Estructures de dades i algorismes

Tema 2 Algorismes de Divideix i Venç

Curs 2018-2019

Introducció

A diferència d'altres temes d'aquesta assignatura on veiem estructures de dades, en aquest tema veurem problemes que, encara que puguen resultar aparentment diferents, es caracteritzen per la forma o estratègia de resoldre'ls.

Objectius

- Conèixer de l'estratègia algorítmica divideix i venceràs (DyV).
- Estudi de dos algorismes d'ordenació basats en aquesta estratègia: mergesort i quicksort.
- Utilització dels conceptes vistos en quicksort per a resoldre eficientment el problema de la selecció (trobar el k-èssim menor element).
- Estudi d'altres algorismes clàssics basats en l'estratègia divideix i venceràs: multiplicació de grans enters, multiplicació de matrius, càlcul de la potència d'un nombre, etc.
- Saber calcular els costos d'aquest tipus d'algorismes.

L'aproximació DyV

L'aproximació divideix i venceràs

L'aproximació divideix i venceràs

- **DIVIDIR** el problema en subproblemes.
- **VÈNCER** (resoldre) els subproblemes de forma recursiva. Si aquests són d'una talla suficientment xicoteta, resoldre'ls de forma directa (cas base de la recursió).
- **COMBINAR** les solucions dels subproblemes per a obtenir la solució al problema original.

Exemple: cerca binària o dicotòmica

Donat un vector de talla n ordenat creixent, trobar x.

- Una cerca seqüencial té cost $O(n)$
- DyV: Aprofitant que el vector està ordenat $O(\log n)$.

```
/** 0<=inici<=fi<v.length */
public static <T extends Comparable<T>>
int cercaBinaria (T[] v, int inici, int fi, T x) {
  if (inici>fi) return -1;
  int meitat = (inici+fi)/2;
  int cmp    =  v[meitat].compareTo(x);
  if (cmp == 0) return meitat;
  if (cmp >  0) return cercaBinaria(v,inici,meitat-1,x);
  else          return cercaBinaria(v,meitat+1,fi,x);
}
```

- De vegades solament és necessari resoldre un subproblema, alguns autors denominen a aquest cas *redueix i venceràs*

Exemple: cerca binària o dicotòmica

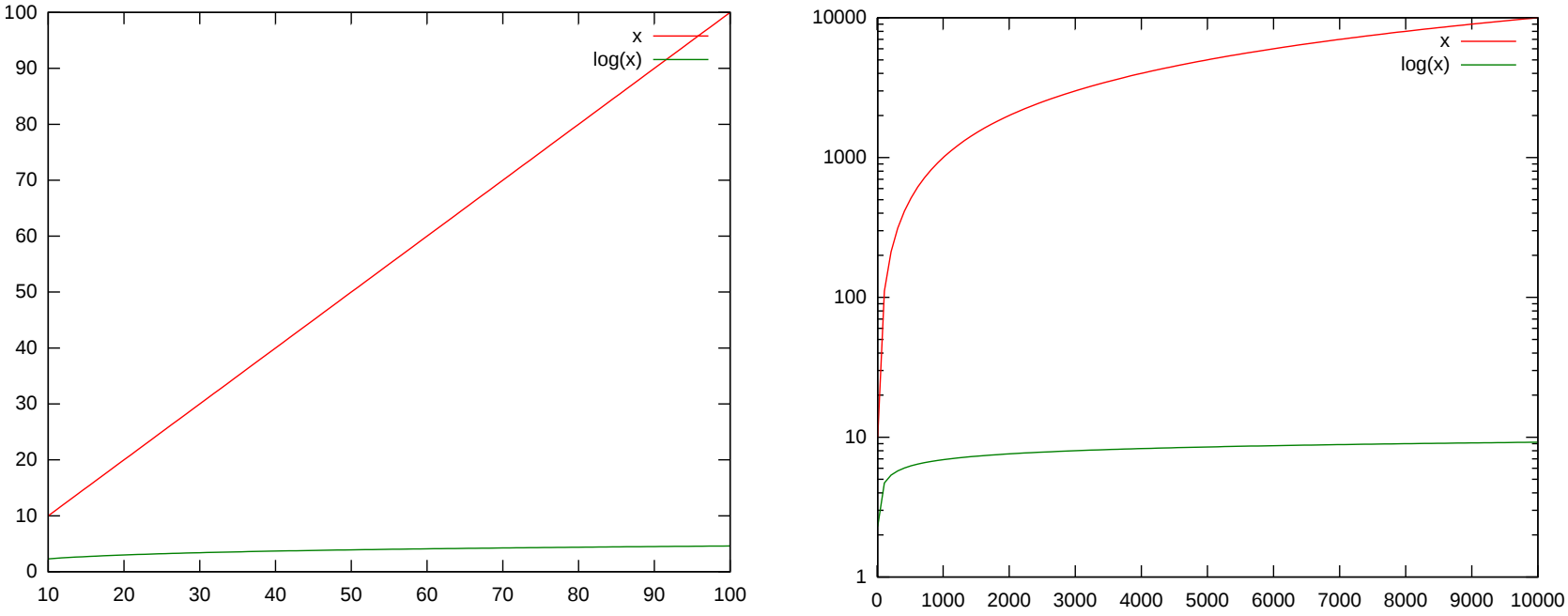

Figura: Comparació costos lineal i logarítmic

Exemple: cerca binària o dicotòmica

Exercici: escriu l'equació en recurrències de la cerca dicotòmica.

Exercici: ja saps que un algorisme recursiu que té una sola trucada recursiva al final es pot convertir en iteratiu (procés conegut com *tail recursion*). Escriu la versió iterativa de la cerca dicotòmica.

Exercici: ens donen una matriu quadrada amb valors sencers de manera que:

- Les files són seqüències creixents.
- Les columnes també són seqüències creixents.

Se t'ocorre una manera de cercar eficientment un valor en aquesta matriu?

Recordatori recursivitat: Torres de Hanoi

- Es disposa de tres torres i un cert nombre de discos de diferents grandàries.
- Inicialment, tots els discos es troben en la torre origen apilats en forma decreixent segons el seu diàmetre.
- Objectiu: desplaçar els discos a la torre destí:
- Els discos s'han de desplaçar un a un.
- No podem apilar un disc sobre un altre més xicotet.

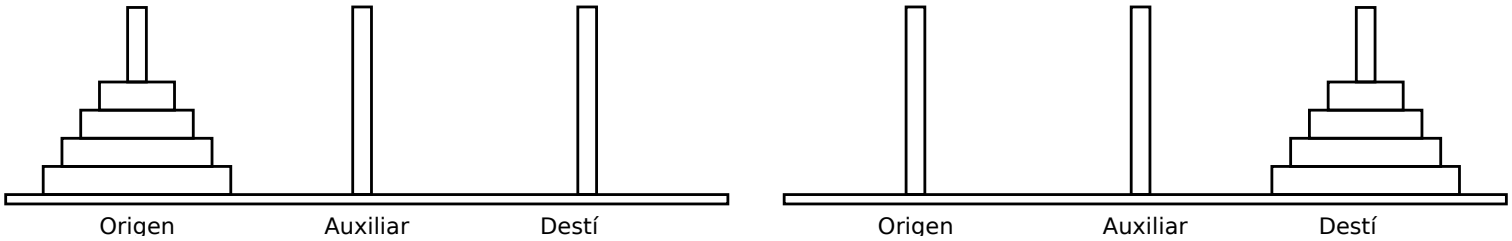

Figura: Torres de Hanoi

Recordatori recursivitat: Torres de Hanoi

```
void torresHanoi(int numDiscos,
    int origen, int auxiliar, int desti) {
    if (numDiscos==1)
        moverDisco(origen,desti);
    else {
        torresHanoi(numDiscos-1, origen, desti, auxiliar);
        moverDisco(origen,desti);
        torresHanoi(numDiscos-1, auxiliar, origen, desti);
  }
}
```

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

Que com veurem més endavant, té un **cost** $\Theta(2^n)$

Mergesort

Breu ressenya sobre algorismes d'ordenació

- Alguns algorismes que ja has estudiat, com és el cas d'**inserció directa** i de **selecció directa**, tenen un cost quadràtic en el cas pitjor (selecció directa sempre, inserció directa depenent de cada instància).
- Existeixen diversos algorismes d'ordenació que no estan basats en comparacions. Per exemple, l'algorisme **bucket sort** pot ordenar vectors de certs tipus en temps lineal.
- Un algorisme d'ordenació es denomina **estable** si preserva l'ordre relatiu original entre valors iguals.

Breu ressenya sobre algorismes d'ordenació

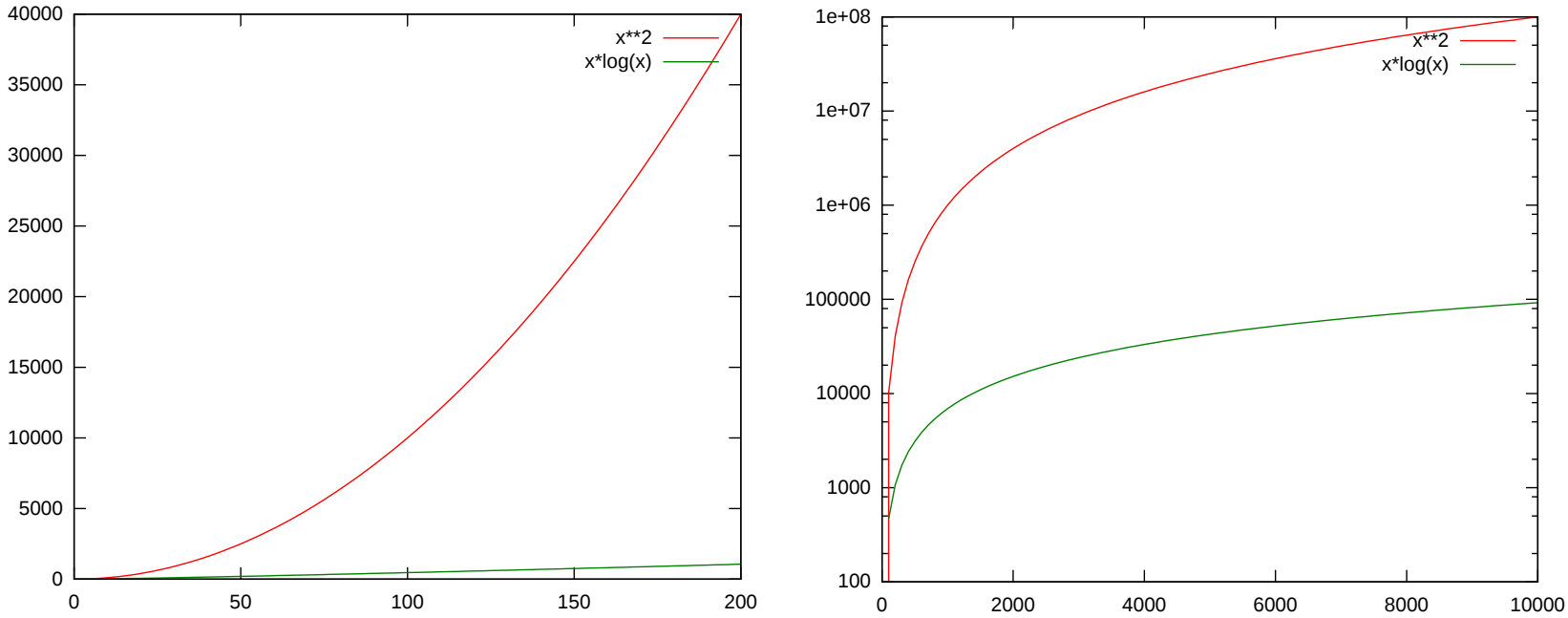

Figura: Comparació costos quadràtic i $O(n \log n)$

Breu ressenya sobre algorismes d'ordenació

Inserció directa

```
public static <T extends Comparable<T>>
  void insercioDirecta(T v[]) {
    for (int i = 0 ; i < v.length; i++) {
       T elem = v[i];
       int pIns = i;
       for (;pIns>0 && elem.compareTo(v[pIns-1])<0;pIns--)
            v[pIns] = v[pIns-1];
       v[pIns] = elem;
    }
}
```

Breu ressenya sobre algorismes d'ordenació

Selecció directa

```
public static <T extends Comparable<T>>
 void seleccioDirecta(T v[]) {
   for (int j = 0; j < v.length-1; j++) {
      int posmin = j; T valmin = v[j];
      for (int i=j+1; i < v.length; i++)
          if (v[i].compareTo(valmin)<0){
             valmin=v[i]; posmin=i;
          }
      v[posmin]=v[j]; v[j]=valmin;
   }
}
```

Estratègia del mergesort

- **Divideix** la seqüència de n elements a ordenar en dues subseqüències de $n/2$ elements cadascuna.
- **Vèncer:** Ordena les dues subseqüències recursivament utilitzant `mergesort`.

  La recursió finalitza quan la seqüència a ordenar conté un únic element, en aquest cas la seqüència ja està ordenada.

- **Combinar:** Combina les dues subseqüències ordenades per a generar la solució mitjançant `merge` (també conegut com a fusió o com a mescla natural).

Esquema de mergesort

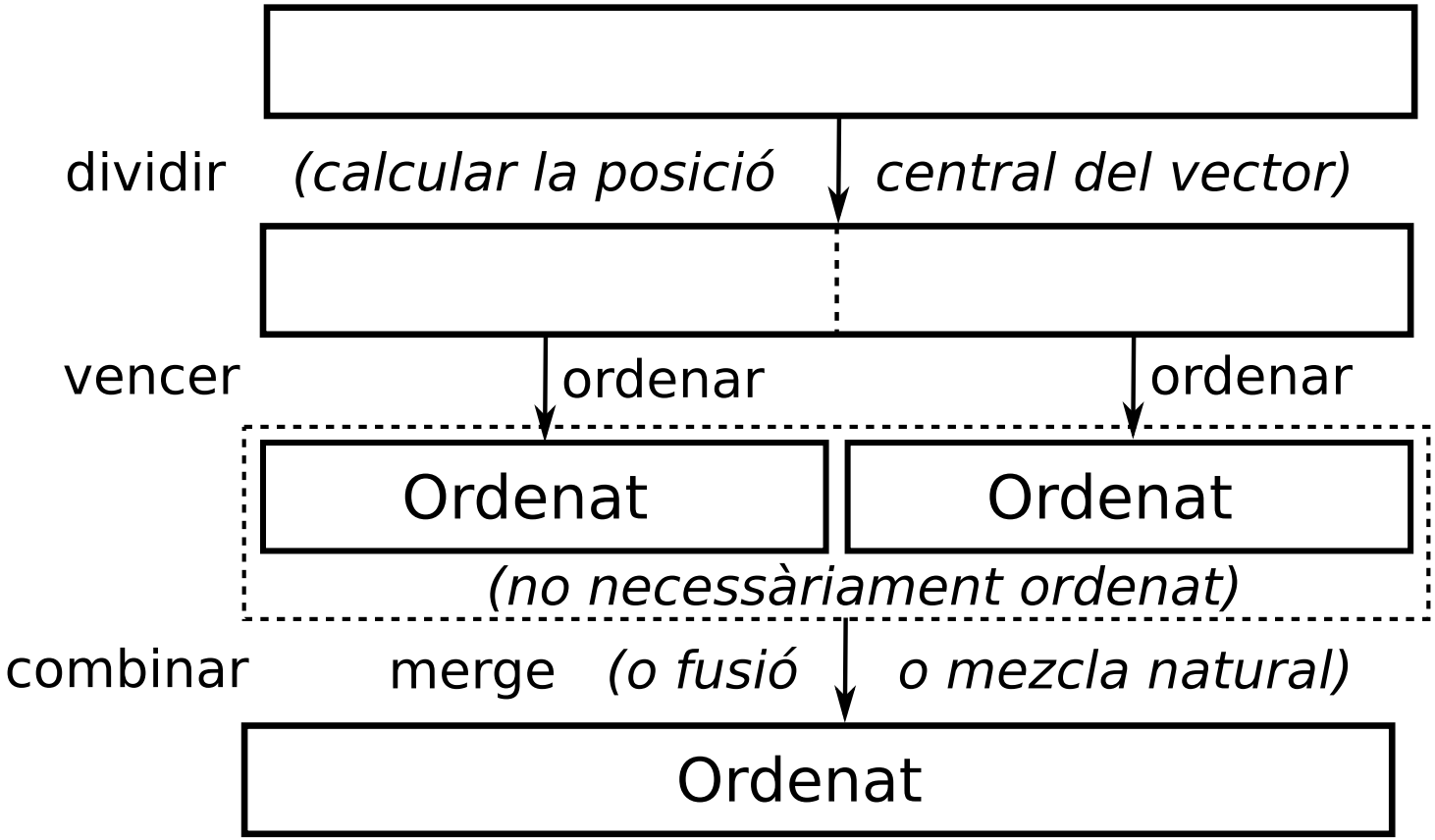

Figura: Esquema de mergesort

Exemple d'ordenació amb mergesort

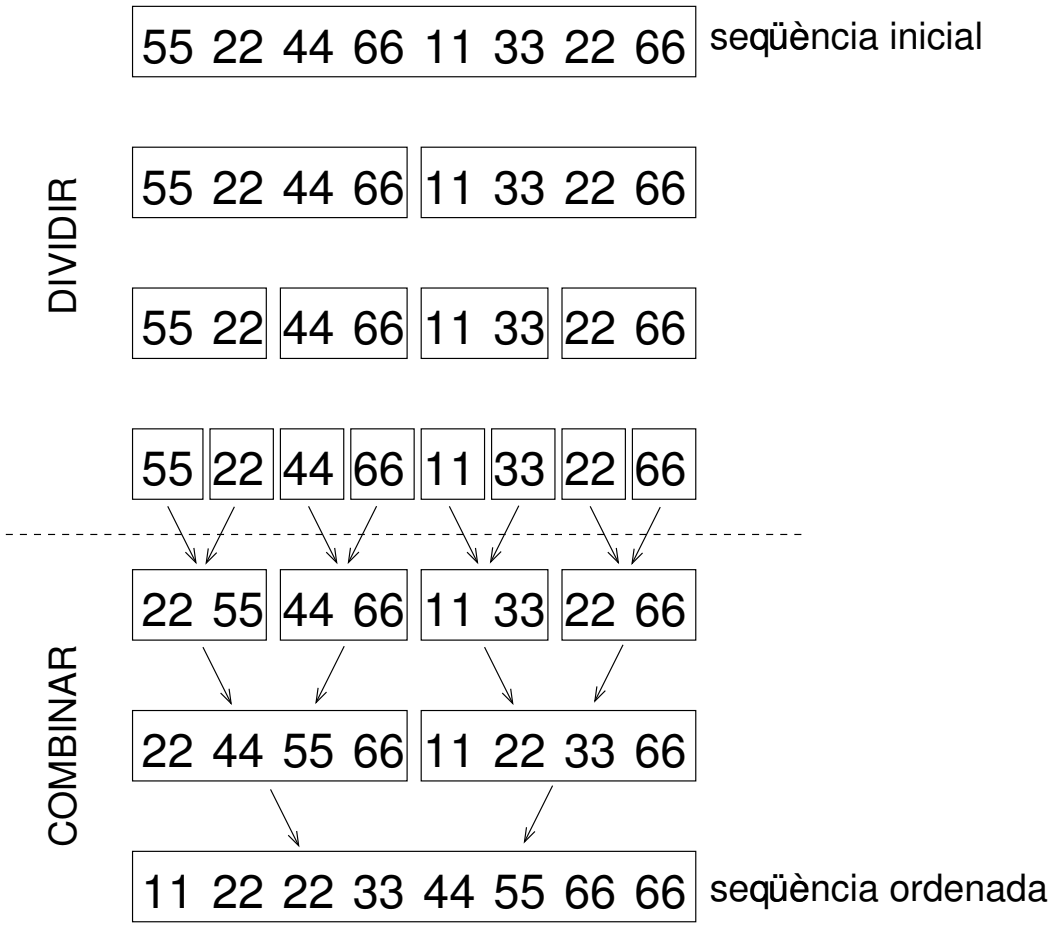

Figura: Exemple d'ordenació amb mergesort

Algorisme mergesort

```
public static <T extends Comparable<T>>
void mergeSort(T v[]){
  mergeSort(v, 0, v.length-1);
}

private static <T extends Comparable<T>>
void mergeSort(T v[], int esq, int dreta) {
  if (esq < dreta){
    int meitat = (esq+dreta)/2;
    mergeSort(v, esq, meitat);
    mergeSort(v, meitat+1, dreta);
    merge(v, esq, meitat+1, dreta);
  }
}
```

Estratègia del merge

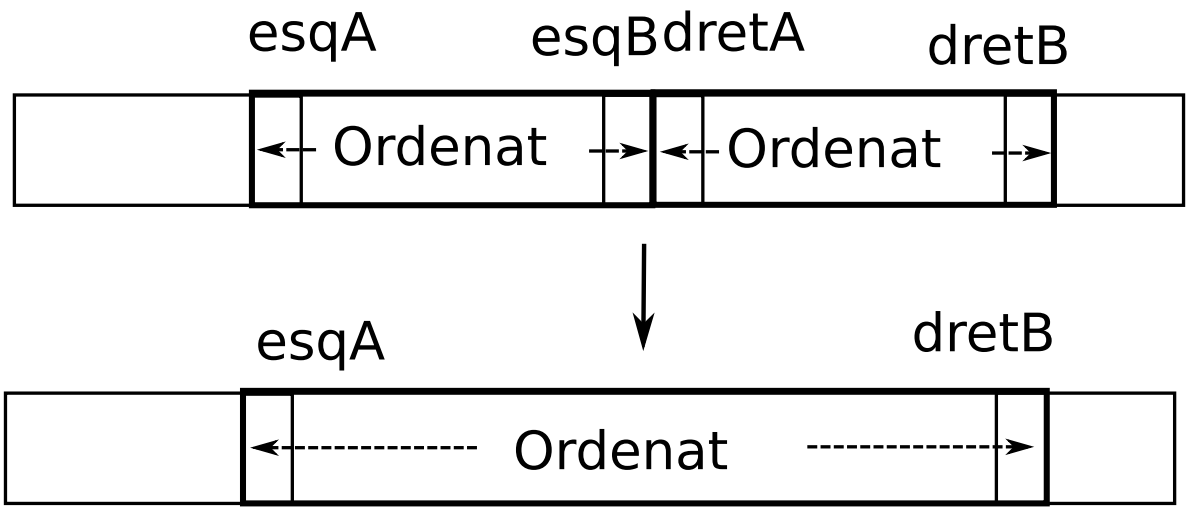

Figura: Barrejar ordenadament dues subseqüències ja ordenades

Algorisme merge

```
private static <T extends Comparable<T>>
void merge(T v[], int esqA, int esqB, int dretB){
    int i = esqA, dretA = esqB - 1, j = esqB, k = 0, r;
    T[] aux = (T[]) new Comparable[dretB-esqA+1];
    while ( i <= dretA && j <= dretB ) {
      if ( v[i].compareTo(v[j]) < 0 )
          aux[k] = v[i++];
      else
          aux[k] = v[j++];
      k++;
    }
    for(r = i; r <= dretA; r++) aux[k++] = v[r];
    for(r = j; r <= dretB; r++) aux[k++] = v[r];
    // tornem a copiar-ho tot al vector original:
    for(k=0, r=esqA; r<=dretB; r++, k++) v[r]=aux[k];
}
```

Mergesort: anàlisi

- **Dividir:** Calcula l'índex mitjà del vector: $\Theta(1)$.
- **Vèncer:** Resol recursivament els dos subproblemes, cadascun d'ells de talla $n/2$: $2T(n/2)$.
- **Combinar:** `merge` (fusió) d'un vector de n elements: $\Theta(n)$.

$$T(n) = \begin{cases} c_1, & \text{si } n \leq 1; \\ 2T(n/2) + c_2 n + c_3, & \text{si } n > 1 \end{cases}$$

Mergesort: anàlisi

$$
\begin{aligned}
T(n) &= 2T(n/2) + c_2 n + c_3 \\
&= 2(2T(n/2^2) + c_2 n/2 + c_3) + c_2 n + c_3 = \\
&= 2^2 T(n/2^2) + 2c_2 n + c_3(2+1) \\
&= 2^2(2T(n/2^3) + c_2 n/2^2 + c_3) + 2c_2 n + c_3(2+1) = \\
&= 2^3 T(n/2^3) + 3c_2 n + c_3(2^2+2+1) \\
&\vdots \quad \{i \text{ iteracions}\} \\
&= 2^i T(n/2^i) + ic_2 n + c_3(2^{i-1} + \ldots + 2^2 + 2 + 1)
\end{aligned}
$$

És a dir:

$$
\begin{aligned}
T(n) &= \{n/2^i = 1, i = \log n\}; \sum_{j=0}^{i-1} 2^j = 2^i - 1 \\
&= nc_1 + c_2 n \log n + c_3(n-1) \in \Theta(n \log n)
\end{aligned}
$$

Divisió equilibrada de subproblemes

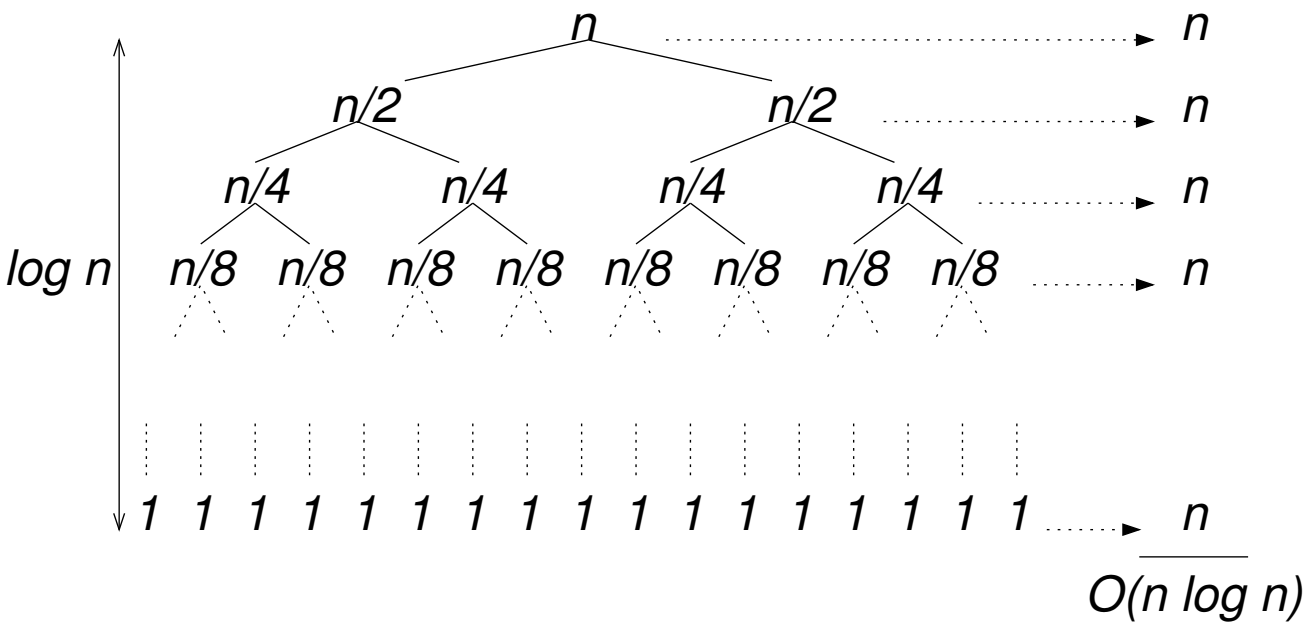

Figura: Divisió equilibrada de subproblemes

$$T(n) = \begin{cases} c_1, & \text{si } n \leq 1; \\ 2T(n/2) + c_2 n + c_3, & \text{si } n > 1 \end{cases}$$

Mergesort amb divisió no equilibrada

```
private static <T extends Comparable<T>>
void mergeSortBad(T v[], int esq, int dret) {
  if (esq < dret){
    mergeSortBad(v, esq, esq);
    mergeSortBad(v, esq+1, dret);
    merge(v, esq, esq+1, dret);
  }
}
```

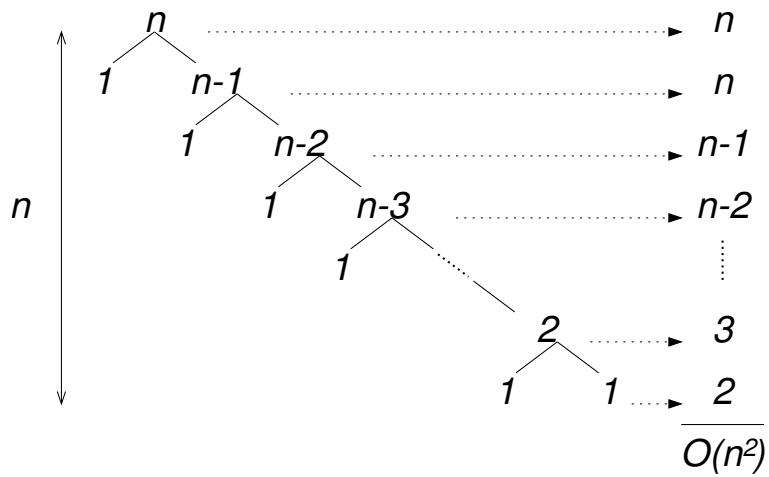

Figura: Divisió no equilibrada

Mergesort amb divisió no equilibrada

Equacions de recurrència:

$$T(n) = \begin{cases} c_1, & \text{si } n \leq 1; \\ T(1) + T(n-1) + c_2 n + c_3, & \text{si } n > 1 \end{cases}$$

$$T(n) = \begin{cases} c_1, & \text{si } n \leq 1; \\ T(n-1) + c_2 n + c_4, & \text{si } n > 1 \end{cases}$$

Cost: $\Theta(n^2)$

Possibles millores de mergesort

- El Mergesort pot ser implementat de manera **estable** sense gran dificultat.
- Un dels inconvenients és que es necessita un espai $2n$ a causa del `merge`.
  - Existeix un algorisme de fusió que no utilitza un vector addicional, però amb una constant molt alta que no ho fa apropiat.
- Un altre inconvenient és la talla de la pila de recursió: la màxima profunditat de la pila serà proporcional a $\log n$.
- Una millora consisteix en no realitzar cridades recursives quan la talla del vector és xicoteta, fent una ordenació per inserció o selecció directa.
- Es poden evitar moviments redundants de dades fent cridades alternativament sobre el vector original i el vector auxiliar.

Quicksort

Algorisme Quicksort

Quicksort, igual que mergesort, es basa en l'estratègia DyV. La diferència estreba que la part costosa de l'algorisme està en el pas de dividir. Els tres passos de la recursió per a ordenar un subvector $A[p..r]$ són:

- **Dividir:** El vector $A[p..r]$ es particiona (reorganitza) en dos subvectors $A[p..q]$ i $A[q+1..r]$, de manera que els elements de $A[p..q]$ són menors o iguals que els de $A[q+1..r]$. L'índex q es calcula també en el procediment de partició.
- **Vèncer:** Els dos subvectors $A[p..q]$ i $A[q+1..r]$ s'ordenen recursivament utilitzant quicksort.
- **Combinar:** Com els subvectors s'ordenen en el seu lloc corresponent, no cal fer gens per a combinar les solucions: el vector complet $A[p..r]$ ja està ordenat.

Esquema de quicksort

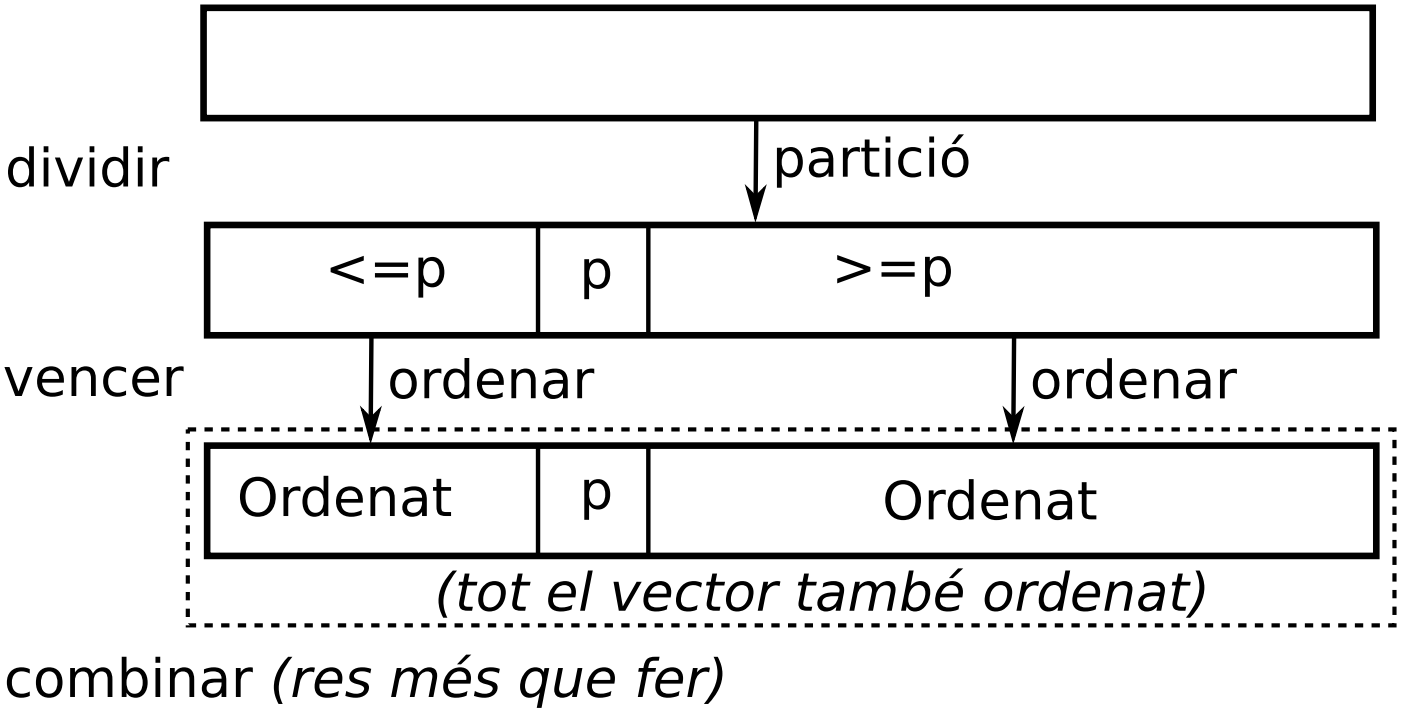

Figura: Esquema de quicksort

Exemple de quicksort

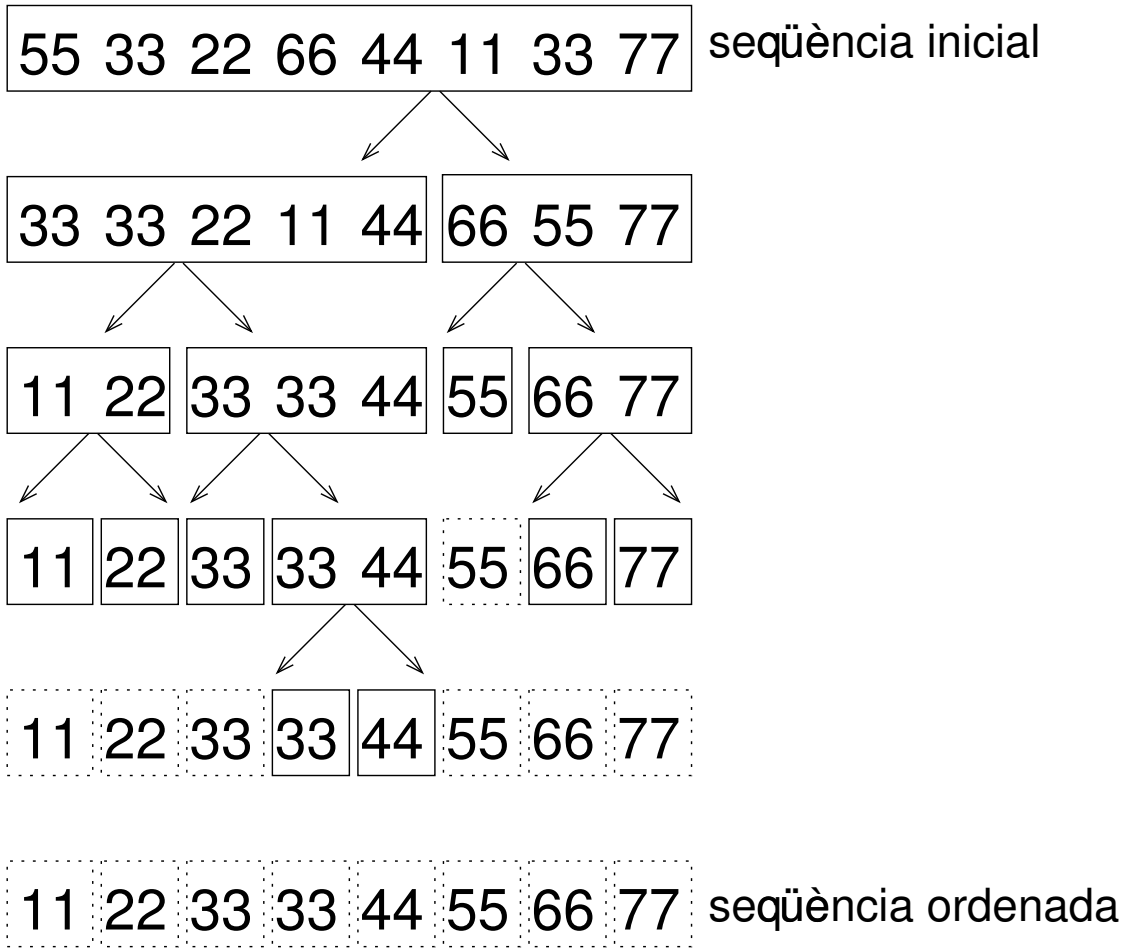

Figura: Exemple de quicksort

Algorisme quicksort

```
public static <T extends Comparable<T>>
void quickSort(T v[]) {
  quickSort(v, 0, a.length - 1);
}
private static <T extends Comparable<T>>
void quickSort(T v[], int esq, int dret ) {
  if (esq < dret) {
    int indexP = particio(v, esq, dret);
    quickSort(v, esq, indexP);
    quickSort(v, indexP + 1, dret);
  }
}
```

Algorisme partició

Objectiu: reorganitzar el vector $A[p..r]$ deixant en $A[p..q]$ elements menors o iguals que en $A[q+1..r]$, de manera que les particions resulten el més equilibrades possible.

Idea: prendre un valor anomenat **pivot** i moure els elements necessaris perquè els elements menors que el pivot queden a l'esquerra i els majors que el pivot a la dreta.

És clar que el valor ideal per al pivot és **la mitjana** del vector, però açò resulta massa car de calcular, així que normalment s'utilitza:

- El primer element del vector.
- La mitjana de 3 elements (primer, últim, central) o de més de 3 elements.
- Pivot aleatori.

Algorisme partició

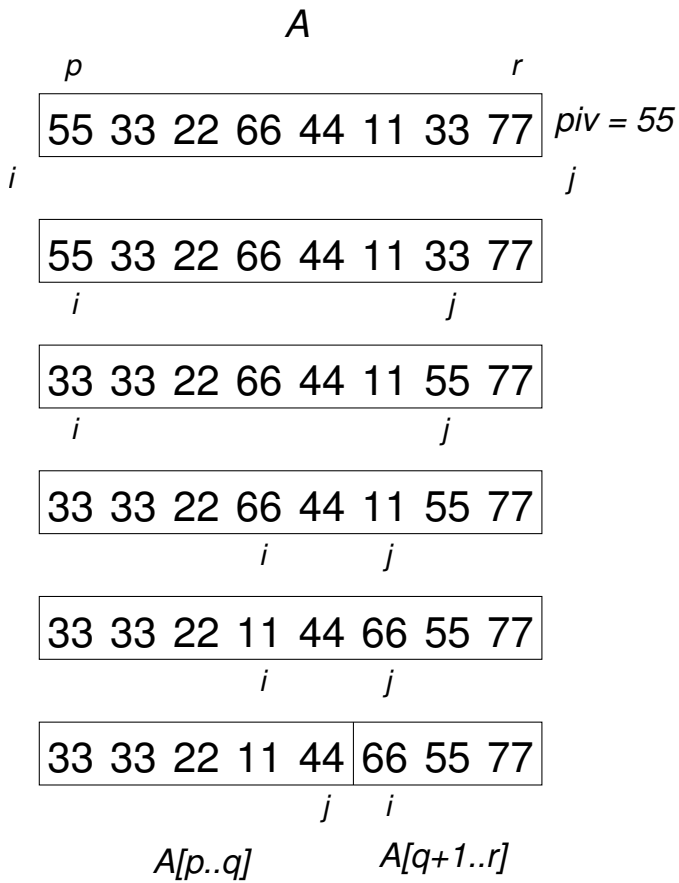

Figura: Exemple de partició

Algorisme partició

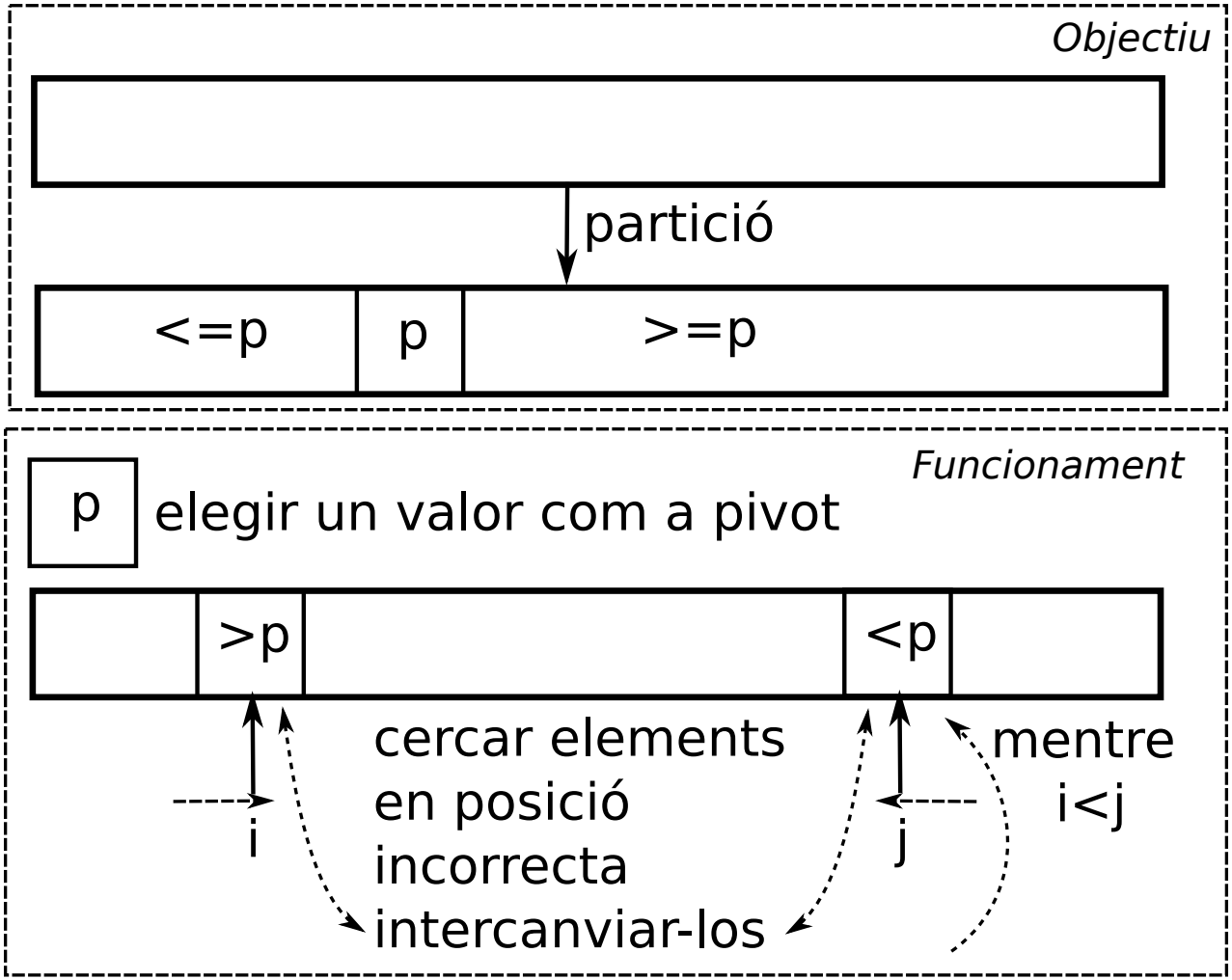

Figura: Esquema algorisme partició

Algorisme partició

Observa que aquest algorisme retorna l'índex on queda dividit el vector (no hi ha garantia que el vector quede partit en 2 parts (iguals)):

```
// precondicio: esq<dret
private static <T extends Comparable<T>>
int particio(T v[], int esq, int dret ) {
  T pivot = mediana3(a,esq,dret);
  int i=esq;
  int j=dret-1;
  while (i<j) {
    while(pivot.compareTo(v[++i])>0); // pivot > v[++i]
    while(pivot.compareTo(v[--j])<0); // pivot < v[--j]
    intercanviar (v,i,j);
  }
  intercanviar (v,i,j);      // desfer l'ultim canvi
  intercanviar (v,i,dret-1); // restaurar el pivot
  return i;
}
```

Cost: $\Theta(n)$

Algorisme partició

```
// Metode per a intercanviar dos elements d'un array
private static <T>
void intercanviar(T v[], int ind1, int ind2) {
  T tmp   = v[ind1];
  v[ind1] = v[ind2];
  v[ind2] = tmp;
}
// calcul de la Mediana de 3, torna el pivot
private static <T extends Comparable<T>>
T mediana3(T v[], int esq, int dret) {
  int mid=(esq+dret)/2;
  if (v[mid].compareTo(v[esq])<0) intercanviar(v,esq,mid);
  if (v[dret].compareTo(v[esq])<0) intercanviar(v,esq,dret);
  if (v[dret].compareTo(v[mid])<0) intercanviar(v,mid,dret);
  // ocultar el pivot en la posicio dret-1
  intercanviar(v,mid,dret-1);
  return v[dret-1];
}
```

Detall de partició amb mitjana de 3

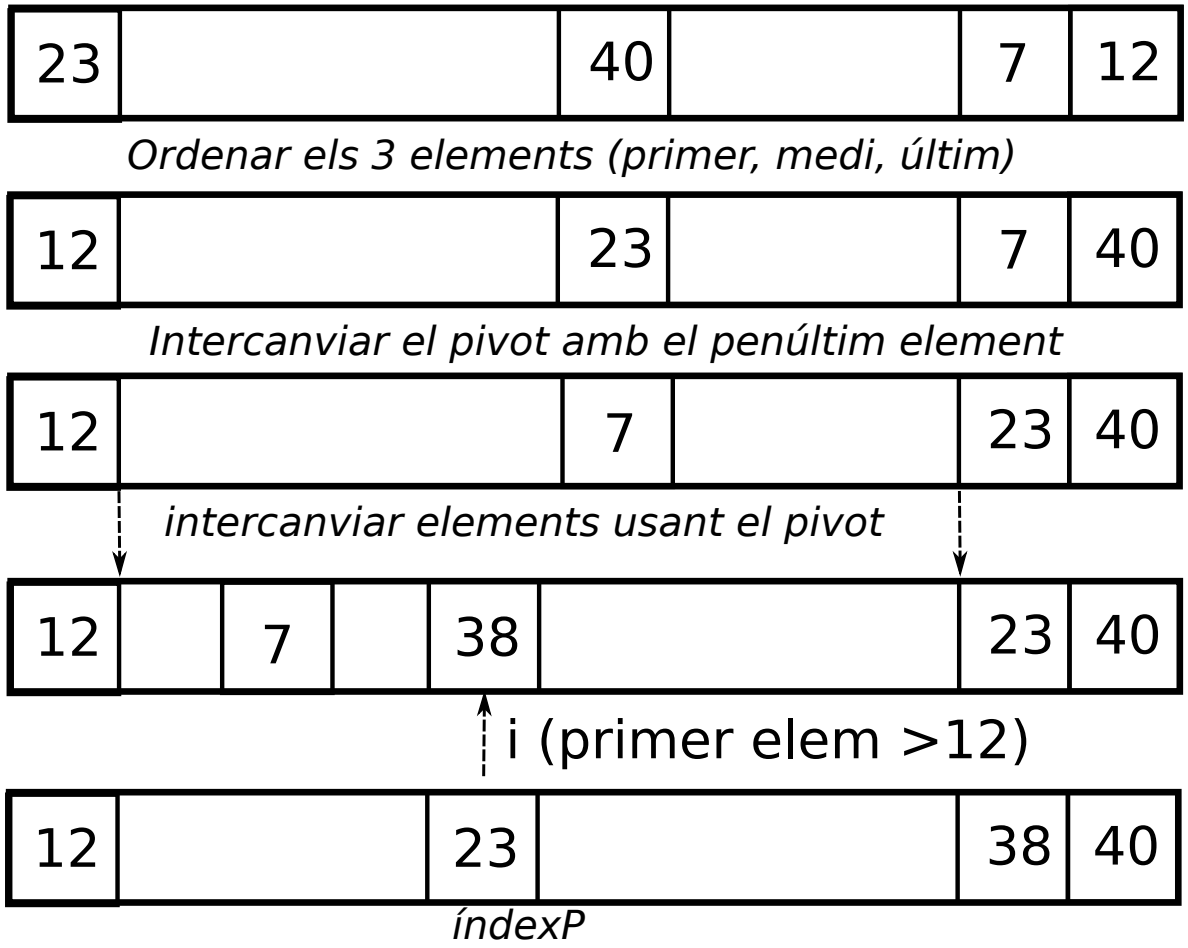

Figura: Detall de partició amb 3 elements

Anàlisi del cost de quicksort

Siga n el nombre d'elements del vector. El cost de partició és $O(n)$ i reorganitza el vector en dos subvectors de grandària i i $(n-i)$. La relació de recurrència és:

$$T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ T(i) + T(n-i) + k'n, & \text{si } n > 1 \end{cases}$$

Anàlisi del cost de quicksort (cas pitjor)

Cas pitjor: *(Partició desequilibrada)* suposem que tots els elements són diferents i ja estan ordenats. En aquest cas, partició divideix el vector original en dos subvectors: un amb un únic element (el mínim, que ha actuat com a pivot), mentre que l'altre conté la resta d'elements.

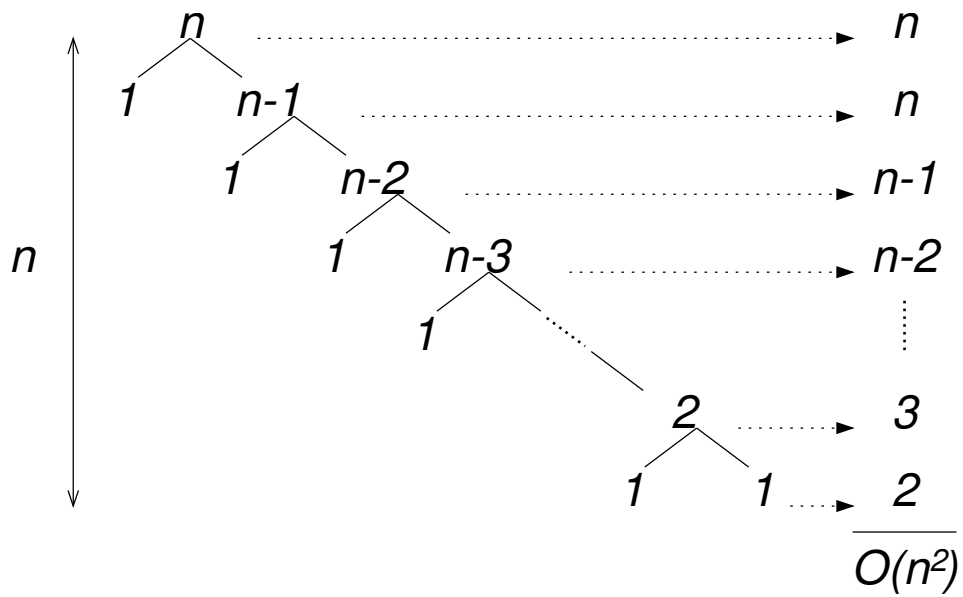

Figura: Divisió no equilibrada

Anàlisi del cost de quicksort (cas pitjor)

$$T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ T(1) + T(n-1) + k'n, & \text{si } n > 1 \end{cases}$$

Cost: $O(n^2)$

- No és tan infreqüent ordenar un vector que pot estar, almenys en part, ja ordenat.
- Un disseny descurat de partició pot causar el mateix problema si tots els elements del vector són iguals. Açò també és molt problemàtic perquè no és gens infreqüent: En ordenar un vector d'un milió d'elements, pot ser que si existeixen (posem) 5000 iguals queden junts en algun moment de l'algorisme. En aqueix cas, el cost d'ordenar aquest subvector amb cost quadràtic pot dominar la resta.

Anàlisi del cost de quicksort (cas millor)

El cas millor serà aquell en el qual en cada trucada a partition deixa les dues parts equilibrades.

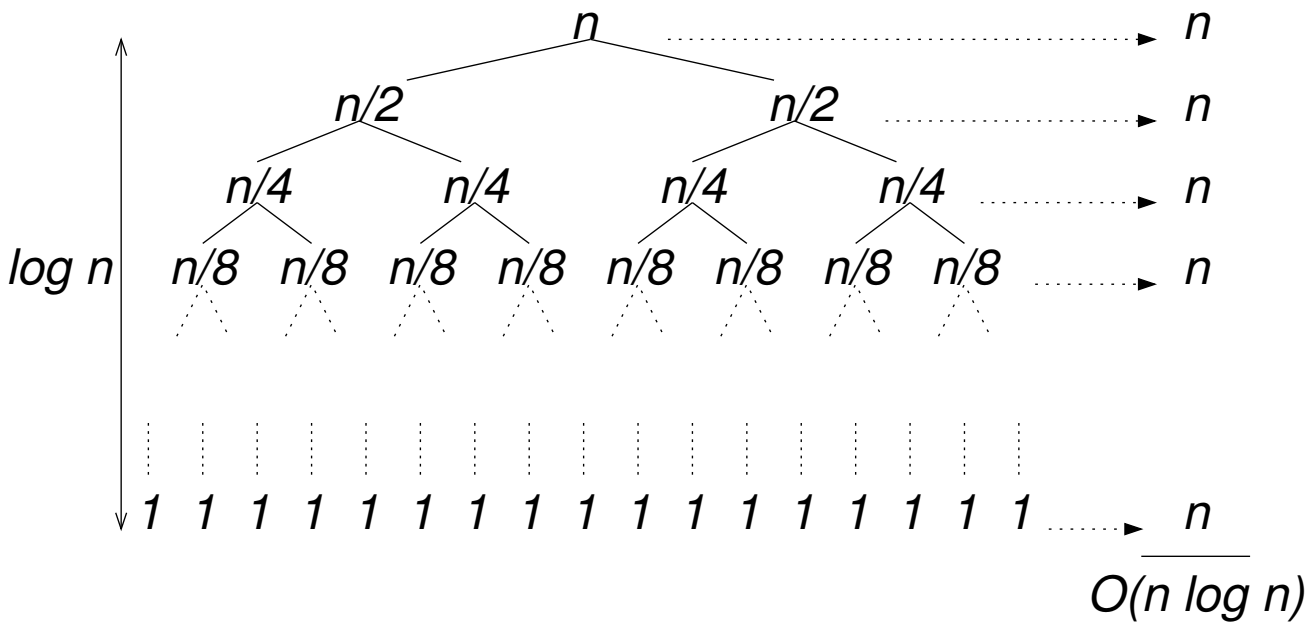

Figura: Divisió equilibrada de subproblemes

Anàlisi del cost de quicksort (case mitjana)

$$
\begin{aligned}
T(n) = k'n &+ \frac{1}{n}(T(1) + T(n-1)) \\
&+ \frac{1}{n}(T(2) + T(n-2)) \\
&+ \frac{1}{n}(T(3) + T(n-3)) \\
&\vdots \\
&+ \frac{1}{n}(T(n-1) + T(1)) \\
= k'n &+ \frac{2}{n}\sum_{i=1}^{n-1} T(i)
\end{aligned}
$$

$$
T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ k'n + 2/n \sum_{i=1}^{n-1} T(i), & \text{si } n > 1 \end{cases}
$$

Anàlisi del cost de quicksort (case mitjana)

$$T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ k'n + 2/n \sum_{i=1}^{n-1} T(i), & \text{si } n > 1 \end{cases}$$

Es pot demostrar que, $\forall n \geq 2,\ T(n) \leq k'' n \log n$, sent $k'' = 2k' + k$. Per tant, el cost mitjana és $O(n \log n)$.

Cas $n = 2$

$$T(2) = 2k' + \sum_{i=1}^{1} T(i) = 2k' + k \leq (2k' + k)2 \log 2$$

Cas $n > 2$

$$T(n) = k'n + 2/n \sum_{i=1}^{n-1} T(i) = k'n + 2/n T(1) + 2/n \sum_{i=2}^{n-1} T(i)$$

$$(H.I.) \leq k'n + 2k/n + 2/n \sum_{i=2}^{n-1} k'' i \log i \leq k'' n \log n \in O(n \log n)$$

Possibles millores de quicksort

- No realitzar cridades recursives quan la talla del vector és xicoteta, canviar a inserció o selecció directa.
- Triar el valor utilitzat com a pivot de forma aleatòria entre tots els valors del vector. Per a açò, abans de realitzar cada partició, s'intercanviaria el valor de l'element utilitzat com a pivot amb el valor de l'element seleccionat a l'atzar.
- Reemplaçar una de les 2 trucades recursives per un bucle (*tail recursion*). Si açò es fa bé (sempre a la meitat més xicoteta) es pot garantir una profunditat màxima de la pila de trucades recursives $O(\log n)$.
- És possible ordenar solament la part esquerra del vector si únicament volem els primers menors elements (anàlogament per als majors). Açò, ben fet, serveix per a implementar un cas particular de cua de prioritat on no hi ha insercions excepte un vector inicial.

Selecció

Selecció (cerca del k-èssim menor element)

Problema: Donat un vector de n elements, el problema de la selecció consisteix a cercar el k-èssim menor element.

Solució directa: ordenar els n elements i accedir al k-èssim. Cost: $O(n \log n)$.

Solució DyV: És possible trobar un algorisme més eficient? Utilitzant la idea de l'algorisme partició:

- **Dividir (partition):** El vector $A[p..r]$ es particiona (reorganitza) en dos subvectors $A[p..q]$ i $A[q+1..r]$, de manera que els elements de $A[p..q]$ són menors o iguals que el pivot i els de $A[q+1..r]$ són majors o iguals.
- **Vèncer:** Cerquem en el subvector corresponent fent cridades recursives a l'algorisme.
- **Combinar:** Si $k \leq q$, llavors el k-èssim menor estarà en $A[p..q]$. Si no, estarà en $A[q+1..r]$.

Algorisme selecció recursiu

```
public static <T extends Comparable<T>>
T seleccion(T v[], int k) {
  return seleccion(v,0,v.length-1,k-1);
}
private static <T extends Comparable<T>>
T seleccion(T v[], int p, int r, int k) {
  if (p == r)
    return v[k];
  else {
    int q = particion(v, p, r);
    if (k <= q)
      return seleccion(v, p, q, k);
    else
      return seleccion(v, q+1, r, k);
  }
}
```

Algorisme selecció iteratiu

Apliquem *tail recursion*:

```
private static <T extends Comparable<T>>
T seleccion(T v[], int p, int r, int k) {
  while (p < r) {
    int q = particion(v, p, r);
    if (k <= q)
      r = q;
    else
      p = q + 1;
  }
  return v[p];
}
```

Exercici: Realitza una traça de selecció recursiu i iteratiu amb $A = \{31, 23, 90, 0, 77, 52, 49, 87, 60, 15\}$ i $k = 7$.

Anàlisi de l'algorisme selecció

Cas pitjor: Vector ordenat de forma no decreixent i cerquem l'element major ($k = n$). Cost: $O(n^2)$.

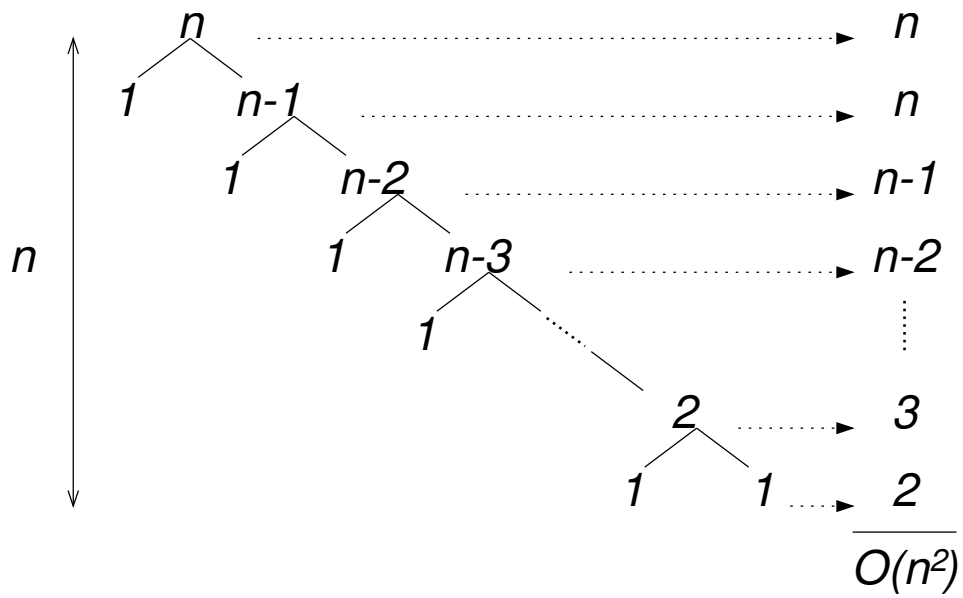

Figura: Cas pitjor selecció

$$T(n) = \begin{cases} c, & \text{si } n \leq 1; \\ T(n-1) + c'n, & \text{si } n > 1 \end{cases}$$

Anàlisi de l'algorisme selecció

Cas millor: L'element a cercar és el menor ($k = 1$) o major ($k = n$) i aquest actua com a pivot $\rightarrow$ Una única trucada a partició. Cost: $O(n)$.

Case mitjana: amb certes assumpcions de aleatorietat, l'algorisme selecció té un cost $\in \Theta(n)$ (pàg. 188, Cormen 90), (pàg. 167, Horowitz 98).

Suposem que en cada trucada a partició, el problema es redueix a la meitat. Relació de recurrència:

$$T(n) = \begin{cases} c, & \text{si } n \leq 1; \\ T(n/2) + c'n, & \text{si } n > 1 \end{cases}$$

Anàlisi de l'algorisme selecció

$$T(n) = \begin{cases} c, & \text{si } n \leq 1; \\ T(n/2) + c'n, & \text{si } n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= T(n/2) + c'n \\
&= (T(n/2^2) + c'n/2) + c'n = T(n/2^2) + (n + n/2)c' \\
&= (T(n/2^3) + c'n/2^2) + (n + n/2)c' = T(n/2^3) + n(1 + 1/2 + 1/2^2)c' \\
&\dots \\
&= T(n/2^i) + c'n \sum_{j=0}^{i-1} 1/2^j
\end{aligned}$$

//Serie geomètrica: $\sum_{j=0}^{i} x^i = \frac{x^{i+1} - 1}{x - 1}$; // $\sum_{j=0}^{i-1} 1/2^j = \frac{1/2^i - 1}{1/2 - 1} = \frac{2(2^i - 1)}{2^i}$

$$\begin{aligned}
&= T(n/2^i) + n\frac{2(2^i - 1)}{2^i}c' \qquad \{n/2^i = 1, i = \log n\} \\
&= T(1) + n\frac{2(n-1)}{n}c' = c_1 + 2(n-1)c' \in O(n)
\end{aligned}$$

Altres problemes

Multiplicació de matrius

Problema: Multiplicació de matrius quadrades $n \times n$: $C = A \times B$.

Algorisme directe: $\Theta(n^3)$

```
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++) {
    C[i][j] = 0;
    for(k = 0; k < n; k++)
      C[i][j] += A[i][k] * B[k][j];
  }
```

Multiplicació per blocs: recursiu dividint la matriu

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{12}b_{22} \end{pmatrix}$$

$$T(n) = \begin{cases} k, & \text{si } n = 1; \\ 8\,T(n/2) + k'n^2, & \text{si } n > 1 \end{cases}$$

$n = 2$: 8 productes i 4 sumes de $n/2 \times n/2 \rightarrow \Theta(n^3)$

Multiplicació de matrius: algorisme de Strassen

Cadascuna d'aquestes operacions necessita una única multiplicació:

$$
\begin{aligned}
m_1 &= (a_{11} + a_{22}) \cdot (b_{11} + b_{22}) \\
m_2 &= (a_{21} + a_{22}) \cdot b_{11} \\
m_3 &= a_{11} \cdot (b_{12} - b_{22}) \\
m_4 &= a_{22} \cdot (b_{21} - b_{11}) \\
m_5 &= (a_{11} + a_{12}) \cdot b_{22} \\
m_6 &= (a_{21} + a_{11}) \cdot (b_{11} + b_{12}) \\
m_7 &= (a_{12} + a_{22}) \cdot (b_{21} + b_{22})
\end{aligned}
$$

$$
C = \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}
$$

7 productes i 18 sumes de $n/2 \times n/2$

Multiplicació de matrius: algorisme de Strassen

7 productes i 18 sumes de $n/2 \times n/2 \rightarrow \Theta(n^{\log_2 7}) = \Theta(n^{2.807})$

(Les matrius es poden sumar i restar en un temps $\Theta(n^2)$)

$$T(n) = \begin{cases} k, & \text{si } n = 1; \\ 7T(n/2) + k'n^2, & \text{si } n > 1 \end{cases}$$

Exercici: Verifica que el producte matricial $A \times B$ està donat per la matriu anterior C. Demostra que $T(n) \in \Theta(n^{2.807})$.

Multiplicació de matrius: algorisme de Strassen

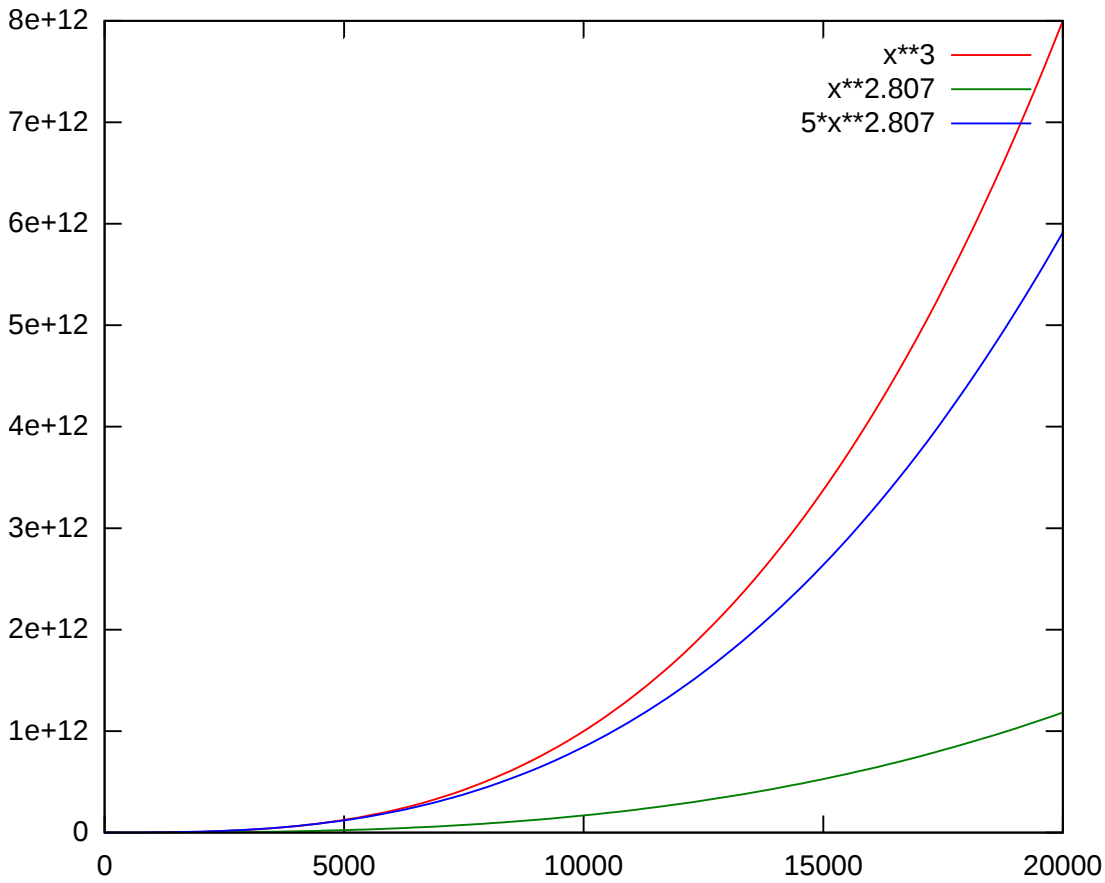

Figura: Comparació costos producte matrius

Càlcul de la potència sencera

Volem calcular la potència a^n sent n un nombre enter no negatiu i sent a un valor que compleix la propietat associativa.

Un esquema iteratiu $a^n = a \times a \times \cdots \times a$ té un cost $\Theta(n)$

Aprofitant la propietat $a^n \times a^m = a^{(n+m)}$ podem fer:

```
double elevar(double a, int n) {
  if (n==0) return 1;
  if (n==1) return a; // no fa falta
  double aux = elevar(a,n/2);
  if (n%2==0)
    return aux*aux;
  return aux*aux*a;
}
```

$$T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ T(n/2) + k', & \text{si } n > 1 \end{cases}$$

Cost $\Theta(\log n)$

Producte de grans enters

Siguen x i y dos enters representats cadascun amb una cadena de n dígits en base B. El càlcul del producte xy de manera *tradicional* requereix un nombre d'operacions elementals de l'ordre $\Theta(n^2)$, vegem la seua descomposició recursiva. Siga $m < n$

$$\begin{aligned} x &= x_1 B^m + x_0 \\ Y &= Y_1 B^m + y_0 \end{aligned}$$

sent x_0 i y_0 menors que B^m, el producte és:

$$xy = (x_1 B^m + x_0)(y_1 B^m + y_0) = z_2 B^{2m} + z_1 B^m + z_0$$

on el càlcul de z_0, z_1, z_2 requereix 4 productes en total:

$$\begin{aligned} z_2 &= x_1 y_1 \\ z_1 &= x_1 y_0 + x_0 y_1 \\ z_0 &= x_0 y_0 . \end{aligned}$$

Producte d'enters: Algorisme Karatsuba

És a dir, aquesta descomposició recursiva té aquesta equació de recurrències (per a $m = n/2$):

$$T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ 4\,T(n/2) + O(n), & \text{si } n > 1 \end{cases}$$

que correspon a un cost $\Theta(n^2)$

En 1960 Anatolii Alexeevitch Karatsuba va observar que z_1 es podia resoldre amb un sol producte (a costa de fer més summes/restes, però aquestes tenen cost lineal): $z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$

$$T(n) = \begin{cases} k, & \text{si } n \leq 1; \\ 3\,T(n/2) + O(n), & \text{si } n > 1 \end{cases}$$

que correspon a un cost $\Theta(n^{\log_2 3}) = \Theta(n^{1.59})$

Producte d'enters: Algorisme Karatsuba

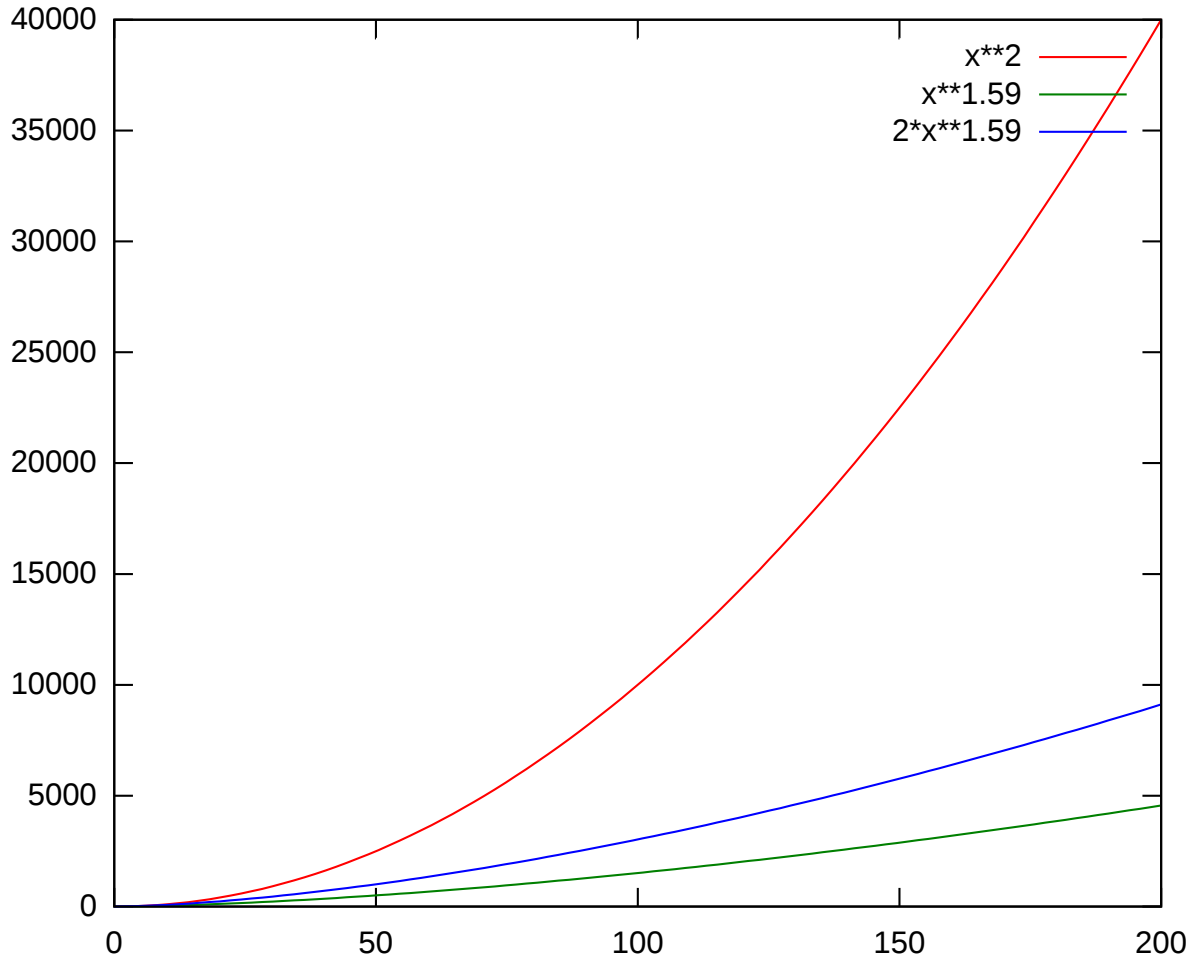

Figura: Comparació costos producte grans enters

Complexitat temporal

Complexitat temporal DyV: recurrència divisòria

Anem a resoldre un problema T per a una instància de talla n mitjançant DyV:

- **Dividir** el problema original de talla n en a subproblemes de grandària n/b. Siga el cost d'aquesta divisió $D(n)$.
- **Resoldre** els a subproblemes de grandària n/b. Aquest cost és $aT(n/b)$.
- **Combinar** els subproblemes per al problema original. Siga aquest cost $C(n)$.

Exemple de l'algorisme mergesort:

- **Dividir** és calcular l'índex mitjà del vector. Cost $\Theta(1)$.
- **Resoldre** en 2 subproblemes de talla $n/2$.
- **Combinar** és l'operació merge (fusió) amb cost $\Theta(n)$.

Complexitat temporal DyV: recurrència divisòria

Relació de recurrència divisòria general:

$$T(n) = \begin{cases} c, & \text{si } n \leq n_0; \\ aT(n/b) + D(n) + C(n), & \text{si } n > n_0 \end{cases}$$

Quan $D(n) + C(n)$ és $\Theta(n^k)$ tenim aquest cas particular:

$$T(n) = \begin{cases} c, & \text{si } n \leq n_0; \\ aT(n/b) + \Theta(n^k), & \text{si } n > n_0 \end{cases}$$

Sent n_0 la talla per sota de la qual apliquem el cas base de la recursió.

Complexitat. temporal DyV: recurrència substractora

Les trucades recursives des d'una talla n són a subproblemes de talla $n - c$, la qual cosa correspon a aquesta equació de recurrència:

$$T(n) = \begin{cases} k, & \text{si } n \leq n_0; \\ aT(n-c) + g(n), & \text{si } n > n_0 \end{cases}$$

Exemples:

- Ordenar amb selecció directa fa una sola trucada a talla $n-1$ i $g(n) = \Theta(n)$, cost $\Theta(n^2)$.
- Sumar un vector de manera recursiva amb trucades a talla $n-1$ té cost lineal.
- Torres de Hanoi, 2 trucades a talla $n-1$ i $g(n) = O(1)$, cost $\Theta(2^n)$.
- Mergesort desequilibrat, tenia cost $\Theta(n^2)$.

Complexitat temporal DyV: teoremes mestres

Fins ara hem calculat el cost d'un algorisme recursiu utilitzant el mètode *de la substitució*, vegem un parell de teoremes mestres molt útils per a multitud de casos típics.

Teorema mestre per a recurrència divisora: La solució a l'equació $T(n) = aT(n/b) + \Theta(n^k)$, amb $a \geq 1$ i $b > 1$, és:

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$$

Complexitat temporal DyV: teoremes mestres

Exemple *(cerca dicotòmica)* $a = 1, b = 2, k = 0.$ $\quad T(n) \in O(\log n)$

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

Exemple *(algorisme selecció)* $a = 1, b = 2, k = 1.$ $\quad T(n) \in O(n)$

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

Complexitat temporal DyV: teoremes mestres

Exemple *(algorismes mergesort, quicksort)* $a = 2, b = 2, k = 1$.
$T(n) \in O(n \log n)$

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

Exemple: *(producte d'enters grans algorisme Karatsuba)*
$a = 3, b = 2, k = 1$. $T(n) \in O(n^{\log_2 3}) = O(n^{1.59})$

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

Complexitat temporal DyV: teoremes mestres

Exemple: *(producte d'enters grans convencional)* $a = 4, b = 2, k = 1$.
$T(n) \in O(n^2)$

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

Exemple: *(algorisme Strassen)* $a = 7, b = 2, k = 2.$ $\quad T(n) \in O(n^{2.807})$

$$T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 7\,T(n/2) + n^2, & \text{si } n > 1 \end{cases}$$

Complexitat temporal DyV: teoremes mestres

Teorema mestre per a recurrència substractora: La solució a l'equació.

$$T(n) = \begin{cases} k, & \text{si } n \leq n_0; \\ aT(n-c) + \Theta(n^k), & \text{si } n > n_0 \end{cases}$$

té aquest cost:

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

No veurem la demostració.

Bibliografia

Bibliografia

- *Introduction to Algorithms*, de Cormen, Leiserson y Rivest
  - Capítols 1.3, 8 y 10
- *Fundamentos de Algoritmia*, de Brassard y Bratley
  - Capítol 7
- *Computer algorithms*, de Horowitz, Sahni y Rajasekaran
  - Capítol 3
- *Estructuras de Datos en Java*. Weiss, M.A. Adisson-Wesley.
  - Apartats del 1 al 4 del Capítol 7 y del 5 al 7 del Capítol 8

Bibliografia

- *Data Structures, Algorithms, and Applications in Java*. Sahni, S. McGraw-Hill, 2000.
  - Capítol 19
- *Data Structures and Algorithms in Java* (4th edition). Michael T. Goodrich and Roberto Tamassia. John Wiley & Sons, Inc., 2005.
  - Apartats 1 y 2 del Capítol 11, sobre la aplicació de DyV al problema de la Ordenació