Cas pitjor

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

Resolent la recurrència directament (o aplicant el teorema mestre I), s'obté:

$$T(n) \in \Theta(n^2).$$

Recurrència

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

En el **cas millor**, el pivot és la mediana del vector i, per tant, els dos subvectors tenen la mateixa mida (la diferència de ≤ 1 és irrelevant per a la Θ).

Cas millor

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

És la mateixa recurrència de l'ordenació per fusió que, pel teorema mestre II, dona el cost:

$$T(n) \in \Theta(n \log n).$$

Recurrència

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

En el **cas millor**, el pivot és la mediana del vector i, per tant, els dos subvectors tenen la mateixa mida (la diferència de ≤ 1 és irrelevant per a la Θ).

Cas millor

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

És la mateixa recurrència de l'ordenació per fusió que, pel teorema mestre II, dona el cost:

$$T(n) \in \Theta(n \log n).$$

Recurrència

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

Per al **cas mitjà**, suposarem que la partició és aleatòria i, per tant, cada crida té probabilitat $1/n$. El valor mitjà de $T(i)$ i de $T(n - i - 1)$ serà $\frac{1}{n} \sum_{j=0}^{n-1} T(j)$.

Cas mitjà

Per a alguna constant c ,

$$T(n) = \frac{2}{n} \left[\sum_{j=0}^{n-1} T(j) \right] + cn \quad (1)$$

Recurrència

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

Per al **cas mitjà**, suposarem que la partició és aleatòria i, per tant, cada crida té probabilitat $1/n$. El valor mitjà de $T(i)$ i de $T(n - i - 1)$ serà $\frac{1}{n} \sum_{j=0}^{n-1} T(j)$.

Cas mitjà

Per a alguna constant c ,

$$T(n) = \frac{2}{n} \left[\sum_{j=0}^{n-1} T(j) \right] + cn \quad (1)$$

Anàlisi: cas mitjà

Multipliquem l'equació 1 per n :

$$nT(n) = 2 \left[\sum_{j=0}^{n-1} T(j) \right] + cn^2. \quad (2)$$

Per eliminar el sumatori, escrivim el cas $n - 1$

$$(n-1)T(n-1) = 2 \left[\sum_{j=0}^{n-2} T(j) \right] + c(n-1)^2 \quad (3)$$

i restem l'equació 3 de la 2

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c. \quad (4)$$

Reordenant els termes i eliminant c ,

$$nT(n) = (n+1)T(n-1) + 2cn. \quad (5)$$

Anàlisi: cas mitjà

Multipliquem l'equació 1 per n :

$$nT(n) = 2 \left[\sum_{j=0}^{n-1} T(j) \right] + cn^2. \quad (2)$$

Per eliminar el sumatori, escrivim el cas $n - 1$

$$(n - 1)T(n - 1) = 2 \left[\sum_{j=0}^{n-2} T(j) \right] + c(n - 1)^2 \quad (3)$$

i restem l'equació 3 de la 2

$$nT(n) - (n - 1)T(n - 1) = 2T(n - 1) + 2cn - c. \quad (4)$$

Reordenant els termes i eliminant c ,

$$nT(n) = (n + 1)T(n - 1) + 2cn. \quad (5)$$

Anàlisi: cas mitjà

Multipliquem l'equació 1 per n :

$$nT(n) = 2 \left[\sum_{j=0}^{n-1} T(j) \right] + cn^2. \quad (2)$$

Per eliminar el sumatori, escrivim el cas $n - 1$

$$(n - 1)T(n - 1) = 2 \left[\sum_{j=0}^{n-2} T(j) \right] + c(n - 1)^2 \quad (3)$$

i restem l'equació 3 de la 2

$$nT(n) - (n - 1)T(n - 1) = 2T(n - 1) + 2cn - c. \quad (4)$$

Reordenant els termes i eliminant c ,

$$nT(n) = (n + 1)T(n - 1) + 2cn. \quad (5)$$

Anàlisi: cas mitjà

Dividim l'equació 5 per $n(n+1)$:

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}. \quad (6)$$

Substituem n per tots els valors des de $n-1$ fins a 2:

$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{2c}{n}. \quad (7)$$

$$\frac{T(n-2)}{n-1} = \frac{T(n-3)}{n-2} + \frac{2c}{n-1}. \quad (8)$$

⋮

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}. \quad (9)$$

La suma de totes les equacions 6–9 dona

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}. \quad (10)$$

Anàlisi: cas mitjà

Dividim l'equació 5 per $n(n+1)$:

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}. \quad (6)$$

Substituem n per tots els valors des de $n-1$ fins a 2:

$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{2c}{n}. \quad (7)$$

$$\frac{T(n-2)}{n-1} = \frac{T(n-3)}{n-2} + \frac{2c}{n-1}. \quad (8)$$

⋮

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}. \quad (9)$$

La suma de totes les equacions 6–9 dona

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}. \quad (10)$$

Anàlisi: cas mitjà

Dividim l'equació 5 per $n(n+1)$:

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}. \quad (6)$$

Substituem n per tots els valors des de $n-1$ fins a 2:

$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{2c}{n}. \quad (7)$$

$$\frac{T(n-2)}{n-1} = \frac{T(n-3)}{n-2} + \frac{2c}{n-1}. \quad (8)$$

⋮

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}. \quad (9)$$

La suma de totes les equacions 6–9 dona

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}. \quad (10)$$

D'altra banda, se sap que

$$\sum_{i=3}^{n+1} \frac{1}{i} = \ln(n+1) + \gamma - \frac{3}{2},$$

on $\gamma \approx 0.577$ és la constant d'Euler. Substituint aquest valor en l'equació 10 obtinguda abans

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}$$

es dedueix que

$$\frac{T(n)}{n+1} \in \Theta(\log n)$$

i, per tant,

$$T(n) \in \Theta(n \log n).$$

Exercici: *Quicksort* a mà

Ordeneu la taula $\langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$ amb l'algorithm d'ordenació ràpida i la partició de Hoare.

Exercici *Quicksort*

Digueu quant triga l'algorithm d'ordenació ràpida amb la partició de Hoare quan l'entrada es troba:

- permutada a l'atzar
- ordenada
- ordenada de l'inrevés
- amb tots els elements iguals

Exercici: *Quicksort* a mà

Ordeneu la taula $\langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$ amb l'algorithm d'ordenació ràpida i la partició de Hoare.

Exercici *Quicksort*

Digueu quant triga l'algorithm d'ordenació ràpida amb la partició de Hoare quan l'entrada es troba:

- permutada a l'atzar
- ordenada
- ordenada de l'inrevés
- amb tots els elements iguals

Tema 2. Dividir i vèncer

1 Ordenació per fusió

- Algorisme de fusió bàsic
- Variants

2 Ordenació ràpida

- Algorisme general
- Variants
- Anàlisi

3 Productes i exponents

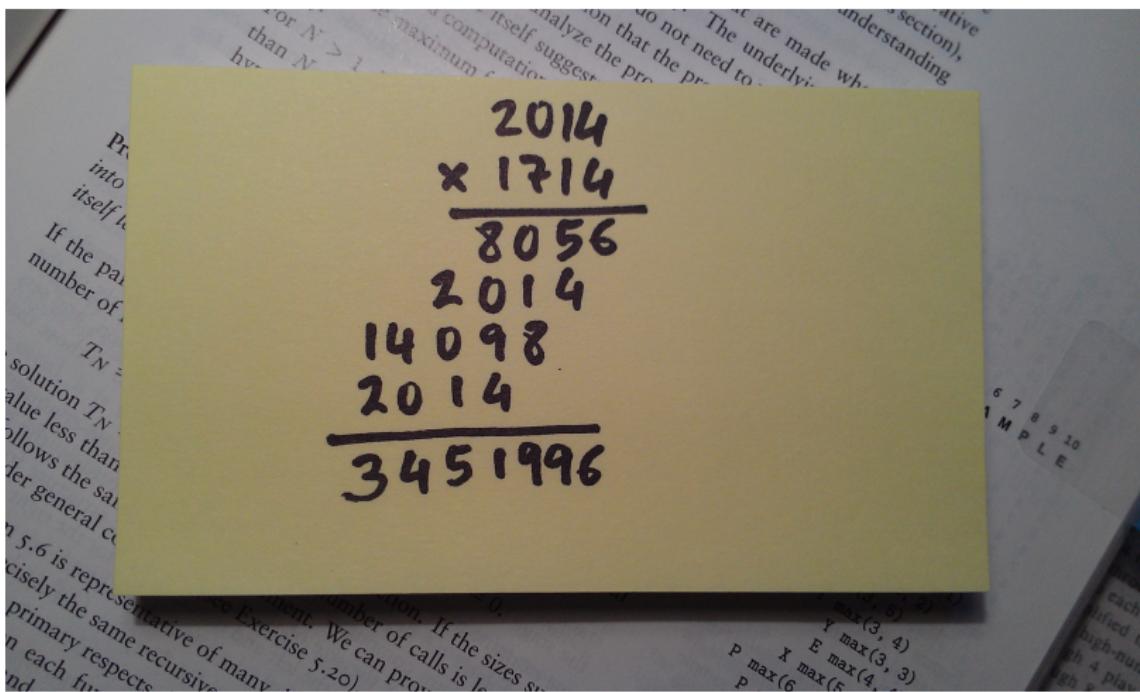
- Algorisme de Karatsuba
- Exponenciació ràpida
- Algorisme de Strassen

4 Altres algorismes

- Torres de Hanoi
- Mediana

Algorisme de Karatsuba

Quin cost té l'algorisme escolar de multiplicació?



Algorisme de Karatsuba

$\Theta(n^2)$, on n és el nombre de díigits del nombre més alt.

$$\begin{array}{r} 2014 \\ \times 1714 \\ \hline 8056 \\ 2014 \\ 14098 \\ 2014 \\ \hline 3451996 \end{array}$$

$n \times n$

Algorisme de Karatsuba

El matemàtic rus A. Kolmogorov va conjecturar que l'algorisme escolar de multiplicació és òptim.

Conjectura (Kolmogorov, 1952)

Qualsevol algorisme per multiplicar dos nombres de n díigits té cost $\Omega(n^2)$.

Un estudiant de 23 anys de Kolmogorov, Anatolii Alexeevitch Karatsuba, va trobar un algorisme de cost $\Theta(n^{1.585})$.

Refutació (Karatsuba, 1960)

Hi ha un algorisme que multiplica dos nombres de n díigits en temps

$$\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585}).$$

Algorisme de Karatsuba

El matemàtic rus A. Kolmogorov va conjecturar que l'algorisme escolar de multiplicació és òptim.

Conjectura (Kolmogorov, 1952)

Qualsevol algorisme per multiplicar dos nombres de n díigits té cost $\Omega(n^2)$.

Un estudiant de 23 anys de Kolmogorov, Anatolii Alexeevitch Karatsuba, va trobar un algorisme de cost $\Theta(n^{1.585})$.

Refutació (Karatsuba, 1960)

Hi ha un algorisme que multiplica dos nombres de n díigits en temps

$$\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585}).$$

Algorisme de Karatsuba

L'origen de la idea de l'algorisme de Karatsuba es remunta al s. XVIII.



Observació

El matemàtic Carl Friedrich Gauss (1777-1855) va observar que malgrat que el producte de dos complexos

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

sembla requerir 4 productes de reals, es pot obtenir només amb 3.
La raó és que

$$bc + ad = (a + b)(c + d) - ac - bd.$$

Algorisme de Karatsuba

Avancem ara al s. XX.

Suposem que x i y són dos enters de n bits. Expresssem x , y en dues parts:

$$x = \boxed{x_E} \quad \boxed{x_D} = 2^{\lfloor n/2 \rfloor} x_E + x_D$$

$$y = \boxed{y_E} \quad \boxed{y_D} = 2^{\lfloor n/2 \rfloor} y_E + y_D$$

Exemple

Si $x = 10010111_2$ i $y = 11001010_2$ (el subíndex 2 vol dir "en binari"), llavors

$$x = \boxed{x_E} \quad \boxed{x_D} = \boxed{1001}_2 \quad \boxed{0111}_2$$

$$y = \boxed{y_E} \quad \boxed{y_D} = \boxed{1100}_2 \quad \boxed{1010}_2$$

Algorisme de Karatsuba

Ara, el producte

$$xy = (2^{\lfloor n/2 \rfloor} x_E + x_D)(2^{\lfloor n/2 \rfloor} y_E + y_D)$$

quan n és parell es pot reescriure com

$$xy = 2^n x_E y_E + 2^{n/2} (x_E y_D + x_D y_E) + x_D y_D \quad (11)$$

i quan n és senar com

$$xy = 2^{n-1} x_E y_E + 2^{\lfloor n/2 \rfloor} (x_E y_D + x_D y_E) + x_D y_D.$$

Un algorisme basat en aquesta expressió tindria un cost

$$T(n) = 4T(n/2) + \Theta(n).$$

Algorisme de Karatsuba

Ara, el producte

$$xy = (2^{\lfloor n/2 \rfloor} x_E + x_D)(2^{\lfloor n/2 \rfloor} y_E + y_D)$$

quan n és parell es pot reescriure com

$$xy = 2^n x_E y_E + 2^{n/2} (x_E y_D + x_D y_E) + x_D y_D \quad (11)$$

i quan n és senar com

$$xy = 2^{n-1} x_E y_E + 2^{\lfloor n/2 \rfloor} (x_E y_D + x_D y_E) + x_D y_D.$$

Un algorisme basat en aquesta expressió tindria un cost

$$T(n) = 4T(n/2) + \Theta(n).$$

Algorisme de Karatsuba

$$T(n) = 4T(n/2) + \Theta(n)$$

Pel teorema mestre II, sabem que $T(n) \in \Theta(n^2)$.

Però si apliquem el **truc de Gauss**, podem obtenir el producte xy fent servir 3 subproductes en lloc de 4 i, així, rebaixar el cost quadràtic.

Algorisme de Karatsuba

$$T(n) = 4T(n/2) + \Theta(n)$$

Pel teorema mestre II, sabem que $T(n) \in \Theta(n^2)$.

Però si apliquem el **truc de Gauss**, podem obtenir el producte xy fent servir **3 subproductes en lloc de 4** i, així, rebaixar el cost quadràtic.

Algorisme de Karatsuba

Com abans, observem que

$$x_E y_D + x_D y_E = (x_E + x_D)(y_E + y_D) - x_E y_E - x_D y_D.$$

Si ara anomenem

$$\textcolor{red}{a} = x_E y_E, \quad \textcolor{red}{b} = x_D y_D, \quad \textcolor{red}{c} = (x_E + x_D)(y_E + y_D),$$

llavors el producte per a n parell (l'equació 11)

$$xy = 2^n x_E y_E + 2^{n/2}(x_E y_D + x_D y_E) + x_D y_D$$

es pot reescrivre com

$$2^n \textcolor{red}{a} + 2^{n/2}(\textcolor{red}{c} - \textcolor{red}{a} - \textcolor{red}{b}) + \textcolor{red}{b}$$

que només depèn de 3 subproductes (com el cas de n senar).

Algorisme de Karatsuba

Com abans, observem que

$$x_E y_D + x_D y_E = (x_E + x_D)(y_E + y_D) - x_E y_E - x_D y_D.$$

Si ara anomenem

$$\textcolor{red}{a} = x_E y_E, \quad \textcolor{red}{b} = x_D y_D, \quad \textcolor{red}{c} = (x_E + x_D)(y_E + y_D),$$

llavors el producte per a n parell (l'equació 11)

$$xy = 2^n x_E y_E + 2^{n/2}(x_E y_D + x_D y_E) + x_D y_D$$

es pot reescriure com

$$2^n \textcolor{red}{a} + 2^{n/2}(\textcolor{red}{c} - \textcolor{red}{a} - \textcolor{red}{b}) + \textcolor{red}{b}$$

que només depèn de 3 subproductes (com el cas de n senar).

Algorisme de Karatsuba

La nova expressió dona lloc a un algorisme de cost

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

i, pel teorema mestre II, sabem que

$$T(n) \in \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585}).$$

Algorisme de Karatsuba

Solució cas parell en base 10

Donat n parell, $x = (10^{n/2}x_E + x_D)$ i $y = (10^{n/2}y_E + y_D)$, tenim que

$$xy = 10^n a + 10^{n/2}(c - a - b) + b$$

on $a = x_E y_E$, $b = x_D y_D$, $c = (x_E + x_D)(y_E + y_D)$.

Exemple: calcular $1234 * 4321$

- Problema: calcular $x * y$ per a $x = 1234$, $y = 4321$
- Subproblemes:
 - 1 Calcular $a = 12 * 43$
 - 2 Calcular $b = 34 * 21$
 - 3 Calcular $c = (12 + 34) * (43 + 21) = 46 * 64$

Algorisme de Karatsuba

Solució cas parell en base 10

Donat n parell, $x = (10^{n/2}x_E + x_D)$ i $y = (10^{n/2}y_E + y_D)$, tenim que

$$xy = 10^n \textcolor{red}{a} + 10^{n/2}(\textcolor{red}{c} - \textcolor{red}{a} - \textcolor{red}{b}) + \textcolor{red}{b}$$

on $\textcolor{red}{a} = x_E y_E$, $\textcolor{red}{b} = x_D y_D$, $\textcolor{red}{c} = (x_E + x_D)(y_E + y_D)$.

Subproblema 1: calcular $a = 12 * 43$

- Subproblemes:
 - 1 Calcular $a_1 = 1 * 4 = 4$
 - 2 Calcular $b_1 = 2 * 3 = 6$
 - 3 Calcular $c_1 = (1 + 2) * (4 + 3) = 21$
- Solució: $a = 10^2 * 4 + 10 * (21 - 4 - 6) + 6 = 516$

Algorisme de Karatsuba

Solució cas parell en base 10

Donat n parell, $x = (10^{n/2}x_E + x_D)$ i $y = (10^{n/2}y_E + y_D)$, tenim que

$$xy = 10^n \textcolor{red}{a} + 10^{n/2}(\textcolor{red}{c} - \textcolor{red}{a} - \textcolor{red}{b}) + \textcolor{red}{b}$$

on $\textcolor{red}{a} = x_E y_E$, $\textcolor{red}{b} = x_D y_D$, $\textcolor{red}{c} = (x_E + x_D)(y_E + y_D)$.

Subproblema 2: calcular $b = 34 * 21$

- Subproblemes:
 - 1 Calcular $a_2 = 3 * 2 = 6$
 - 2 Calcular $b_2 = 4 * 1 = 4$
 - 3 Calcular $c_2 = (3 + 4) * (2 + 1) = 21$
- Solució: $b = 10^2 * 6 + 10 * (21 - 6 - 4) + 4 = 714$

Solució cas parell en base 10

Donat n parell, $x = (10^{n/2}x_E + x_D)$ i $y = (10^{n/2}y_E + y_D)$, tenim que

$$xy = 10^n \textcolor{red}{a} + 10^{n/2}(\textcolor{red}{c} - \textcolor{red}{a} - \textcolor{red}{b}) + \textcolor{red}{b}$$

on $\textcolor{red}{a} = x_E y_E$, $\textcolor{red}{b} = x_D y_D$, $\textcolor{red}{c} = (x_E + x_D)(y_E + y_D)$.

Subproblema 3: calcular $c = 46 * 64$

- Subproblemes:
 - 1 Calcular $a_3 = 4 * 6 = 24$
 - 2 Calcular $b_3 = 6 * 4 = 24$
 - 3 Calcular $c_3 = (4 + 6) * (6 + 4) = 100$
- Solució: $c = 10^2 * 24 + 10 * (100 - 24 - 24) + 24 = 2944$

Algorisme de Karatsuba

Solució cas parell en base 10

Donat n parell, $x = (10^{n/2}x_E + x_D)$ i $y = (10^{n/2}y_E + y_D)$, tenim que

$$xy = 10^n a + 10^{n/2}(c - a - b) + b$$

on $a = x_E y_E$, $b = x_D y_D$, $c = (x_E + x_D)(y_E + y_D)$.

Resultat

- Problema: calcular $x * y$ per a $x = 1234$, $y = 4321$
- Subproblemes:
 - 1 $a = 12 * 43 = 516$
 - 2 $b = 34 * 21 = 714$
 - 3 $c = 46 * 64 = 2944$
- Solució: $10^4 * 516 + 10^2 * (2944 - 516 - 714) + 714 = 5332114$

Algorisme de Karatsuba

Exercici 1

Digueu per què entre n i $2n$ sempre hi ha una potència de 2 per a qualsevol natural n .

Exercici 2

Proposeu un mètode per implementar l'algorisme de Karatsuba sense diferenciar entre el cas de n parell i n senar.

Raoneu per què té el mateix cost $\Theta(n^{\log_2 3})$ que Karatsuba.

Algorisme de Karatsuba

Exercici 1

Digueu per què entre n i $2n$ sempre hi ha una potència de 2 per a qualsevol natural n .

Exercici 2

Proposeu un mètode per implementar l'algorisme de Karatsuba sense diferenciar entre el cas de n parell i n senar.

Raoneu per què té el mateix cost $\Theta(n^{\log_2 3})$ que Karatsuba.

Exponenciació ràpida

- L'algorisme iteratiu evident per calcular x^n fa servir la descomposició

$$x^n = \underbrace{x \cdot x \cdot \dots \cdot x}_{n \text{ factors}}$$

que requereix $\Theta(n - 1) = \Theta(n)$ multiplicacions.

- Però amb un enfocament recursiu, tenim

$$x^n = x^{n/2} \cdot x^{n/2}$$

quan n és parell i

$$x^n = x^{(n-1)/2} \cdot x^{(n-1)/2} \cdot x$$

quan n és senar.

Exponenciació ràpida

- L'algorisme iteratiu evident per calcular x^n fa servir la descomposició

$$x^n = \underbrace{x \cdot x \cdot \dots \cdot x}_{n \text{ factors}}$$

que requereix $\Theta(n - 1) = \Theta(n)$ multiplicacions.

- Però amb un enfocament recursiu, tenim

$$x^n = x^{n/2} \cdot x^{n/2}$$

quan n és parell i

$$x^n = x^{(n-1)/2} \cdot x^{(n-1)/2} \cdot x$$

quan n és senar.

Exponenciació ràpida

Definició recursiva de x^n

$$x^n = \begin{cases} 1, & \text{si } n = 0 \\ x^{n/2} \cdot x^{n/2}, & \text{si } n > 0 \text{ i parell} \\ x^{(n-1)/2} \cdot x^{(n-1)/2} \cdot x, & \text{si } n \text{ és senar} \end{cases}$$

Exemple

Amb un algorisme basat en la definició anterior, tindríem

$$\begin{aligned} x^{62} &= (\textcolor{brown}{x^{31}})^2, \quad x^{31} = (\textcolor{brown}{x^{15}})^2 \cdot x, \quad x^{15} = (\textcolor{brown}{x^7})^2 \cdot x \\ x^7 &= (\textcolor{brown}{x^3})^2 \cdot x, \quad x^3 = (\textcolor{brown}{x^1})^2 \cdot x, \quad x^1 = (\textcolor{brown}{x^0})^2 \cdot x, \quad x^0 = 1 \end{aligned}$$

(en groc, els valors calculats recursivament)

Exponenciació ràpida

Definició recursiva de x^n

$$x^n = \begin{cases} 1, & \text{si } n = 0 \\ x^{n/2} \cdot x^{n/2}, & \text{si } n > 0 \text{ i parell} \\ x^{(n-1)/2} \cdot x^{(n-1)/2} \cdot x, & \text{si } n \text{ és senar} \end{cases}$$

Exemple

Amb un algorisme basat en la definició anterior, tindríem

$$\begin{aligned} x^{62} &= (\textcolor{blue}{x}^{31})^2, \quad x^{31} = (\textcolor{blue}{x}^{15})^2 \cdot x, \quad x^{15} = (\textcolor{blue}{x}^7)^2 \cdot x \\ x^7 &= (\textcolor{blue}{x}^3)^2 \cdot x, \quad x^3 = (\textcolor{blue}{x}^1)^2 \cdot x, \quad x^1 = (\textcolor{blue}{x}^0)^2 \cdot x, \quad x^0 = 1 \end{aligned}$$

(en groc, els valors calculats recursivament)

Exponenciació ràpida

Exponenciació ràpida

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

El càlcul del cost és directe i ve donat per la recurrència

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

que, segons el teorema mestre II, implica

$$T(n) \in \Theta(\log n).$$

Qüestió

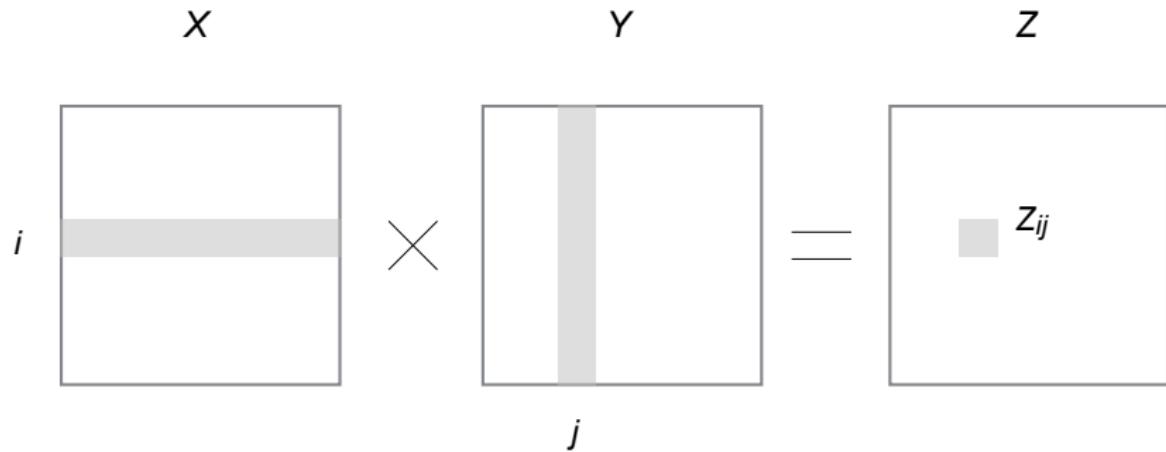
Digueu si l'algorisme d'exponenciació ràpida és **lineal** o **logarítmic** quan el seu cost s'expressa en funció de la mida de l'entrada.

Algorisme de Strassen

El producte de dues matrius X i Y de mida $n \times n$ és una matriu Z de mida $n \times n$ tal que

$$Z_{ij} = \sum_{k=1}^n X_{ik} Y_{kj},$$

que és el producte de la fila i -èsima de X per la columna j -èsima de Y



La fórmula

$$Z_{ij} = \sum_{k=1}^n X_{ik} Y_{kj}.$$

calculada per a cada i, j amb valors $i, j = 1 \dots n$ implica un algorisme $\Theta(n^3)$.

Es va pensar que el cost $\Theta(n^3)$ era òptim fins al 1969, quan Volker Strassen va anunciar un algorisme de cost $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$.

La fórmula

$$Z_{ij} = \sum_{k=1}^n X_{ik} Y_{kj}.$$

calculada per a cada i, j amb valors $i, j = 1 \dots n$ implica un algorisme $\Theta(n^3)$.

Es va pensar que el cost $\Theta(n^3)$ era òptim fins al 1969, quan Volker Strassen va anunciar un algorisme de cost $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$.

Algorisme de Strassen

Producte estàndard de matrius

Algorisme $\Theta(n^3)$, adaptat de *Data Structure and Alg. Analysis in C++, M.A. Weiss*.

```
matrix<int> producte_matrius
    (const matrix<int> & X, const matrix<int> & Y)
{
    int n = X.numrows();
    matrix<int> Z(n, n, 0);
    int i;
    for (i = 0; i < n; j++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                Z[i][j] += X[i][k] * Y[k][j];
    return Z;
}
```

Algorisme de Strassen

Una primera idea és que el producte de matrius es pot fer *per blocs*.

Dividim X i Y en quatre quadrants cadascuna:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Aleshores, es pot veure que

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

Algorisme de Strassen

Exemple

Per fer el producte de X i Y

$$X = \begin{bmatrix} 4 & 3 & 1 & 6 \\ 1 & 5 & 2 & 7 \\ 2 & 1 & 5 & 9 \\ 3 & 4 & 2 & 6 \end{bmatrix}, \quad Y = \begin{bmatrix} 2 & 6 & 9 & 4 \\ 3 & 2 & 4 & 1 \\ 1 & 1 & 8 & 3 \\ 2 & 1 & 3 & 1 \end{bmatrix},$$

definim les vuit matrius 2×2

$$A = \begin{bmatrix} 4 & 3 \\ 1 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 6 \\ 2 & 7 \end{bmatrix}, \quad E = \begin{bmatrix} 2 & 6 \\ 3 & 2 \end{bmatrix}, \quad F = \begin{bmatrix} 9 & 4 \\ 4 & 1 \end{bmatrix},$$

$$C = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}, \quad D = \begin{bmatrix} 5 & 9 \\ 2 & 6 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, \quad H = \begin{bmatrix} 8 & 3 \\ 3 & 1 \end{bmatrix}.$$

Algorisme de Strassen

Exemple

Per fer el producte de X i Y

$$X = \begin{bmatrix} 4 & 3 & 1 & 6 \\ 1 & 5 & 2 & 7 \\ 2 & 1 & 5 & 9 \\ 3 & 4 & 2 & 6 \end{bmatrix}, \quad Y = \begin{bmatrix} 2 & 6 & 9 & 4 \\ 3 & 2 & 4 & 1 \\ 1 & 1 & 8 & 3 \\ 2 & 1 & 3 & 1 \end{bmatrix},$$

i l'expressem en termes de les submatrius

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix} =$$

$$\left[\begin{array}{c} \left[\begin{array}{cc} 4 & 3 \\ 1 & 5 \end{array} \right] \left[\begin{array}{cc} 2 & 6 \\ 3 & 2 \end{array} \right] + \left[\begin{array}{cc} 1 & 6 \\ 2 & 7 \end{array} \right] \left[\begin{array}{cc} 1 & 1 \\ 2 & 1 \end{array} \right] \quad \left[\begin{array}{cc} 4 & 3 \\ 1 & 5 \end{array} \right] \left[\begin{array}{cc} 9 & 4 \\ 4 & 1 \end{array} \right] + \left[\begin{array}{cc} 1 & 6 \\ 2 & 7 \end{array} \right] \left[\begin{array}{cc} 8 & 3 \\ 3 & 1 \end{array} \right] \\ \left[\begin{array}{cc} 2 & 1 \\ 3 & 4 \end{array} \right] \left[\begin{array}{cc} 2 & 6 \\ 3 & 2 \end{array} \right] + \left[\begin{array}{cc} 5 & 9 \\ 2 & 6 \end{array} \right] \left[\begin{array}{cc} 1 & 1 \\ 2 & 1 \end{array} \right] \quad \left[\begin{array}{cc} 2 & 1 \\ 3 & 4 \end{array} \right] \left[\begin{array}{cc} 9 & 4 \\ 4 & 1 \end{array} \right] + \left[\begin{array}{cc} 5 & 9 \\ 2 & 6 \end{array} \right] \left[\begin{array}{cc} 8 & 3 \\ 3 & 1 \end{array} \right] \end{array} \right]$$

Quin cost té el nou algorisme? Recordem que es basa en el producte

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

El cost $T(n)$ és la suma de fer:

- vuit productes de matrius de mida $n/2$: $8T(n/2)$
- quatre sumes de matrius de mida $n/2$: $\Theta(n^2)$

Per tant, tenim la recurrència

$$T(n) = 8T(n/2) + \Theta(n^2)$$

que, pel teorema mestre II, dona $T(n) \in \Theta(n^{\log_2 8}) = \Theta(n^3)$.

Algorisme de Strassen

Però el nombre de productes es pot reduir a 7.

Volker Strassen (1969)

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

on

$$P_1 = A(F - H), \quad P_4 = D(G - E)$$

$$P_2 = (A + B)H, \quad P_5 = (A + D)(E + H)$$

$$P_3 = (C + D)E, \quad P_6 = (B - D)(G + H)$$

$$P_7 = (A - C)(E + F)$$

Fent servir la descomposició de Strassen, obtenim un algorisme amb cost

$$T(n) = 7T(n/2) + \Theta(n^2)$$

que, pel teorema mestre II, dona $T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$.

Algorisme de Strassen

Però el nombre de productes es pot reduir a 7.

Volker Strassen (1969)

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

on

$$P_1 = A(F - H), \quad P_4 = D(G - E)$$

$$P_2 = (A + B)H, \quad P_5 = (A + D)(E + H)$$

$$P_3 = (C + D)E, \quad P_6 = (B - D)(G + H)$$

$$P_7 = (A - C)(E + F)$$

Fent servir la descomposició de Strassen, obtenim un algorisme amb cost

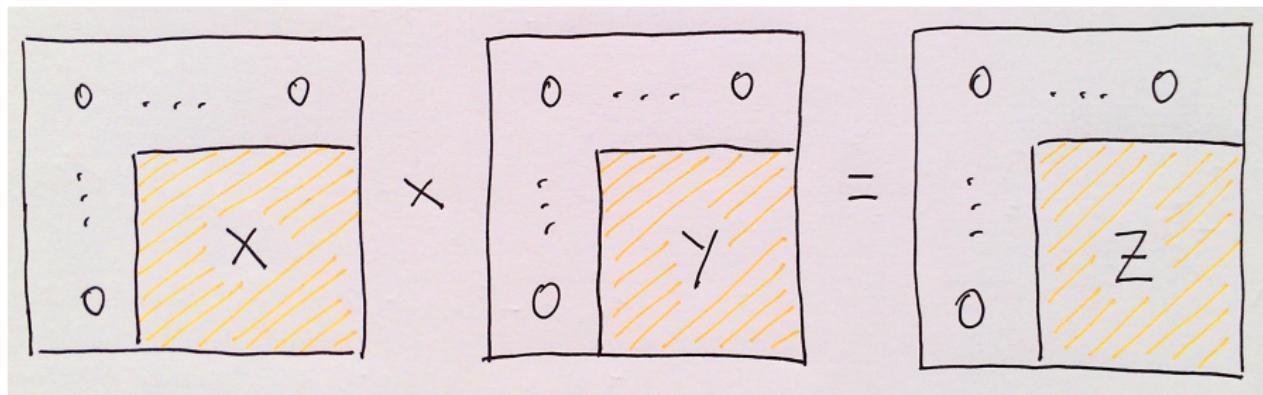
$$T(n) = 7T(n/2) + \Theta(n^2)$$

que, pel teorema mestre II, dona $T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$.

Algorisme de Strassen

Exercici

Si n no és potència de 2, generem matrius $m \times m$ on m és la primera potència de 2 més gran que n i encabim les matrius originals a dins. Per què no varia el cost de l'algorisme?



Tema 2. Dividir i vèncer

1 Ordenació per fusió

- Algorisme de fusió bàsic
- Variants

2 Ordenació ràpida

- Algorisme general
- Variants
- Anàlisi

3 Productes i exponents

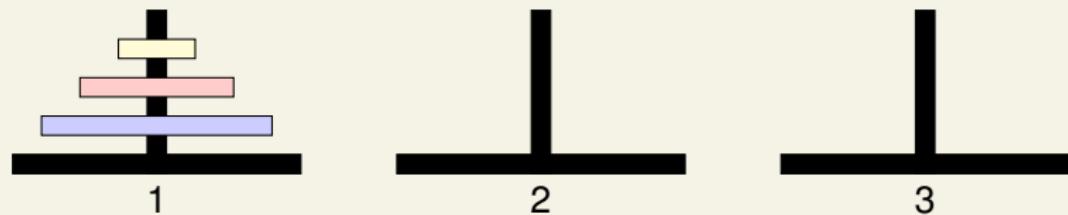
- Algorisme de Karatsuba
- Exponenciació ràpida
- Algorisme de Strassen

4 Altres algorismes

- Torres de Hanoi
- Mediana

Joc de les torres de Hanoi

Donades tres varetes 1, 2 i 3 i una sèrie de discs col·locats en la primera com indica el dibuix



cal traslladar-los a la vareta 2 amb dues restriccions:

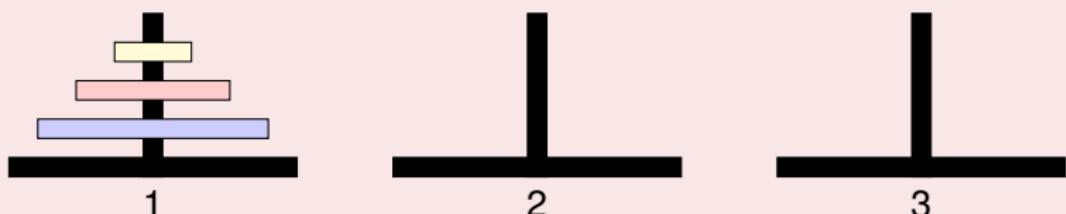
- només es pot moure un disc a la vegada
- no es pot col·locar un disc a sobre d'un de més petit

Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

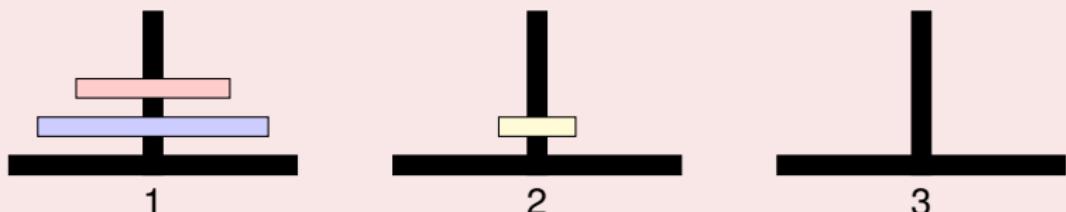


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

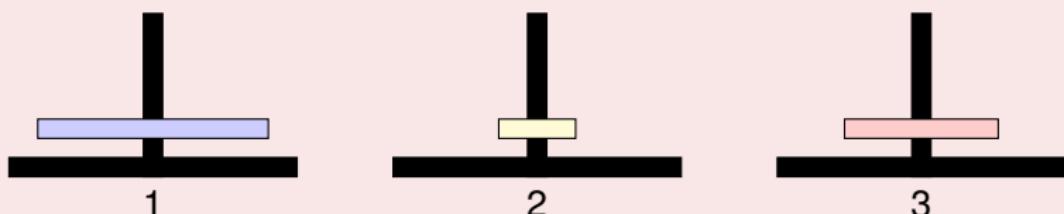


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

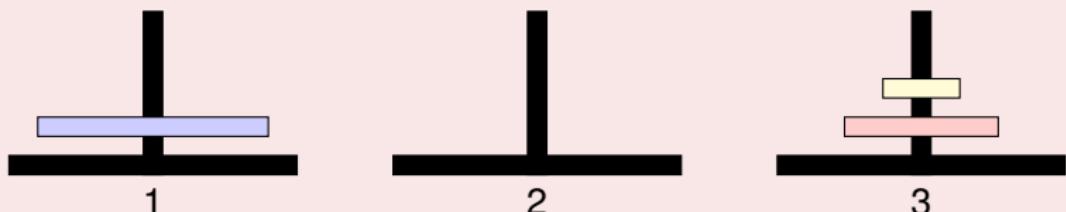


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

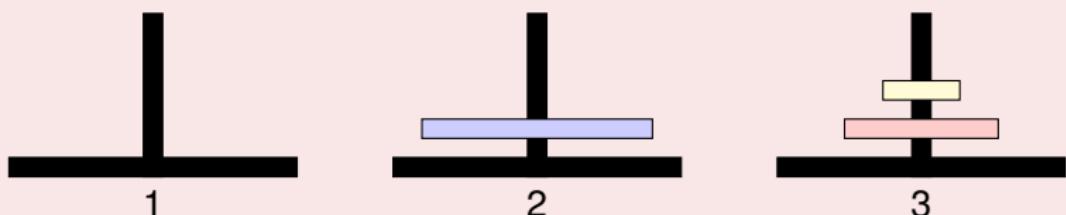


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

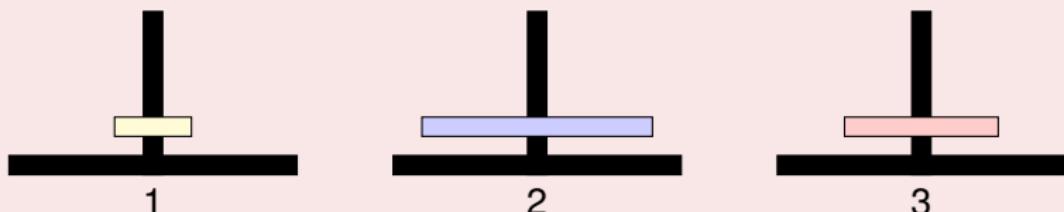


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

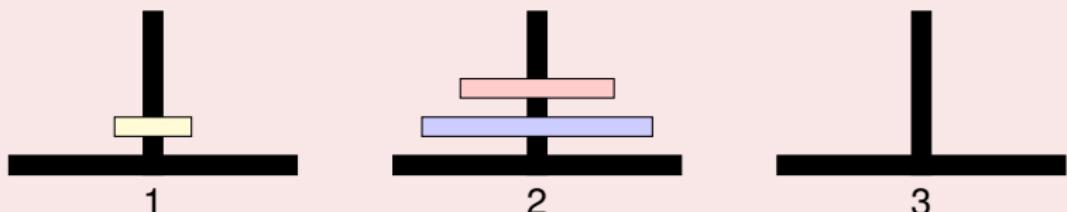


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

Moviment senar – Moviment parell

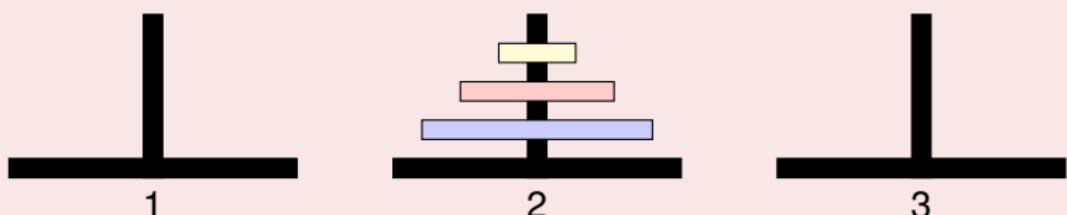


Solució 1

Imaginem les varetes disposades en forma de triangle. Ara, en els moviments:

- **senars**: movem el disc més petit una vareta en sentit horari
- **parells**: fem l'únic moviment possible que no involucri el disc més petit

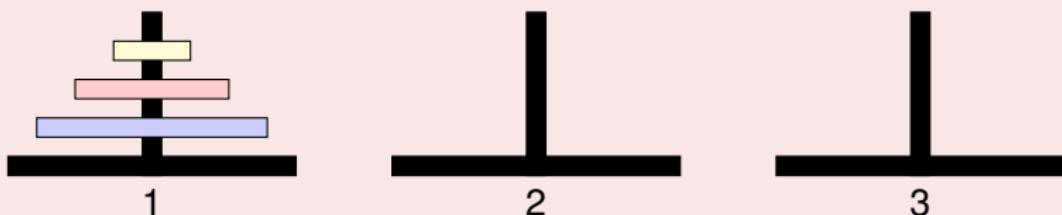
Moviment senar – Moviment parell



Solució 2

Per traslladar n discs de la vareta 1 a la 2,

- si $n = 1$, moure l'únic disc de 1 a 2.
- si $n > 1$,
 - 1 traslladar $n - 1$ discs de 1 a 3
 - 2 moure el disc restant (el més gros) de 1 a 2
 - 3 traslladar $n - 1$ discs de 3 a 2

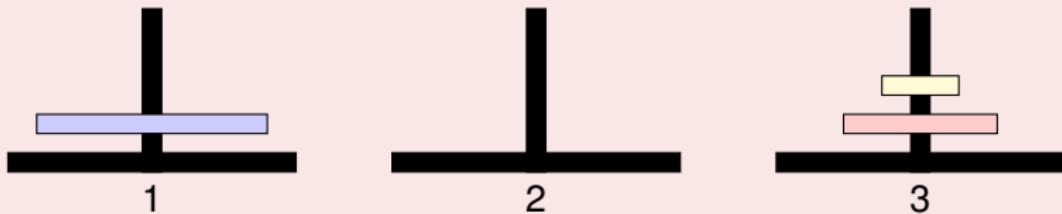


Solució 2

Per traslladar n discs de la vareta 1 a la 2,

- si $n = 1$, moure l'únic disc de 1 a 2.
- si $n > 1$,
 - 1 traslladar $n - 1$ discs de 1 a 3
 - 2 moure el disc restant (el més gros) de 1 a 2
 - 3 traslladar $n - 1$ discs de 3 a 2

Traslladar 2 discs de 1 a 3

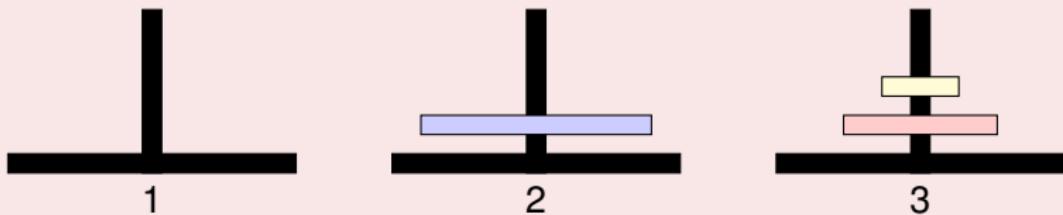


Solució 2

Per traslladar n discs de la vareta 1 a la 2,

- si $n = 1$, moure l'únic disc de 1 a 2.
- si $n > 1$,
 - 1 traslladar $n - 1$ discs de 1 a 3
 - 2 moure el disc restant (el més gros) de 1 a 2
 - 3 traslladar $n - 1$ discs de 3 a 2

Moure el disc restant de 1 a 2

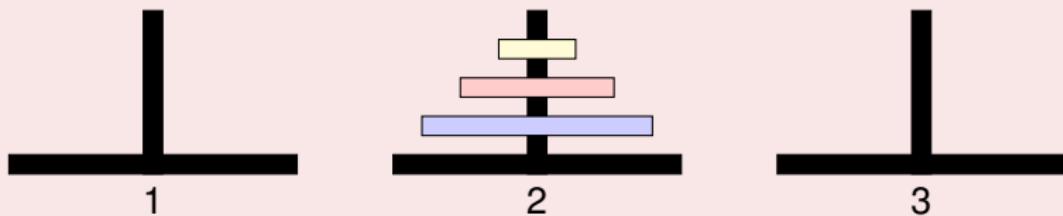


Solució 2

Per traslladar n discs de la vareta 1 a la 2,

- si $n = 1$, moure l'únic disc de 1 a 2.
- si $n > 1$,
 - 1 traslladar $n - 1$ discs de 1 a 3
 - 2 moure el disc restant (el més gros) de 1 a 2
 - 3 traslladar $n - 1$ discs de 3 a 2

Traslladar 2 discs de 3 a 2



Torres de Hanoi

Algorisme

```
void hanoi(int n, int a, int b, int c) {
    // descriu els moviments de n discs des d'a fins a b
    // fent servir c com a auxiliar
    if (n > 0) {
        hanoi(n-1, a, c, b);
        cout << a, b;
        hanoi(n-1, c, b, a);
    }
}
```

Cost

El cost creix com el nombre de moviments. Definim

$$T(n) = \text{nombre de moviments que fa } \text{hanoi}(n, 1, 2, 3).$$

Llavors,

$$T(0) = 0$$

$$T(n) = 2T(n-1) + 1, \quad \text{si } n > 0.$$

Recurrència

$$T(0) = 0$$

$$T(n) = 2T(n-1) + 1, \quad \text{si } n > 0.$$

Solució asimptòtica

Aplicant el teorema mestre I, obtenim $T \in \Theta(2^n)$.

Solució exacta

Definim $S(n) = T(n) + 1$ i l'escrivim sense dependre de $T(n)$:

$$S(0) = 1$$

$$S(n) = 2T(n-1) + 2 = 2(S(n-1) - 1) + 2 = 2S(n-1), \quad \text{si } n > 0.$$

Ara, $S(n)$ es resol directament i dona: $S(n) = 2^n$ per a tot $n \geq 0$.
Per tant,

$$T(n) = 2^n - 1 \text{ per a tot } n \geq 0.$$

Recurrència

$$T(0) = 0$$

$$T(n) = 2T(n-1) + 1, \quad \text{si } n > 0.$$

Solució asimptòtica

Aplicant el teorema mestre I, obtenim $T \in \Theta(2^n)$.

Solució exacta

Definim $S(n) = T(n) + 1$ i l'escrivim sense dependre de $T(n)$:

$$S(0) = 1$$

$$S(n) = 2T(n-1) + 2 = 2(S(n-1) - 1) + 2 = 2S(n-1), \quad \text{si } n > 0.$$

Ara, $S(n)$ es resol directament i dona: $S(n) = 2^n$ per a tot $n \geq 0$.
Per tant,

$$T(n) = 2^n - 1 \text{ per a tot } n \geq 0.$$

Recurrència

$$T(0) = 0$$

$$T(n) = 2T(n-1) + 1, \quad \text{si } n > 0.$$

Solució asimptòtica

Aplicant el teorema mestre I, obtenim $T \in \Theta(2^n)$.

Solució exacta

Definim $S(n) = T(n) + 1$ i l'escrivim sense dependre de $T(n)$:

$$S(0) = 1$$

$$S(n) = 2T(n-1) + 2 = 2(S(n-1) - 1) + 2 = 2S(n-1), \quad \text{si } n > 0.$$

Ara, $S(n)$ es resol directament i dona: $S(n) = 2^n$ per a tot $n \geq 0$.

Per tant,

$$T(n) = 2^n - 1 \quad \text{per a tot } n \geq 0.$$

Mediana

Definició

La **mediana** d'una llista de nombres és l'element per al qual n'hi tants de més petits com de més grans.

Si la llista té llargària senar prenem, per exemple, el més petit.

Exemple

La mediana de [15, 3, 34, 5, 10] és 10 perquè quan s'escriuen en ordre, és el que queda al mig:

3 5 10 15 34

La mediana de [15, 3, 12, 34, 5, 10] també és 10 perquè la llista és

3 5 10 12 15 34

Mediana

Definició

La **mediana** d'una llista de nombres és l'element per al qual n'hi tants de més petits com de més grans.

Si la llista té llargària senar prenem, per exemple, el més petit.

Exemple

La mediana de [15, 3, 34, 5, 10] és 10 perquè quan s'escriuen en ordre, és el que queda al mig:

3 5 10 15 34

La mediana de [15, 3, 12, 34, 5, 10] també és 10 perquè la llista és

3 5 10 12 15 34

Característiques de la mediana:

- Sempre és **un dels valors** del conjunt de dades
- És **menys sensible a les observacions atípiques (outliers)**. Per exemple:
 - ① Donats els nombres 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 100
(10 vegades 1 i una vegada 100),
 - la mitjana és 10
 - la mediana és 1
 - ② Donats 2, 4, 6, 8, 10000,
 - la mitjana és 1002
 - la mediana és 6

Mediana

Per **calcular** la mediana, n'hi ha prou a ordenar els elements.

Es pot fer en temps $\Theta(n \log n)$

[8, 0, 3, 10, 5, 7, 12, 3, 7, 2, 9, 1, 6]



[0, 1, 2, 3, 3, 5, 6, 7, 8, 9, 10, 12]

El problema és que es fa més feina de la necessària:

només volem l'element del mig i no caldria ordenar la resta

{0, 3, 5, 3, 2, 1}, 6, {8, 10, 7, 12, 7, 9}

Mediana

Per **calcular** la mediana, n'hi ha prou a ordenar els elements.

Es pot fer en temps $\Theta(n \log n)$

[8, 0, 3, 10, 5, 7, 12, 3, 7, 2, 9, 1, 6]



[0, 1, 2, 3, 3, 5, 6, 7, 8, 9, 10, 12]

El problema és que es fa més feina de la necessària:

només volem l'element del mig i no caldria ordenar la resta

{0, 3, 5, 3, 2, 1}, 6, {8, 10, 7, 12, 7, 9}

Mediana

Sovint és més fàcil treballar amb una versió més general d'un problema. En aquest cas, triem **el problema de selecció**.

Definició

Si S és una llista i $k \geq 1$, anomenem

$$\text{selecció}(S, k)$$

al k -èsim element més petit de S .

Problema de selecció

Donada una llista S i un natural k , determinar $\text{selecció}(S, k)$.

La mediana d'una llista S de n nombres és $\text{selecció}(S, \lfloor n/2 \rfloor)$.

Mediana

Sovint és més fàcil treballar amb una versió més general d'un problema. En aquest cas, triem **el problema de selecció**.

Definició

Si S és una llista i $k \geq 1$, anomenem

$\text{selecció}(S, k)$

al k -èsim element més petit de S .

Problema de selecció

Donada una llista S i un natural k , determinar $\text{selecció}(S, k)$.

La mediana d'una llista S de n nombres és $\text{selecció}(S, \lfloor n/2 \rfloor)$.

Problema de selecció

Donada una llista S i un natural k , determinar selecció(S, k).

Idea per a un algorisme

Per a cada nombre x , dividim la llista en 3 conjunts d'elements:

- els més petits que x
- els que són iguals a x
- els més grans que x

Si tenim el vector

$$S : [\begin{array}{cccccccccccc} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{array}]$$

per a $x = 5$ el dividim en

$$S_E : [\begin{array}{ccc} 2 & 4 & 1 \end{array}] \quad S_x : [\begin{array}{cc} 5 & 5 \end{array}] \quad S_D : [\begin{array}{cccccccc} 36 & 21 & 8 & 13 & 11 & 20 \end{array}]$$

Mediana

Idea per a un algorisme

$$S_E : [2 \ 4 \ 1] \quad S_x : [5 \ 5] \quad S_D : [36 \ 21 \ 8 \ 13 \ 11 \ 20]$$

Suposem ara que volem el 8è element de S

$$S : [2 \ 36 \ 5 \ 21 \ 8 \ 13 \ 11 \ 20 \ 5 \ 4 \ 1]$$

Sabem que serà el 3r element de S_D perquè $|S_E| + |S_x| = 5$.

$$S_E : [2 \ 4 \ 1] \quad S_x : [5 \ 5] \quad S_D : [36 \ 21 \ 8 \ 13 \ 11 \ 20]$$

Podem definir l'operador selecció(S, k) de manera recursiva:

$$\text{selecció}(S, k) = \begin{cases} \text{selecció}(S_E, k), & \text{si } k \leq |S_E| \\ x, & \text{si } |S_E| < k \leq |S_E| + |S_x| \\ \text{selecció}(S_D, k - |S_E| - |S_x|), & \text{si } k > |S_E| + |S_x| \end{cases}$$

Mediana

Idea per a un algorisme

$$S_E : [2 \ 4 \ 1] \quad S_x : [5 \ 5] \quad S_D : [36 \ 21 \ 8 \ 13 \ 11 \ 20]$$

Suposem ara que volem el 8è element de S

$$S : [2 \ 36 \ 5 \ 21 \ 8 \ 13 \ 11 \ 20 \ 5 \ 4 \ 1]$$

Sabem que serà el 3r element de S_D perquè $|S_E| + |S_x| = 5$.

$$S_E : [2 \ 4 \ 1] \quad S_x : [5 \ 5] \quad S_D : [36 \ 21 \ 8 \ 13 \ 11 \ 20]$$

Podem definir l'operador **selecció**(S, k) de manera recursiva:

$$\text{selecció}(S, k) = \begin{cases} \text{selecció}(S_E, k), & \text{si } k \leq |S_E| \\ x, & \text{si } |S_E| < k \leq |S_E| + |S_x| \\ \text{selecció}(S_D, k - |S_E| - |S_x|), & \text{si } k > |S_E| + |S_x| \end{cases}$$

Esbós de l'algorithm

Algorithm:

- 1 Triar un element x a l'atzar
- 2 Dividir S respecte de x : S_E , S_x , S_D
- 3 Calcular selecció(S, k) recursivament

A quin ritme convergeix?

- diem que un element triat és **bo** si està entre el 25% i el 75% del vector
- \Rightarrow un element triat a l'atzar té una probabilitat del 50% de ser bo
- \Rightarrow cal triar 2 elements de mitjana abans de trobar-ne un de bo

Això dona

$$T(n) = T(3n/4) + O(n)$$

que implica que $T(n) \in O(n)$ en mitjana.

Esbós de l'algorithm

Algorithm:

- 1 Triar un element x a l'atzar
- 2 Dividir S respecte de x : S_E , S_x , S_D
- 3 Calcular selecció(S, k) recursivament

A quin ritme convergeix?

- diem que un element triat és **bo** si està entre el 25% i el 75% del vector
- \Rightarrow un element triat a l'atzar té una probabilitat del 50% de ser bo
- \Rightarrow cal triar 2 elements de mitjana abans de trobar-ne un de bo

Això dona

$$T(n) = T(3n/4) + O(n)$$

que implica que $T(n) \in O(n)$ en mitjana.

Esbós de l'algorithm

Algorithm:

- 1 Triar un element x a l'atzar
- 2 Dividir S respecte de x : S_E , S_x , S_D
- 3 Calcular selecció(S, k) recursivament

A quin ritme convergeix?

- diem que un element triat és **bo** si està entre el 25% i el 75% del vector
- \Rightarrow un element triat a l'atzar té una probabilitat del 50% de ser bo
- \Rightarrow cal triar 2 elements de mitjana abans de trobar-ne un de bo

Això dona

$$T(n) = T(3n/4) + O(n)$$

que implica que $T(n) \in O(n)$ en mitjana.

Esbós de l'algorithm

Algorithm:

- 1 Triar un element x a l'atzar
- 2 Dividir S respecte de x : S_E , S_x , S_D
- 3 Calcular selecció(S, k) recursivament

A quin ritme convergeix?

- diem que un element triat és **bo** si està entre el 25% i el 75% del vector
- \Rightarrow un element triat a l'atzar té una probabilitat del 50% de ser bo
- \Rightarrow cal triar 2 elements de mitjana abans de trobar-ne un de bo

Això dona

$$T(n) = T(3n/4) + O(n)$$

que implica que $T(n) \in O(n)$ en mitjana.

Esbós de l'algorithm

Algorithm:

- 1 Triar un element x a l'atzar
- 2 Dividir S respecte de x : S_E , S_x , S_D
- 3 Calcular selecció(S, k) recursivament

A quin ritme convergeix?

- diem que un element triat és **bo** si està entre el 25% i el 75% del vector
- \Rightarrow un element triat a l'atzar té una probabilitat del 50% de ser bo
- \Rightarrow **cal triar 2 elements de mitjana abans de trobar-ne un de bo**

Això dona

$$T(n) = T(3n/4) + O(n)$$

que implica que $T(n) \in O(n)$ en mitjana.

Esbós de l'algorithm

Algorithm:

- 1 Triar un element x a l'atzar
- 2 Dividir S respecte de x : S_E , S_x , S_D
- 3 Calcular selecció(S, k) recursivament

A quin ritme convergeix?

- diem que un element triat és **bo** si està entre el 25% i el 75% del vector
- \Rightarrow un element triat a l'atzar té una probabilitat del 50% de ser bo
- \Rightarrow **cal triar 2 elements de mitjana abans de trobar-ne un de bo**

Això dona

$$T(n) = T(3n/4) + O(n)$$

que implica que **$T(n) \in O(n)$ en mitjana.**

Exercici

Suposeu que fem servir l'algorisme anterior per trobar la mediana del vector

$$S : [\begin{array}{cccccccccc} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{array}].$$

Quantes vegades s'executa el bucle principal si l'element triat inicialment és $x = 5$? Per què?