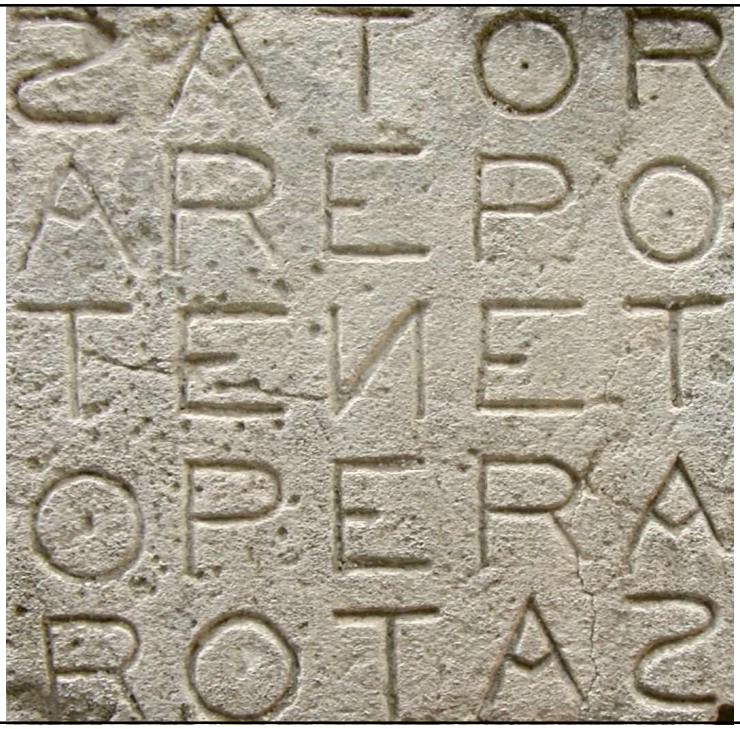
Ancora ricorsione e ordinamenti

Lezione 3

1



Quadrato magico

2

Array palindroma

```
int palindroma(int *a, int i=0, int j=n-1){
      if (j<i) return 1;
      if (a[i]==a[j]) return palindroma(a,i+1,j-1);
      return 0;
}
```

3

- T(1)=k1
- T(n)=K2+T(n-2)

- Palindroma su un'array di n elementi è **O(n)**

4

Ordinamento di un insieme

quicksort(array A ,inf, sup)

Lavora sulla porzione di array compresa fra inf e sup:

1. **Scegli un perno**
2. **Dividi l'array in due parti: nella prima metti gli elementi <= perno e nella seconda quelli >= perno**
3. **chiama quicksort sulla prima parte (se contiene almeno 2 elementi)**
4. **chiama quicksort sulla seconda parte (se contiene almeno 2 elementi)**

metodo

la divisione in due parti dell'array avviene mediante scambi fra elementi utilizzando due cursori s e d, inizialmente posti uno su inf e l'altro su sup:

1. Fino a quando s<=d:

2. {Incrementa s fino a che A[s] < perno;
Decresci d fino a che A[d] > perno;
Scambia A[s] con A[d];
S++;
D--;
}

3. Se inf<d chiama quicksort sulla prima parte (da inf a d);
4. Se s<sup chiama quicksort sulla seconda parte (da s a sup);

QuickSort

```
void quickSort(int A[], int inf=0, int sup=n-1) {

        int perno = A[(inf + sup) / 2], s = inf, d = sup;

        while (s <= d) {
            while (A[s] < perno) s++;
            while (  A[d] > perno) d--;
            if (s > d) break;
            exchange(A[s], A[d]);
            s++;
            d--;
        };

        if (inf < d)
            quickSort(A, inf, d);
        if (s < sup)
            quickSort(A, s, sup);
}
```

La complessità del while è O(n)

	0	1	2	3	4
Array A	**2**	**5**	**3**	**1**	**7**

quicksort(A, 0,4)

Inf=0 , Sup=4, perno=A[2]=3

0	1	2	3	4
2	**5**	**3**	**1**	**7**
s				d

while

0	1	2	3	4
2	**5**	**3**	**1**	**7**
	s		d	

Scambio, poi s++, d--

0	1	2	3	4
2	**1**	**3**	**5**	**7**
	s, d			

while

0	1	2	3	4
2	**1**	**3**	**5**	**7**
		s, d		

Scambio, poi s++, d--

0	1	2	3	4
2	**1**	**3**	**5**	**7**
	d		s	

s>d: fine del while

0	1	2	3	4
2	**1**	**3**	**5**	**7**
	d		s	

Due chiamate ricorsive:

```
quicksort(A, 0, 1);
quicksort(A, 3, 4);
```



```
quicksort(A, 0, 1);
```

Inf=0 , Sup=1, perno=2

0	1	2	3	4
2	**1**	**3**	**5**	**7**
s	d			

while

0	1	2	3	4
2	**1**	**3**	**5**	**7**
s	d			

Scambio, poi s++, d--

0	1	2	3	4
1	**2**	**3**	**5**	**7**
d	s			

s>d: fine del while,
inf=d, sup=s:
nessuna chiamata ricorsiva

quicksort(A, 3, 4);
Inf=3 , Sup=4, perno=5

0	1	2	3	4
1	**2**	**3**	**5**	**7**
			s	d

while

0	1	2	3	4
1	**2**	**3**	**5**	**7**
			s,d	

Scambio, poi s++, d--

0	1	2	3	4
1	**2**	**3**	**5**	**7**
		d		s

s>d: fine del while,
inf>d, sup=s:
nessuna chiamata ricorsiva

Quicksort

T (1) = a
T (n) = bn + T(k) + T(n-k)

Se k=1:

T (1) = a
T (n) = bn + T(n-1)

$O(n^2)$

Se k=n/2?

T (1) = a
T (n) = bn + 2T(n/2)

$T(1) = a$
$T(n) = bn + 2T(n/2)$

$T(1) = a$

$T(2) = 2b + 2a$

$T(4) = 4b + 4b + 4a$

$T(8) = 8b + 8b + 8b + 8a = 3\,(8b) + 8a$

$T(16) = 16b + 16b + 16b + + 16\,b + 16a = 4\,(16b) + 16a$

.

.

$T(n) = (n \log n)\, b + na$

T(n) è O(nlogn)

caso migliore e caso medio

La complessità nel caso medio è uguale a quella nel caso migliore: O(nlogn) (ma con una costante nascosta maggiore). Questo se tutti i possibili contenuti dell'array in input (tutte le permutazioni degli elementi) sono equiprobabili.

Per ottenere questo risultato indipendentemente dal particolare input, ci sono versioni "randomizzate" di quicksort in cui il perno ad ogni chiamata è scelto in modo casuale.

Ricerca in un insieme

```
int RlinearSearch (int A [], int x, int m, int i=0) {
  if (i == m) return 0;
  if (A[i] == x)  return 1;
  return RlinearSearch(A, x, m, i+1);
}
```

T (0) = a
T (n) = b + T(n-1)

O(n)

Ricerca in un insieme

```
int binSearch (int A [],int x, int i=0, int j=m-1) {
   if (i > j) return 0;
   int k=(i+j)/2;
   if (x == A[k]) return 1;
   if (x < A[k]) return binSearch(A, x, i, k-1);
         else return binSearch(A, x, k+1, j);
 }
```

T (0) = a
T (n) = b + T(n/2)

soluzione

$T(0) = a$
$T(n) = b + T(n/2)$

$T(0) = a$

$T(1) = b + a$

$T(2) = b + b + a$

$T(4) = b + b + b + a$

.

.

$T(n) = (\log n + 1)\, b + a$

T(n) è O(logn)

Ricerca in un insieme

```
int Search (int A [],int x, int i=0, int j=n-1) {
   if (i > j) return 0;
   int k=(i+j)/2;
   if (x == A[k]) return 1;
   return Search(A, x, i, k-1) || Search(A, x, k+1, j);
 }
```

$T(0) = a$
$T(n) = b + 2T(n/2)$

soluzione

$T(0) = a$
$T(n) = b + 2T(n/2)$

$T(0) = a$

$T(1) = b + 2a$

$T(2) = b + 2b + 4a = 3b + 4a$

$T(4) = b + 6b + 8a = 7b + 8a$

.

.

$T(n) = (2n-1)b + 2n\,a$

T(n) è O(n)