Algoritmoen analisia

Algoritmoen konplexutasunaren neurketa:
Bilatze- eta ordenatze-algoritmoen analisia

Aurkibidea

- ***Zergatik da beharrezkoa?***
- ***Nola neurtu exekuzio-denbora?***
    - Zeren mende dago?
    - Nola kalkulatu?
        - Esperimentalki
        - Matematikoki
- ***Algoritmoen analisia***
    - Oinarrizko eragiketak
    - Notazio asintotikoa
- ***Analisi asintotikoaren murriztapenak***
- ***Bilatze- eta ordenazio-algoritmoak eta beren konplexutasunaren neurketa***

Konplexutasuna
Zergatik da beharrezkoa konplexutasuna aztertzea?

Algoritmo bat garatu eta zuzena den frogatu ondoren. bere **konplexutasun konputazionala** zehaztu behar da (exekuziorako behar dituen baliabideak)

Nola neurtu?

- Exekuzio denbora
- Hartzen duen espazioa
  - Memorian
  - Diskoan

Konplexutasuna
Nola neurtu exekuzio-denbora ?

Zer neurtzen da? Exekuzio-denbora

Zeren mende dago?

- Sarrerako datuen tamainaren arabera
- Beste hainbat faktore (Hw & Sw)
  - Ordenadorearen abiadura
  - Konpiladoreen kalitatea
  - Programaren kalitatea

Nola neurtu?

- Esperimentalki
- Matematikoki estimatuz

Konplexutasuna

Nola neurtu? Esperimentalki

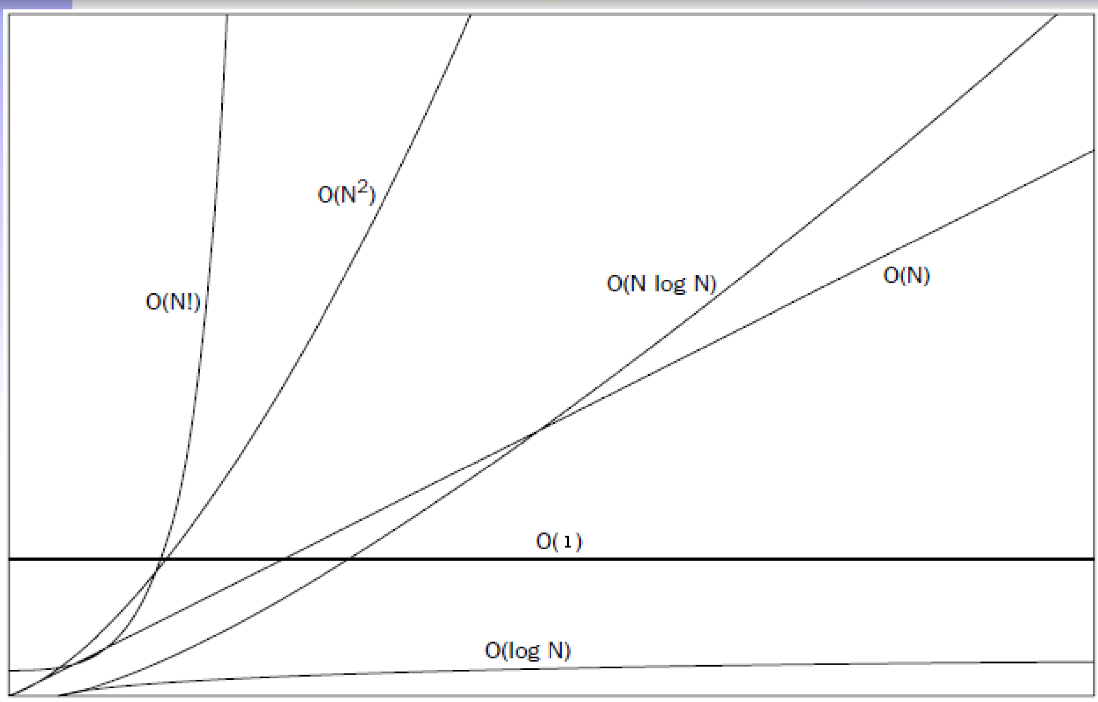
Exekuzio denbora neurtuz, sarrerako datuen tamainaren arabera

- Ezin ditugu sarrera posible guztiak frogatu
- Algoritmoa inplementatzea beharrezkoa da
- Softwarearen eta hardwarearen mende dago

Beste neurri bat behar dugu:

- Abstraktuagoa
- Lortzen errazagoa

O Notazioa



Algoritmoen konplexutasuna

Demagun array baten lehenengo eta bigarren elementuak berdinak diren kalkulatzeko algoritmo bat dugula.

a	b	c	d	e	f	g	h	i	j

?

* Zenbat konparaketa behar ditugu arrayak 10 elementu baditu?

* Zenbat konparaketa behar ditugu arrayak 100 elementu baditu?

Algoritmoen konplexutasuna

Algoritmoa, arrayaren neurriarekiko independentea da

Algoritmoaren konplexutasuna $O(1)$ da

Egin behar den konparaketa kopurua konstantea da
(ez dugu konparaketa gehiago egin behar, arrayan elementu gehiago izateagatik)

Algoritmoen konplexutasuna

Array baten lehenengo elementua bigarrenaren edo hirugarrenaren berdina den kalkulatzeko algoritmoa:

a	b	c	d	e	f	g	h	i	j

? ?

O(1)

Egin behar den konparaketa kopurua konstantea da
(ez dugu konparaketa gehiago egin behar, arrayan elementu gehiago izateagatik)

Algoritmoen konplexutasuna

Array baten lehenengo elementua arrayan errepikatuta dagoen ala ez aztertzen duen algoritmo bat.

a	b	c	d	e	f	g	h	i	j

* Kasu txarrenean zenbat konparaketa egin beharko ditugu?

$\boxed{n - 1}$ (n arrayaren elementu kopurua izanik) → n oso handia denean...

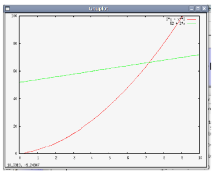
Konplexutasuna

Nola neurtu? Estimazio matematikoa

Algoritmo bakoitzarentzat, bere **sarrerako datuen mende dagoen funtzio bat** bilatzen da: f(n)

Exekuzio denbora gisa, **kasu okerrena** hartuko da

Sarrerako datuen neurriaren arabera, **algoritmoen hazkunde abiadura** kalkulatzea da helburua



Konplexutasuna

Adibidea

Hurrengo metodoa garatu nahi bada:

```
public boolean isAnagrama(Hitza h1, Hitza h2)
```

```
//aurre: h1 eta h2 letra xehez osatutako hitzak dira
//post: true, h1eta h2 letra berdinez osatutako hitzak
//        badira; false, bestela.
// Adibidea: h1="donostia" eta h2="itsaondo"
// anagramak dira
```

Adibidea : Anagrama? (I)

hitza1				hitza2				taula			
A	B	A	B	B	A	B	A	0	0	0	0
i				j							

hitza1				hitza2				taula			
A	B	A	B	B	A	B	A	0	1	0	0
i=0					j=1						

hitza1				hitza2				taula			
A	B	A	B	B	A	B	A	1	1	0	0
	i=1			j=0							

Adibidea: Anagrama? (I)

hitza1				hitza2				taula			
A	B	A	B	B	A	B	A	1	1	0	1
		i=2					j=3				

hitza1				hitza2				taula			
A	B	A	B	B	A	B	A	1	1	1	1
			i=3			j=2					

Egin beharreko eragiketak:
- 4 hasierako marka
- hitza1-en letra guztiak hitza2-n bilatu:
  4*4 konparaketa, kasurik okerrenean
- ikusi ea 4 letrak markatuak izan diren:
  kasurik okerrenean, 4 konparaketa

Anagrama(I)

Java programa

```
public static boolean isAnagrama(Hitza h1, Hitza h2) {
  char[] hitza1=h1.getWord();
  char[] hitza2=h2.getWord();
  int[] taula=new int[4]; int p,j;
  //Initialize visited positions
  for (int i=0;i<4;i++)
    taula[i]=0;
 for (int i=0;i<4;i++) {
   j=0;
   while ( (j<4) && !(hitza1[i]==hitza2[j] && taula[j]==0) )
    j++;
   if ( (j<4) && hitza1[i]==hitza2[j] && taula[j]==0 )
     taula[j]=1;
  }
  p=0;
  while ((p<4) && (taula[p]==1))
  p++;
  if (p==4) return true;
    else return false;
 }
```


Adibidea: Anagrama? (II)

hitza1

A	H	A	H

hitza2

H	A	H	A

0	0	0	0	0	0	0	0	....	0	0	0	0
A	B	C	D	E	F	G	H		W	X	Y	Z

2	0	0	0	0	0	0	2	....	0	0	0	0
A	B	C	D	E	F	G	H		W	X	Y	Z

0	0	0	0	0	0	0	0	....	0	0	0	0
A	B	C	D	E	F	G	H		W	X	Y	Z

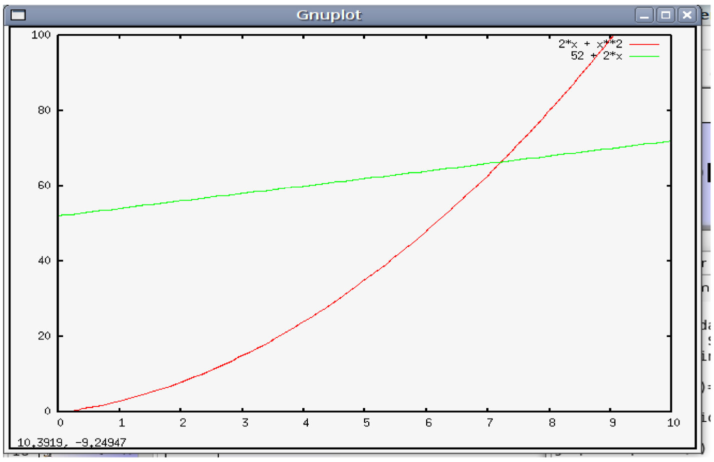
Egin beharreko eragiketak:

- 26 zero ipini (alfabeteko letra bakoitzeko bat)
- 4 batuketa egin (hitza1)
- 4 kenketa egin (hitza2)
- 26 konprobaketa egin (kasurik okerrenean)

Algoritmoen konparaketa

Elementuak	Algoritmo 1	Algoritmo 2
4	4+4*4+4=24	26+4+4+26=60
n	$n+n*n+n=2n+n^2$	26+n+n+26=52+2n

n handia denean, bigarren algoritmoa lehenengoa baina hobea da.



O Notazioa

Nola adierazten da algoritmo baten exekuzio-denbora?

Sarrerako datuen tamainaren arabera exekuzio-denbora nola igotzen den aztertuz.

$t(n) = 2n+n^2$

$t(n) = 52+2n$

$t(n) = 2n^2/5+6n+3\pi \log_2 n+2$

O Notazioa

-

O(1)	konstantea
O(log n)	logaritmikoa
O(n)	lineala
O(n log n)	quasilineala
$O(n^2)$	koadratikoa
$O(n^3)$	kubikoa
$O(n^m)$	polinomikoa
$O(2^n)$	esponentziala

+

Funtzioen hazkundea

log n	n	n log n	n^2	n^3	2^n
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4.096	65.536
5	32	160	1.024	32.768	4.294.967.296
6	64	384	4.096	262.144	$1{,}8 * 10^{19}$
7	128	896	16.536	2.097.152	$3{,}4 * 10^{38}$

Konplexutasuna
O Notazioa

$7n-3 \in O(n)$

$20n^3+10n\log n+5 \in O(n^3)$

$3 \log n+\log\log n \in O(\log n)$

$2^{100} \in O(1)$

$5/n \in O(1/n)$

Definizioak

$20n^3+10n\log n+5 \in O(n^3)$

Funtzio baten **termino nagusia**, hazkunde-abiadura handiena duen terminoa da

Funtzio baten **hazkunde-abiadura**, bere termino nagusien mende dago, beste elementu guztiak kontuan hartu gabe.

Funtzio baten **ordena** bere hazkunde-abiadurarekin erlazionatuta dago: termino nagusia soilik aztertu beharko da.

Adibidea I

Array baten elementu handiena lortu

Sarrera: n elementuz (osoak) osatutako A array bat

Irteera: A arrayaren elementu handiena

Algoritmo: arrayMax(A,n)

```
current=A[0];
for(i=1;i<n;i++)
    if (A[i]>current) current=A[i]
return current
```


Adibidea I

Ebazpena

```
current=A[0];
for(i=1;i<n;i++)
    if (A[i]>current) current=A[i]
return current
```

Kasurik onenean = 2+4*(n-1)+1 = 4n-1

Kasurik txarrenean = 2+6*(n-1)+1 = 6n-3

Adibidea II

- A array bat izanik, n elementuz (osoak) osatuta
- Kalkulatu B array bat, non:

$$B[i]=\sum_{j=0}^{i} A[j]$$

Adibidea II
Ebazpena I

```
for (i=0;i<n;i++) {
        b=0;
        for(j=0;j<=i;j++)
                b=b+A[j]
        B[i]=b;
}
return B
```

Pausoak:
  Hasieratu eta taula itzuli :O(n)
  i begizta: n aldiz exekutatzen da: O(n)
  j begizta: 1+2+3+...+n aldiz exekutatzen da=
    $((1+n)n)/2 = (n + n^2)/2$: $O(n^2)$

Totala: $O(n)+O(n)+O(n^2) \in O(n^2)$

Adibidea II
Ebazpena II

B[i]=A[0]+.......+A[i-1]+A[i]

B[i]=B[i-1]+A[i]

Aurreko terminoan egin ditugun eragiketak erabiltzen ditugu algoritmoaren konplexutasuna murrizteko

Adibidea II
Ebazpena II

```
b=0;
B[0]=A[0]
for(i=1;i<n;i++)
    B[i]=B[i-1]+A[i]
return B
```

Eragiketak:
- Hasieratu eta taula itzuli:O(n)
- Hasierako eragiketak:O(1)
- i begizta: n aldiz exekutatzen da

Guztira: $O(n)+O(1)+O(n) \in O(n)$ (Ordena lineala)

Beste adibide bat (I). Zenbat batuketa?

```
function Es_Separable (V, inicio, fin) return integer is
begin
  for i in inicio .. fin loop
    izq:= 0;
    for k in inicio .. i − 1 loop izq:= izq+V(k); end loop;
    der:= 0;
    for k in i .. fin loop der:= der+V(k); end loop;
    if izq = der then return i; end if;
  end loop;
  return 0;
end
```


Beste adibide bat (II). Zenbat batuketa?

```
function Suma_Escalonada (V, inicio, fin)
            return (integer×integer) is
begin
  maximo:= −∞;
  indice:= inicio−1;
  m:= (inicio + fin)/2;
  for i in inicio .. m loop
    suma:= 0;
    for k in i .. fin loop suma:= suma+V(k); end loop;
    if suma > maximo then
      maximo:= suma;
      indice:= i;
    end if;
  end loop;
  return (maximo, indice);
end
```

Beste adibide bat (III)

```
function Cua (n: natural) return natural is
begin
  i:= 0;
  while i*i <= n loop
     i:= i+1;
  endloop;
  return i-1;
end
```


Konplexutasuna

Analisi asintotikoaren murriztapenak

- Elementu gutxi daudenean ez da egokia

- Analisi asintotikoa **goi-estimazio bat da**

- Exekuzio-denbora: programaren batezbesteko denbora, kasurik okerreneko exekuzio-denbora baino askoz txikiagoa izan daiteke

- Batzuetan kasu okerrena ez da esanguratsua

Konplexutasuna

Analisi asintotikoaren murriztapenak

- **Elementu gutxi daudenean** ez da egokia

- Analisi asintotikoa **goi-estimazio bat da**

- **Exekuzio-denbora: programaren batezbesteko denbora, kasurik okerreneko exekuzio-denbora baino askoz txikiagoa izan daiteke**

- Batzuetan kasu okerrena ez da esanguratsua

Bilatze-algoritmoak (I)

Elementu bat array batean dagoen ala ez

Sarrera: A Array bat (n zenbaki osoz osatua) eta e osoko bat

Irteera: true bsb e $\in$ A; false, bestela

Algoritmoa: dago(A,n,e)

```
for(i=0;i<n;i++)
    if (A[i]==e) return true;
return false
```

Bilatze-algoritmoak (II)

Bilaketa dikotomikoa

Elementu bat array batean dagoen ala ez
Sarrera: A Array **ordenatu** bat (n zenbaki osoz osatua) eta e osoko bat
Irteera: true bsb e $\in$ A; false, bestela
Algoritmoa: dago(A,n,e)

```
int k=0;
int a=n-1;
while (k<=a) {
        int i=(k+a)/2;
        if (A[i]==e) return true;
        else {
            if (A[i] > e) a=i-1;
            else k=i+1;}
        }
return false;
```

Zenbat aldiz exekutatzen da while begizta?

n, n/2, n/4, …, $n/2^i$

Kasurik okerrenean: $2^i = n$

Orduan: i=log n

O(log N)

Ordenazio-algoritmoak

Ordenaziorako Java klaseak

public class SortAndSearch <T **extends** Comparable<T>>{

public SortAndSearch() { }

public void bubbleSort(T[] taula)
public void selectionSort(T[] taula)
public void insertionSort(T[] taula)
public void mergeSort(T[] taula)
public void quickSort(T[] taula)

Ordenazio-algoritmoak
Burbuilaren metodoa

```
public void bubbleSort(T[] taula) {
    int out, in;

    for (out = taula.length - 1; out > 0; out--)
      for (in = 0; in < out; in++)
        if ( taula[in].compareTo(taula[in + 1]) > 0 )
          swap(taula, in, in + 1);
  }

 private void swap(T[] taula, int one, int two) {
    T temp = taula[one];
    taula[one] = taula[two];
    taula[two] = temp;
 }
```


Ordenazio-algoritmoak
Hautaketa bidezko metodoa

```
public void selectionSort(T[] taula) {
    int out, in, min;

    for (out = 0; out < taula.length - 1; out++)
    {
      min = out; // minimoaren indizea, oraingoz
      for (in = out + 1; in < taula.length; in++)
        if (taula[in].compareTo(taula[min]) < 0)
          min = in; // minimoaren indizea, eguneratua
      swap(taula, out, min);
    }
  }
```

Ordenazio-algoritmoak

Txertaketa bidezko metodoa

```
public void insertionSort(T[] taula) {
    int in, out;

    for (out = 1; out < taula.length; out++)
        // out indizearen aurrekoak ordenatuta daude
    {
        T temp = taula[out];
        in = out;
        while (in > 0 && taula[in - 1].compareTo(temp) > 0)
        {
            taula[in] = taula[in - 1]; // (in-1) indizeko elementua eskuinera ekarri
            in--; // indizea ezkerrera ekarri
        }
        taula[in] = temp;
        // temp balioa txertatzen dugu tokatzen zaion posizioan
    }
}
```


Ordenazio-algoritmoak

Fusioz ordenaketa edo bateratze-metodoa

```
public void mergeSort(T[] taula){
    mergeSort(taula, 0, taula.length-1);
}

private void mergeSort (T[] taulaBat, int hasiera, int bukaera){
    if (hasiera < bukaera) { // taulan elementu bat baino gehiago badago
        mergeSort(taulaBat, hasiera, (hasiera+bukaera)/2);
        mergeSort(taulaBat, ((hasiera+bukaera)/2)+1, bukaera);
        bateratze(taulaBat, hasiera, (hasiera+bukaera)/2, bukaera);
    }
}
```

Ordenazio-algoritmoak
Bateratzea, bi taula ordenatuena

```
private void bateratze (T[ ] taula, int i, int erdikoa, int f){
   T[ ] bateratua = (T[ ])(new Comparable[f-i+1]);
   int ezker = i;
   int eskuin = erdikoa+1;
   int k = 0; //bateratua taula betetzeko indizea
   while ( ezker<=erdikoa && eskuin<=f ){
     if ( taula[ezker].compareTo(taula[eskuin])<= 0 ){
       bateratua[k] = taula[ezker];
       k++;
       ezker++;
     }
     else {
       bateratua[k] = taula[eskuin];
       k++;
       eskuin++;
     }
   }
   if ( ezker > erdikoa )
       while ( eskuin<=f ){
          bateratua[k] = taula[eskuin];
          k++;
          eskuin++;
       }
   else {
       while ( ezker<=erdikoa ){
          bateratua[k] = taula[ezker];
          k++;
          izq++;
       }
   }
   for (int j=i; j<=f; j++)
      taula[j] = bateratua[j-i];
}
```


Ordenazio-algoritmoak
Quicksort

```
public void quickSort(T[] taula){
   quickSort(taula, 0, taula.length-1);
}


private void quickSort(T[ ] taulaBat, int hasiera, int bukaera){
   if ( bukaera - hasiera> 0 ) {  // taulan elementu bat baino gehiago
      int indizeaZatiketa = zatiketa(taulaBat, hasiera, bukaera);
      quickSort(taulaBat, hasiera, indizeaZatiketa - 1);
      quickSort(taulaBat, indizeaZatiketa + 1, bukaera);
   }
}
```

Ordenazio-algoritmoak

Quicksort

```
private int zatiketa(T[] taula, int i, int f){

    T lag = taula[i];
    int ezker = i;
    int eskuin = f;

    while ( ezker < eskuin ){
      while ( taula[ezker].compareTo(lag) <= 0 && ezker < eskuin)
        ezker++;
      while ( taula[eskuin].compareTo(lag) > 0 )
        eskuin--;
      if ( ezker < eskuin )
        swap(taula, ezker, eskuin);
    }
    taula[i] = taula[eskuin];
    taula[eskuin] = lag;

    return eskuin;
  }
```


Bestelakoak

- Irakurketa gehigarria:
  - [Lewis, Chase 2010], 8. kapitulua
- Wikipedia (ingelesez). Algoritmoen analisia:
  http://en.wikipedia.org/wiki/Analysis_of_algorithms
- Wikipedia (gaztelaniaz). Ordenazio-algoritmoak:
  http://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento
- Simulatzaileak:
  - http://math.hws.edu/eck/jsdemo/sortlab.html
  - http://www.sorting-algorithms.com/
- Quicksorten dantza:
  - http://www.youtube.com/watch?v=ywWBy6J5gz8