COMPLESSITÀ DEGLI ALGORITMI

Complessità computazionale

Complessità asintotica

COME SI GIUDICA UN ALGORITMO?

COME SI GIUDICA UN ALGORITMO?

- ❑ Quanto è facile capire ciò che fa
- ❑ Quanto è facile modificarlo
- ❑ Si può fare lo stesso con meno istruzioni
- ❑ È abbastanza user-friendly

PARAMETRI SOGGETTIVI

- ❑ Il tempo impiegato dall'esecuzione
- ❑ La quantità di RAM e di disco necessaria
- ❑ Il tempo necessario ad acquisire/produrre I/O
- ❑ Il tempo necessario a comunicare con computer remoti

PARAMETRI OGGETTIVI

IL TEMPO D'ESECUZIONE

Supponiamo di voler confrontare due algoritmi per l'ordinamento di un vettore.

IL TEMPO D'ESECUZIONE

```
// riempiamo un vettore
var vett = new Array();
for (i=0; i<50; i++)
    vett[i] = Math.floor(Math.random()*100)+1;
```

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

19

MILLESIMI DI SECONDO

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++){
    // cerca il minimo da qui in avanti
    minimo = vett[i];
    indice = i;
    for (j=i+1; j<vett.length; j++) {
        // aggiorna il minimo
        if (vett[j] < minimo) {
            minimo = vett[j];
            indice = j;
        }}
    // metti il minimo in prima pos
    vett[indice] = vett[i];
    vett[i] = minimo;
}
```

20

MILLESIMI DI SECONDO

IL TEMPO D'ESECUZIONE

```
// riempiamo un vettore
var vett = new Array();
for (i=0; i<50; i++)
    vett[i] = Math.floor(Math.random()*100)+1;
```

100

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

55

MILLESIMI DI SECONDO

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++){
    // cerca il minimo da qui in avanti
    minimo = vett[i];
    indice = i;
    for (j=i+1; j<vett.length; j++) {
        // aggiorna il minimo
        if (vett[j] < minimo) {
            minimo = vett[j];
            indice = j;
        }}
    // metti il minimo in prima pos
    vett[indice] = vett[i];
    vett[i] = minimo;
}
```

68

MILLESIMI DI SECONDO

IL TEMPO D'ESECUZIONE

```
// riempiamo un vettore
var vett = new Array();
for (i=0; i<50; i++)
    vett[i] = Math.floor(Math.random()*100)+1;
```

200

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

200

MILLESIMI DI SECONDO

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++){
    // cerca il minimo da qui in avanti
    minimo = vett[i];
    indice = i;
    for (j=i+1; j<vett.length; j++) {
        // aggiorna il minimo
        if (vett[j] < minimo) {
            minimo = vett[j];
            indice = j;
        }}
    // metti il minimo in prima pos
    vett[indice] = vett[i];
    vett[i] = minimo;
}
```

260

MILLESIMI DI SECONDO

IL TEMPO D'ESECUZIONE

```
// riempiamo un vettore
var vett = new Array();
for (i=0; i<50; i++)
    vett[i] = Math.floor(Math.random()*100)+1;
```

500

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

1200

MILLESIMI DI SECONDO

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++){
    // cerca il minimo da qui in avanti
    minimo = vett[i];
    indice = i;
    for (j=i+1; j<vett.length; j++) {
        // aggiorna il minimo
        if (vett[j] < minimo) {
            minimo = vett[j];
            indice = j;
        }}
    // metti il minimo in prima pos
    vett[indice] = vett[i];
    vett[i] = minimo;
}
```

1600

MILLESIMI DI SECONDO

IL TEMPO D'ESECUZIONE

```
// riempiamo un vettore
var vett = new Array();
for (i=0; i<50; i++)
    vett[i] = Math.floor(Math.random()*100)+1;
```

1000

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

4800

MILLESIMI DI SECONDO

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++){
    // cerca il minimo da qui in avanti
    minimo = vett[i];
    indice = i;
    for (j=i+1; j<vett.length; j++) {
        // aggiorna il minimo
        if (vett[j] < minimo) {
            minimo = vett[j];
            indice = j;
        }}
    // metti il minimo in prima pos
    vett[indice] = vett[i];
    vett[i] = minimo;
}
```

6400

MILLESIMI DI SECONDO

IL TEMPO D'ESECUZIONE

```
// riempiamo un vettore
var vett = new Array();
for (i=0; i<50; i++)
    vett[i] = Math.floor(Math.random()*100)+1;
```

5000

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

120000

MILLESIMI DI SECONDO

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++){
    // cerca il minimo da qui in avanti
    minimo = vett[i];
    indice = i;
    for (j=i+1; j<vett.length; j++) {
        // aggiorna il minimo
        if (vett[j] < minimo) {
            minimo = vett[j];
            indice = j;
        }}
    // metti il minimo in prima pos
    vett[indice] = vett[i];
    vett[i] = minimo;
}
```

160000

MILLESIMI DI SECONDO

IL TEMPO D'ESECUZIONE

n	Primo Algoritmo	Secondo Algoritmo
50	19 msec	20 msec
100	55 msec	68 msec
200	200 msec	260 msec
500	1,2 sec	1,6 sec
1000	4,8 sec	6,4 sec
5000	2 min	2 min e mezzo

Sembra che
– anche se per poco –
sia meglio il primo algoritmo.

IL TEMPO D'ESECUZIONE

Cronometrando il tempo di esecuzione del programma siamo riusciti a determinare qual era il più veloce tra i due, ma questa informazione è dipendente da:

- ❑ il compilatore usato
- ❑ la macchina usata
- ❑ l'istante in cui avviene il test

Non sembra un metodo molto "scientifico" per giudicare un algoritmo.

COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
i = 8;
vett[i] = "Ale";
alert(i);
k=i+1;
```

1

COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
i = 8;
vett[i] = "Ale";
alert(i);
k=i+1;
```

1

2

COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
i = 8;
vett[i] = "Ale";
alert(i);
k=i+1;
```

1
2
2

COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
i = 8;
vett[i] = "Ale";
alert(i);
k=i+1;
```

1
2
2
3

Questa porzione di codice impiega quindi 8 unità di tempo.

COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
if (vett[i] > vett[j]) {
    temp = vett[i];
    vett[i] = vett[j];
    vett[j] = temp;
}
```

5

COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
if (vett[i] > vett[j]) {
    temp = vett[i];
    vett[i] = vett[j];
    vett[j] = temp;
}
```

5

3

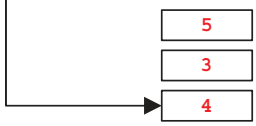
COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
if (vett[i] > vett[j]) {
    temp = vett[i];
    vett[i] = vett[j];
    vett[j] = temp;
}
```



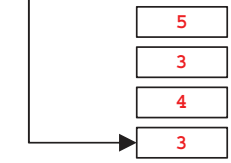
COMPLESSITÀ COMPUTAZIONALE

Invece di cronometrare il tempo di esecuzione del programma, consideriamo le istruzioni dell'algoritmo e vediamo quante volte vengono eseguite durante un'esecuzione tipo.

Consideriamo di impiegare una unità di tempo per eseguire:

- un'assegnazione
- un'operazione di I/O
- la lettura del valore di una variabile
- l'esecuzione di un'operazione aritmetica o logica

```
if (vett[i] > vett[j]) {
    temp = vett[i];
    vett[i] = vett[j];
    vett[j] = temp;
}
```



Questa porzione di codice impiega quindi 15 unità di tempo.

COMPLESSITÀ COMPUTAZIONALE

Invece che in termini assoluti
esprimiamo la complessità in relazione alla cardinalità dell'input.

Per esempio, un algoritmo che lavora con i vettori
avrà una complessità computazionale espressa in termini di n
dove n rappresenta la dimensione del vettore.

```
var vett = new Array();
const n = 50;
for ( i=0; i<n; i++ )
    vett[i] = Math.floor(Math.random()*100)+1;
```

1

COMPLESSITÀ COMPUTAZIONALE

Invece che in termini assoluti
esprimiamo la complessità in relazione alla cardinalità dell'input.

Per esempio, un algoritmo che lavora con i vettori
avrà una complessità computazionale espressa in termini di n
dove n rappresenta la dimensione del vettore.

```
var vett = new Array();
const n = 50;
for ( i=0; i<n; i++ )
    vett[i] = Math.floor(Math.random()*100)+1;
```

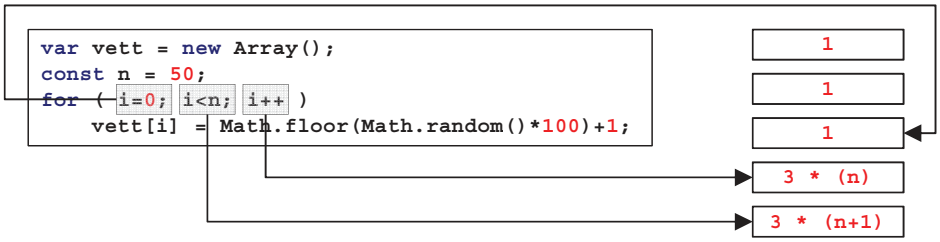
1

1

COMPLESSITÀ COMPUTAZIONALE

Invece che in termini assoluti
esprimiamo la complessità in relazione alla cardinalità dell'input.

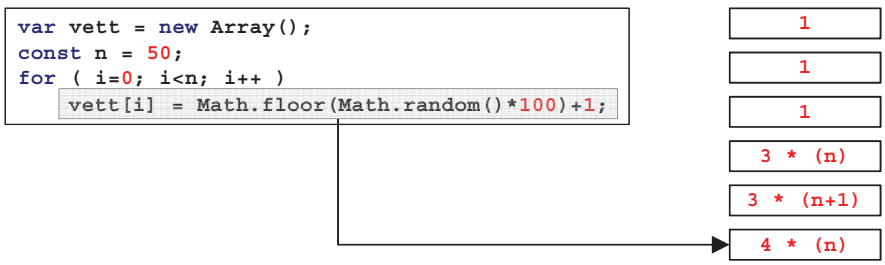
Per esempio, un algoritmo che lavora con i vettori
avrà una complessità computazionale espressa in termini di n
dove n rappresenta la dimensione del vettore.



COMPLESSITÀ COMPUTAZIONALE

Invece che in termini assoluti
esprimiamo la complessità in relazione alla cardinalità dell'input.

Per esempio, un algoritmo che lavora con i vettori
avrà una complessità computazionale espressa in termini di n
dove n rappresenta la dimensione del vettore.



Questa porzione di codice impiega quindi (10n + 6) unità di tempo.

COMPLESSITÀ COMPUTAZIONALE

Diciamo quindi
che questa porzione di codice
ha complessità computazionale
T(n) = (10n + 6)

Contiamo quante unità di tempo
impieghiamo al cambiare di n
e - supponendo che una unità di tempo venga
eseguita in un microsecondo –
calcoliamo il tempo d'esecuzione.

n	Unità di tempo	Secondi
50	506	< 0,001
100	1006	0,001
200	2006	0,002
500	5006	0,005
1000	10006	0,01
5000	50006	0,05
10000	100006	0,1

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)
20n + 71
20n + 131
20n + 1289

Osserviamo che il coefficiente di n è sempre lo stesso per ciascun algoritmo

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100
20n + 71	0,002
20n + 131	0,002
20n + 1289	0,003

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000
20n + 71	0,002	0,02
20n + 131	0,002	0,02
20n + 1289	0,003	0,02

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000	n = 10^6
20n + 71	0,002	0,02	20
20n + 131	0,002	0,02	20
20n + 1289	0,003	0,02	20

Osserviamo
che le unità di tempo
costanti
sono trascurabili.

CC BY

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000	n = 10^6
20n + 71	0,002	0,02	20
20n + 131	0,002	0,02	20
20n + 1289	0,003	0,02	20

Osserviamo
che le unità di tempo
costanti
sono trascurabili.

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000	n = 10^6
20n + 71	0,002	0,02	20
20n + 131	0,002	0,02	20
20n + 1289	0,003	0,02	20

Osserviamo
che le unità di tempo
costanti
sono trascurabili.

T(n)
5n
19n
34n

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000	n = 10^6
20n + 71	0,002	0,02	20
20n + 131	0,002	0,02	20
20n + 1289	0,003	0,02	20

Osserviamo
che le unità di tempo
costanti
sono trascurabili.

T(n)	n = 100
5n	0,0005
19n	0,0019
34n	0,0034

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000	n = 10^6
20n + 71	0,002	0,02	20
20n + 131	0,002	0,02	20
20n + 1289	0,003	0,02	20

Osserviamo
che le unità di tempo
costanti
sono trascurabili.

T(n)	n = 100	n = 1000
5n	0,0005	0,005
19n	0,0019	0,019
34n	0,0034	0,034

COMPLESSITÀ COMPUTAZIONALE

Supponiamo di avere per uno stesso problema
tre algoritmi diversi con complessità computazionale diversa.
Vediamo che tempi di esecuzione hanno con dimensioni di dati di input diverse.

T(n)	n = 100	n = 1000	n = 10^6
20n + 71	0,002	0,02	20
20n + 131	0,002	0,02	20
20n + 1289	0,003	0,02	20

Osserviamo
che le unità di tempo
costanti
sono trascurabili.

T(n)	n = 100	n = 1000	n = 10^6
5n	0,0005	0,005	5
19n	0,0019	0,019	19
34n	0,0034	0,034	34

Ci domandiamo
quindi se anche i
coefficienti di n sono
trascurabili oppure no..

COMPLESSITÀ COMPUTAZIONALE

..per scoprirlo
confrontiamo quattro algoritmi la cui complessità computazionale
è rappresentata da funzioni di n diverse.

T(n)
3logn + 5
28n + 12
$3n^2 + 82$
$2^n + 44$

COMPLESSITÀ COMPUTAZIONALE

Completiamo le nostre considerazioni sui tempi di esecuzione
confrontando quattro algoritmi la cui complessità computazionale
è rappresentata da funzioni di n diverse.

T(n)	n = 100
3logn + 5	< 0,0001
28n + 12	0,0001
$3n^2 + 82$	0,01
$2^n + 44$	molti secoli

COMPLESSITÀ COMPUTAZIONALE

Completiamo le nostre considerazioni sui tempi di esecuzione confrontando quattro algoritmi la cui complessità computazionale è rappresentata da funzioni di n diverse.

T(n)	n = 100	n = 1000
$3\log n + 5$	< 0,0001	< 0,0001
$28n + 12$	0,0001	0,001
$3n^2 + 82$	0,01	1
$2^n + 44$	molti secoli	moltissimi

COMPLESSITÀ COMPUTAZIONALE

Completiamo le nostre considerazioni sui tempi di esecuzione confrontando quattro algoritmi la cui complessità computazionale è rappresentata da funzioni di n diverse.

T(n)	n = 100	n = 1000	$n = 10^6$
$3\log n + 5$	< 0,0001	< 0,0001	< 0,0001
$28n + 12$	0,0001	0,001	1
$3n^2 + 82$	0,01	1	11 giorni
$2^n + 44$	molti secoli	moltissimi	troppi

Osserviamo quindi che via via che la dimensione dell'input aumenta si delineano sempre più nettamente quattro classi

Per n che tende all'infinito quindi ciò che differenzia un algoritmo da un altro è l'ordine di grandezza (complessità asintotica)

COMPLESSITÀ ASINTOTICA

A (n)

ALGORITMO (dimensione n)

T (n)

COMPLESSITÀ COMPUTAZIONALE

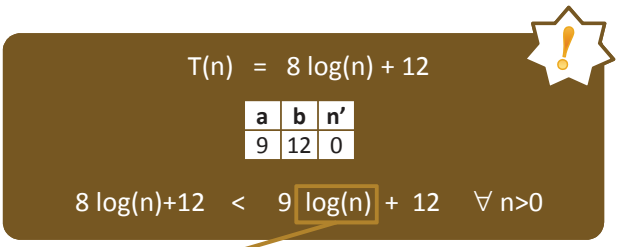
O (f(n))

se esistono tre costanti a, b ed n' tali che

T(n) < a * f(n) + b

per ogni n > n'

COMPLESSITÀ ASINTOTICA



O(log(n))

CC BY

COMPLESSITÀ ASINTOTICA

A (n)

ALGORITMO (dimensione n)

T (n)

COMPLESSITÀ COMPUTAZIONALE

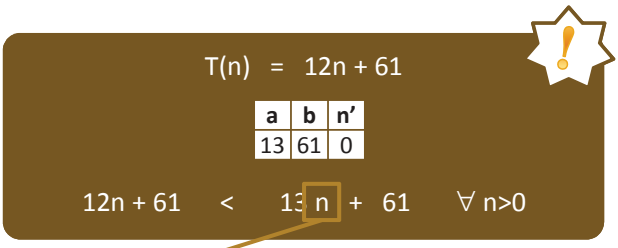
O (f(n))

se esistono tre costanti a, b ed n' tali che

T(n) < a * f(n) + b

per ogni n > n'

COMPLESSITÀ ASINTOTICA



O(n)

COMPLESSITÀ ASINTOTICA

A (n)

ALGORITMO (dimensione n)

T (n)

COMPLESSITÀ COMPUTAZIONALE

O (f(n))

se esistono tre costanti a, b ed n' tali che

$T(n) < a * f(n) + b$

per ogni $n > n'$

COMPLESSITÀ ASINTOTICA

$T(n) = 3n^2 + 2n + 4$

a	b	n'
4	0	3

$3n^2 + 2n + 4 < 4n^2 \quad \forall\ n>3$

$O(n^2)$

VERIFICA SE UN NUMERO È PRIMO

```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n-1; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```

VERIFICA SE UN NUMERO È PRIMO

```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n-1; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```

1

VERIFICA SE UN NUMERO È PRIMO

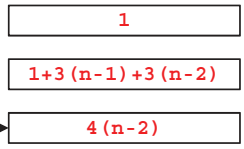
```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```

1

1+3(n-1)+3(n-2)

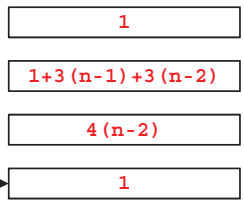
VERIFICA SE UN NUMERO È PRIMO

```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```



VERIFICA SE UN NUMERO È PRIMO

```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```



VERIFICA SE UN NUMERO È PRIMO

```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```

1
1+3(n-1)+3(n-2)
4(n-2)
1
1

VERIFICA SE UN NUMERO È PRIMO

```
function isPrimo(n) {
    // vedo se esiste un divisore diverso da 1 e n
    for (i=2; i<n; i++)
        // se trovo un divisore
        if (n%i == 0)
            return false;
    // se non ne ho trovati
    return true;
}
```

1
1+3(n-1)+3(n-2)
4(n-2)
1
1

$T(n) = 10n - 13$

$T(n) < 11n \quad \forall\, n>0$

$O(n)$

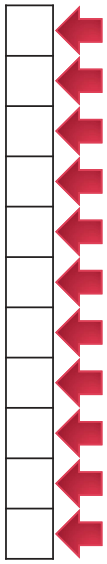
RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```



RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1

1 + 2(n+1) + 3n

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1

1 + 2(n+1) + 3n

4n

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

```
1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
```


RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

```
1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
```

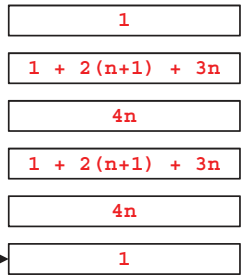
RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```



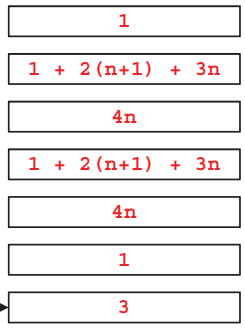
RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```



RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

```
1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
1
3
1
```


RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

```
1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
1
3
1
1 + 3(n+1) + 3n
```

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
1
3
1
1 + 3(n+1) + 3n
4n

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
1
3
1
1 + 3(n+1) + 3n
4n
n

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
1
3
1
1 + 3(n+1) + 3n
4n
n
3

RICERCA SEQUENZIALE

```
// riempo un vettore
var vett = new Array();
for (i=0; i<10; i++)
    vett[i] = Math.floor(Math.random()*100)+1;

// lo stampo
for (i=0; i<10; i++)
    document.write(vett[i]+";");
document.write("<br>");

// acquisisco il valore da cercare
valDaCercare = prompt("Che valore vuoi cercare?");

// CERCO IL VALORE (RICERCA SEQUENZIALE)
trovato = false;
for (i=0; i<vett.length; i++) {
    if (vett[i] == valDaCercare)
        trovato = true;
}
// output
alert( (trovato==true) ? "Trovato" : "Non trovato" );
```

1
1 + 2(n+1) + 3n
4n
1 + 2(n+1) + 3n
4n
1
3
1
1 + 3(n+1) + 3n
4n
n
3

$T(n) = 30n + 20$

$T(n) < 31n \quad \forall n>0$

$O(n)$

ORDINAMENTO DI UN VETTORE

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

$1 + 4n + 3(n-1)$

ORDINAMENTO DI UN VETTORE

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

$1 + 4n + 3(n-1)$

$(n-1)(3+3n+3(n-1))$

ORDINAMENTO DI UN VETTORE

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

1 + 4n + 3(n-1)

(n-1)(3+3n+3(n-1))

(n-1)(5(n-1))

ORDINAMENTO DI UN VETTORE

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

1 + 4n + 3(n-1)

(n-1)(3+3n+3(n-1))

(n-1)(5(n-1))

(n-1)(3(n-1))

ORDINAMENTO DI UN VETTORE

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

```
1 + 4n + 3(n-1)
(n-1)(3+3n+3(n-1))
(n-1)(5(n-1))
(n-1)(3(n-1))
(n-1)(4(n-1))
```


ORDINAMENTO DI UN VETTORE

```
// dal primo al penultimo
for (i=0; i<vett.length-1; i++)
    //dal successivo all'ultimo
    for (j=i+1; j<vett.length; j++)
        // se sono disordinati
        if (vett[i] > vett[j]) {
            // scambiali
            temp = vett[i];
            vett[i] = vett[j];
            vett[j] = temp;
        }
```

```
1 + 4n + 3(n-1)
(n-1)(3+3n+3(n-1))
(n-1)(5(n-1))
(n-1)(3(n-1))
(n-1)(4(n-1))
(n-1)(3(n-1))
```

$T(n) = 21n^2 - 36n + 13$

$T(n) < 22n^2 \quad \forall\, n>0$

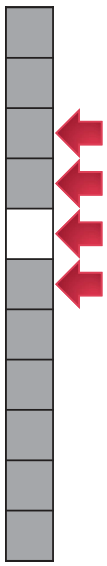
$O(n^2)$

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO


// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```



RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO


// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

1

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO

// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

1

3

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO

// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

1

3

5

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO

// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

1

3

5

(4 + 4 + 1)logn

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO

// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

1

3

5

(4 + 4 + 1)logn

4logn

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO


// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

- 1
- 3
- 5
- (4 + 4 + 1)logn
- 4logn
- 2logn

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO


// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

- 1
- 3
- 5
- (4 + 4 + 1)logn
- 4logn
- 2logn
- 2logn

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO


// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

1
3
5
(4 + 4 + 1)logn
4logn
2logn
2logn
5logn

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO


// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

$O(\log n)$

1
3
5
(4 + 4 + 1)logn
4logn
2logn
2logn
5logn

$T(n) = 22\log n + 9$

$T(n) < 23\log n + 9 \quad \forall n>0$

RICERCA DICOTOMICA

```
// supponiamo di avere un vettore ORDINATO

// definisco gli indici
inizio = 0;
fine = vett.length-1;
centro = Math.floor((fine-inizio)/2);

// esco dal ciclo quando si verifica
// una delle seguenti condizioni:
//      il valore cercato è al centro dell'intervallo
//      l'intervallo non ha il centro
while ( (vett[centro]!=valDaCercare) && (inizio!=fine-1) ) {
    // aggiorno l'intervallo
    if (vett[centro] < valDaCercare)
        inizio = centro;
    else
        fine = centro;
    // aggiorno il centro
    centro = Math.floor((fine+inizio)/2);
}
```

$O(n^2)$

$O(\log n)$

$O(n^2) + O(\log n) = O(n^2)$

CASO OTTIMO
CASO MEDIO
CASO PESSIMO

QUALCHE OSSERVAZIONE

Abbiamo appena visto tre algoritmi

Ricerca sequenziale	O(n)
Ordinamento	$O(n^2)$
Ricerca dicotomica	O(log n)

La ricerca dicotomica ha una complessità asintotica migliore ma prevede che il vettore sia ordinato

L'algoritmo di ordinamento che abbiamo visto non è l'unico: potrebbero esistere degli altri algoritmi più efficienti

L'algoritmo di inserimento in un vettore ordinato ha una complessità maggiore dell'algoritmo di inserimento in un vettore non ordinato

Oltre a considerare il caso peggiore
è possibile valutare il caso a noi più favorevole così come pure il caso medio.

NON SOLO IL CASO PEGGIORE

	Caso peggiore	Caso migliore	Caso medio
Ricerca sequenziale	n	1	n/2
Ricerca dicotomica	log n	1	log n

La prassi è porre attenzione al caso peggiore
(e al caso medio)

CLASSI DI COMPLESSITÀ

degli algoritmi

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

gli algoritmi che eseguono
lo stesso numero di operazioni
indipendetemente dalla
dimensione dei dati in input

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

gli algoritmi che eseguono
un numero di operazioni
proporzionali a
log n

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

gli algoritmi che eseguono
un numero di operazioni
proporzionali a
n

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

gli algoritmi che eseguono
un numero di operazioni
proporzionali a
n log n

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

gli algoritmi che eseguono
un numero di operazioni
proporzionali a
n^2

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
O(n^2)	quadratico
O(n^k)	polinomiale
O(k^n)	esponenziale

gli algoritmi che eseguono un numero di operazioni proporzionali a n^k

CLASSI DI COMPLESSITÀ DEGLI ALGORITMI

Possiamo facilmente classificare gli algoritmi
secondo l'ordine di grandezza della loro complessità computazionale

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
O(n^2)	quadratico
O(n^k)	polinomiale
O(k^n)	esponenziale

gli algoritmi che eseguono un numero di operazioni proporzionali a k^n

CLASSI DI COMPLESSITÀ

dei problemi

CLASSI DI COMPLESSITÀ DEI PROBLEMI

Teniamo presente le classi di complessità degli algoritmi.

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

Semplicisticamente possiamo dire che:
dati due problemi A e B
se esiste un algoritmo per risolvere A in O(n)
e se il miglior algoritmo che risolve B impiega $O(2^n)$
allora A è un problema più facile di B

In realtà – per nostra fortuna –
la classificazione che si fa dei problemi
è molto più semplice.

CLASSI DI COMPLESSITÀ DEI PROBLEMI

Teniamo presente le classi di complessità degli algoritmi.

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

La classificazione dei problemi deriva da una questione che abbiamo già affrontato.

Gli algoritmi di classe esponenziale richiedono un tempo d'esecuzione che non è ragionevole

CLASSI DI COMPLESSITÀ DEI PROBLEMI

CLASSI DI COMPLESSITÀ DEI PROBLEMI

T(n)	n = 100	n = 1000	n = 10^6
3logn + 5	< 0,0001	< 0,0001	< 0,0001
28n + 12	0,0001	0,001	1
$3n^2 + 82$	0,01	1	11 giorni
$2^n + 44$	molti secoli	moltissimi	troppi

CLASSI DI COMPLESSITÀ DEI PROBLEMI

Teniamo presente le classi di complessità degli algoritmi.

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

Per cui dividiamo i problemi in **trattabili** e **non trattabili**

i problemi trattabili sono risolti da algoritmi che hanno una complessità al più polinomiale → problemi di classe P

i problemi non trattabili sono risolti da algoritmi che hanno una complessità esponenziale (non polinomiale) → problemi di classe NP

CLASSI DI COMPLESSITÀ DEI PROBLEMI

Teniamo presente le classi di complessità degli algoritmi.

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

Per cui dividiamo i problemi in **trattabili** e **non trattabili**

Ovvero in problemi di **classe P** e problemi di **classe NP**

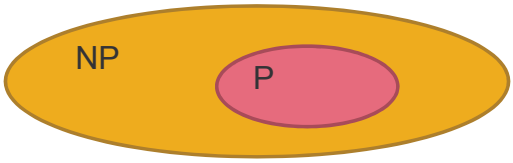
CLASSI DI COMPLESSITÀ DEI PROBLEMI

Teniamo presente le classi di complessità degli algoritmi.

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

Ovviamente preso un problema di **classe P** è facile immaginare che ci sia un'altra soluzione (meno performante) che impieghi tempo esponenziale

Pertanto P è un sottinsieme di NP



Non è possibile invece dire se i due insiemi coincidono

CLASSI DI COMPLESSITÀ DEI PROBLEMI

Teniamo presente le classi di complessità degli algoritmi.

O(1)	costante
O(log n)	logaritmico
O(n)	lineare
O(n log n)	n log n
$O(n^2)$	quadratico
$O(n^k)$	polinomiale
$O(k^n)$	esponenziale

L'esistenza dei problemi NP è un bene o un male?

Un sistema di crittografia è sicuro proprio perché la sua rottura non è possibile in tempo polinomiale!

ALESSANDRO URSOMANDO

ESERCIZI

Riempi un vettore con dieci numeri interi tra 1 e 100.
Successivamente stampa il vettore.
In seguito sostituisci gli elementi iniziali inferiori a 50 con il valore -1.
Stampa nuovamente il vettore

```
for (int i=0; i<10; i++) {
    if (v[i] < 50) {
        v[i] = -1;
    } else {
        break;
    }
}
```

```
int i=0;
while ( (i<=9) && (v[i] < 50) ){
    v[i]=-1;
    i++;
}
```


Riempi un vettore con dieci numeri interi tra 1 e 10 tutti diversi tra loro.
Successivamente stampa il vettore.

```
for (int i=0; i<v.length; i++) {
    boolean numeroNuovo;
    int num;
    do {
        num = (int)(Math.random()*10) + 1;
        numeroNuovo = true;
        for (int j=0; j<i; j++) {
            if (v[j]==num) {
                numeroNuovo = false;
            }
        }
    } while ( numeroNuovo == false );
    v[i] = num;
}
```

Riempi un vettore con dieci numeri interi tra 1 e 10 tutti diversi tra loro.
Successivamente stampa il vettore.

```
for (int i=0; i<v.length; i++) {
    v[i] = (int)(Math.random()*10) + 1;
    int j=0;
    while ( (j<i) && (v[j] != v[i]) ) {
         j++;
    }
    if (j!=i) {
         i--;
    }
}
```