Ricorsione

più efficente e più semplice degli iterativi.
costruisco approccio bottom-up.
Non vanno usati sempre ma neanche mai.

fattoriale iterativo

```
int  fatt(n){   iterativo
       fatt=1
       for(i=1 to  n    ){
           fatt. i
       }
       return  fatt
}
```

ric

$fatt(n) = fatt(n-1) \cdot n$

$fatt(0) = 1$

```
int potenza (int a, int n) // a>0  n≥0
c.1  if(n==0){
c.t.if    return 1
     }
     else {
2.C.f.if  {  ris = a*potenza(a,n-1)
             return ris
     }
}
```

$T(n) = c + Ct_{if} + 3ct_{if}$ <u>EH VOLEVI!</u> quando chiamo potenza non eseguo una istruzione, eseguo una f.ne

$$T(n) = \begin{cases} 2c + T(n-1) & n>0 \\ 2c & n=0 \end{cases}$$ equazione di ricorrenza

ricorsivo quante volte z in array di char

caso base: ho un array di 1 elemento che è z (rispondo 1)
se non lo è rispondo 0
Oppure
quando ho l'array vuoto

```
int trovaZ(V[], int f)

C if( F<=0){
Cthen return 0   caso base, la porzione che controllo è vuota
}else
C  if(V[f]="z"){
C      ris1=1
   }else{
       ris1=0
   }
C  risz = trovaZ(V,f-1)
C  return ris1+risz
}
```

Alla fine non conta quale fa perché hanno gli stessi tempi

$$T(n) = \begin{cases} 2C & n=0 \quad \text{caso base} \\ 5C + T(n-1) & n>0 \end{cases}$$

↑ la chiamata ricorsiva

$T(n) = 5C + T(n-1) = 5C + [5C + T(n-2)] = 2 \cdot 5C + [T(n-2)] =$
$2 \cdot 5C + [5C + T(n-3)] =$
$3 \cdot 5C [5C + T(n-4)]$

Non conviene

$T(n) = K \cdot 5C + T(n-K)$ in generale

$T(n) = n \cdot 5C + T(0)$

$n\,5 \cdot C + 2C = C(5n+2) = \Theta(n)$

```
int contaSucc(V[], int f)
    if(F < 2){
        return 0
    }else
        if (V[f]==V[f-1]+1){
            ris1=1
        else {
            ris1=0
        }
        ris2 = trovaz(V,f-1)
        return ris1+ris2
    }
}
```

$MCD(0, n) = n$

$MCD(m, n) = MCD(m, n-m)$

$MCD(m, n) = MCD(n, m)$

```
int mcd (int n, int m)
    if (m>n){
        app=m
        m=n
        n=app
    }
    if (m==0){
        return n
    }
    else {
        ris= mcd ( m, n-m )
        return ris
    }
}
```

$n > m$ sempre perché controllo ed eventualmente scambio

caso base

caso migliore $n = m$ faccio una sola chiamata ricorsiva (senza considerare caso banale di uno dei due $= 0$)

```
void f_y(A[], int i){
```

stampa un array

```
    if(i< length(A)){
        print A[i]        stampa in ordine
        f_y(A, i+1)
    }
}
```

```
f_y(A, i+1)        stampa al contrario
print(A[i])
```

caso base: $i > length$ ho finito e non faccio nulla

```
int somma(V[], int i, int f)
    if (i > f){
    return 0
    }
    else if (i == f){
    return V[i]
    }
    else {
        ris1 = V[f] + V[i]
        ris2 = somma(V, i+1, f-1)
        return ris1+ris2
    }
}
```

approssimo e sti cazzi

$$T(n) = \begin{cases} \Theta(1) \quad n=1 \text{ OR } n=0 \begin{cases} 2C & n=1 \\ 3C & n=0 \end{cases} \\ 4C + T(n-2) = \Theta(n) \end{cases}$$

$4C + T(n-2) = 4C + [4C + T(n-4)] = 2 \cdot 4C + T(n-4) = K4C + T(n-2k)$ assumiamo n pari

$K = \frac{n}{2}$

$\Downarrow$

$\frac{n}{2} 4C + T(0) =$

$2nC + 3C = \Theta(n)$

```
boolean Palindroma (V[], int i, int f){
    C if (i==f OR i>f){   //assumiamo che una parola senza char sia palindroma
        C return true
    }                          caso base
    else{
        C if( V[i] ≠ V[f]){
            CTif return false
        }
        else{
            C·fif ris = (palindroma (V, i+1, f-1))
            C·fif return ris
        }
    }
```

$$T(n)=\begin{cases} 2C & |\ n\leq 1 \\ 2C+C_{Tif}+C_{Fif}+f_{if}\cdot T(n-2) & |\ n>1 \end{cases}$$

caso migliore

V[1] ≠ V[n] esco subito e torno false

$f_{if}=0$ $n=\text{length}(v)$

$t_m(n)=2C+C+0+0=3C=\Omega(1)$

caso peggiore v é palindroma

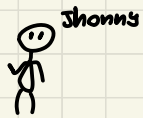
$T_p(n)=2C+0+C+T(n-2)$

$$T(n)=3C+T(n-2)=3C\left[+3C+(T(n-4)\right]=2\cdot 3C+\left[T(n-4)\right]=$$

$$2\cdot 3C+\left[3C+T(n)-6\right]=3\cdot 3C+T(n)-6=\boxed{K\cdot 3C+T(n-2K)}$$

Assumiamo n pari $n-2K=0$ $K=\frac{n}{2}$

$K=\frac{n}{2}$ $T(n)=\frac{n}{2}\cdot 3C+T(n-2\frac{n}{2})=\frac{n}{2}\cdot 3C+T(0)=\frac{3}{2}nC+2C=O(n)$




```
int max(A[ ], int i ){
    if(i==length(A) ) {
        return(i)
    }
    else{
       max_r= max(A, i+1)
        if ( A[max_r ] > A[i]){
            return(max_r )
        }
        else{
            return(i)
        }
    }
}
```