Prestazioni : proc. / tempo / effic. CREW → EREW

Fase 1) $p(n) = n^2$ $T(n, n^2) = 4$ (LD, LD, JZ, ST)

al solito posso ottenere:

$p(n) = n^2/\log n$ $T = \log n$

Fase 2) n moduli SOMMATORIA $p = n/\log n$ $t = \log n$

$p(n) = n^2/\log n$ $T \sim \log n$

Fase 3) $p(n) = n$ $T = 3$ (LD, LD, ST)

TOT. $p \sim n^2/\log n$ $T \sim \log n$

$$E = \frac{n \log n}{n^2/\log n \cdot \log n} = \frac{\log n}{n} \to 0$$

Algoritmi di ordinamento parallelo

- counting-sort $\quad E = \frac{\log n}{n} \to 0$
- bit-sort $\quad E = \frac{1}{\log n} \to 0$ lentamente

↑ lo vediamo

- Cole (1988) $\quad E = \text{Costante} \neq 0$

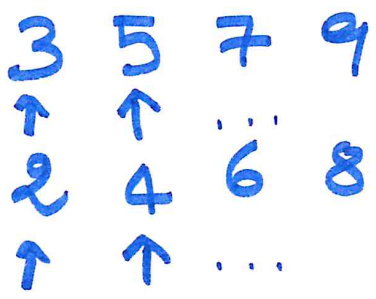
↑ non lo vediamo

Algoritmo sequenziale Mergesort ← TECNICA DIVIDE ET IMPERA

```
Procedura Mergesort ( A[1], A[2], ...., A[n] )
{   if ( |A| > 1 )
        { As ← Mergesort ( A[1], ..., A[n/2] )
          Ad ← Mergesort ( A[n/2+1], ..., A[n] )
           A ← merge ( As, Ad )
        }
        return ( A );
}
```

routine merge:

Esempio:



A_s di lung. $\frac{n}{2}$

A_d di lung. $\frac{n}{2}$

A di lung. n

quanto costa? (n caso peggiore)

Tempo di mergesort

$$t(n) = \begin{cases} 0 & n = 1 \\ 2\,t(n/2) + n & \text{altrimenti} \end{cases}$$

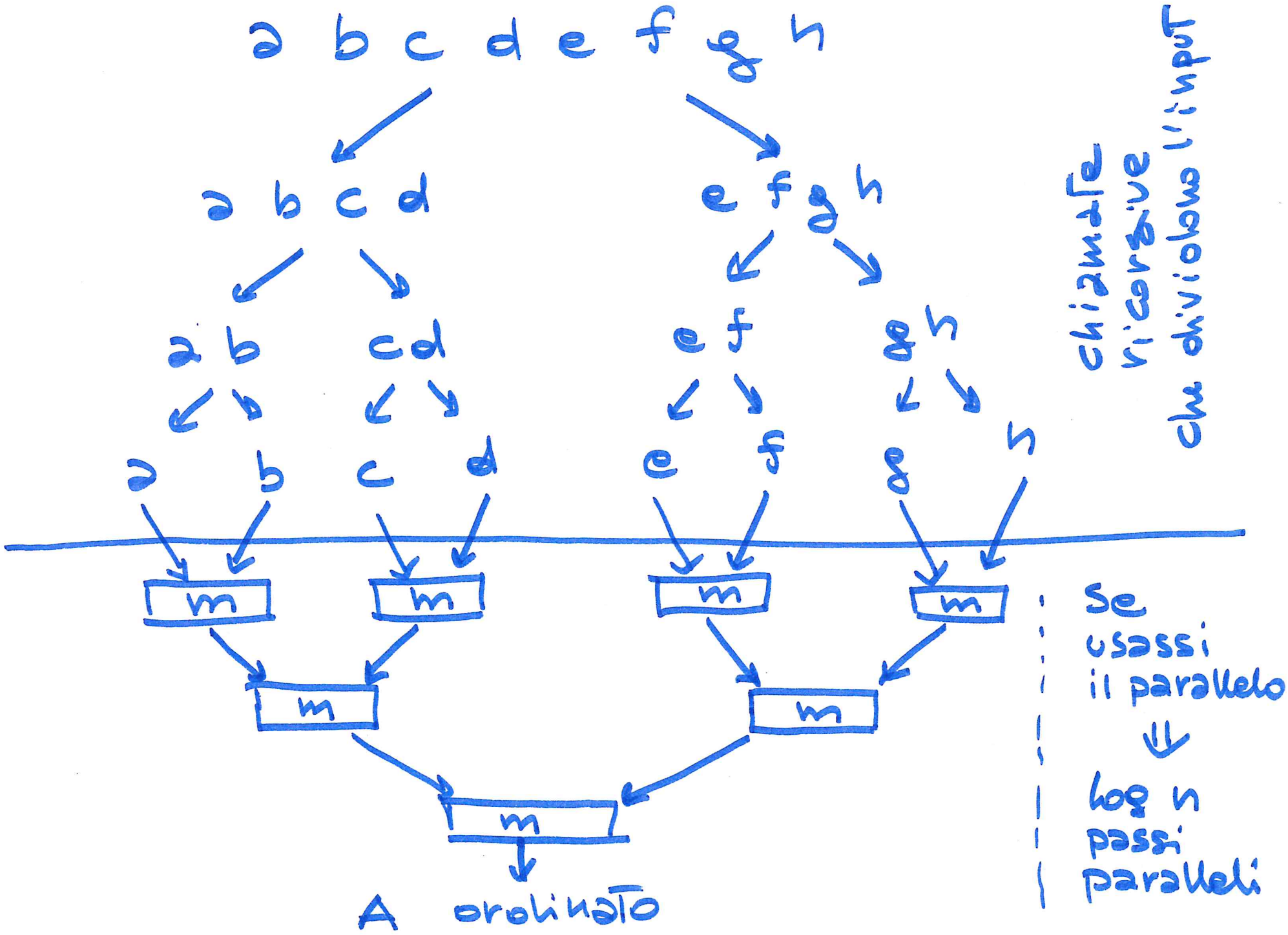
n: merge

$$t(n) = 2\,t\left(\tfrac{n}{2}\right) + n \sim 2\left(2\,t\left(\tfrac{n}{4}\right)\right) + n + n$$

$$\sim 2^k\, t\left(\tfrac{n}{2^k}\right) + kn$$

$$\sim \cancel{n \cdot 0} + \log n \cdot n = n \log n$$

Vediamo la dinamica dell'algoritmo per prendere ispirazione (su 8 elementi)

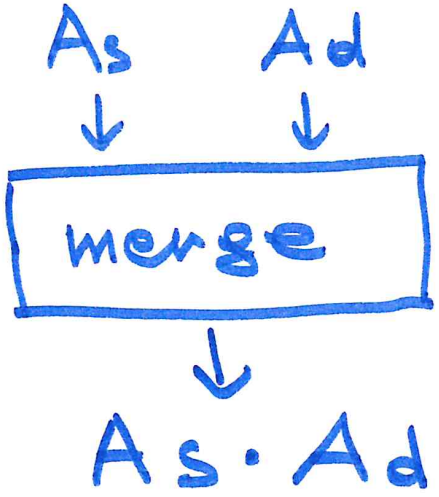

a b c d e f g h
a b c d
e f g h
a b
c d
e f
g h
a
b
c
d
e
f
g
h
m
chiamate ricorsive che dividono l'input
Se usassi il parallelo
log n passi paralleli
A ordinato

Purtroppo merge NON è parallelizzabile

e ottengo ancora $T \sim n \log n$

Domanda:

quando merge è facile?



NOTA:

As e Ad sono ordinati

MA

se gli elem. di As sono tutti minori di Ad basta concatenarli

IDEA per un algoritmo parallelo (BIT-SORT) → + uso di sequenze di numeri particolari: BITONICHE

Operazioni elementari su sequenze:

- REV (A[1], A[2], ..., A[n])

  A[1] ← A[n] A[2] ← A[n-1], ..., A[n] ← A[1]

- MINMAX (A[1], A[2], ..., A[n])

A[1], ..., A[k], ..., A[n/2], ..., A[k+n/2], ..., A[n]

min { A[k], A[k+n/2] } max { A[k], A[k+n/2] }

Amin Amax

Algoritmi paralleli per queste op.

```
Procedura REV(A)                                   p = n/2
{ for 1 ≤ k ≤ n/2 par do
      SWAP( A[k], A[n+k+1] )                       ⇐ LD, LD
}                                                    ST, ST
                                                   t = 4
```

```
Procedura MINMAX(A)                                p = n/2
{ for 1 ≤ k ≤ n/2 par do
    if ( A[k] > A[k+n/2] )                         ← 1 operaz.
        SWAP( A[k], A[k+n/2] )                     ← 4 operaz.
                                                   T = 5
```

Particolari sequenze numeriche

Def. formali:

- UNIMODALE: A è unimodale sse $\exists$ k t.c.

$$A[1] > A[2] > \ldots > A[k] < A[k+1] < \ldots < A[n]$$

oppure

$$A[1] < A[2] < \ldots < A[k] > A[k+1] > \ldots > A[n]$$

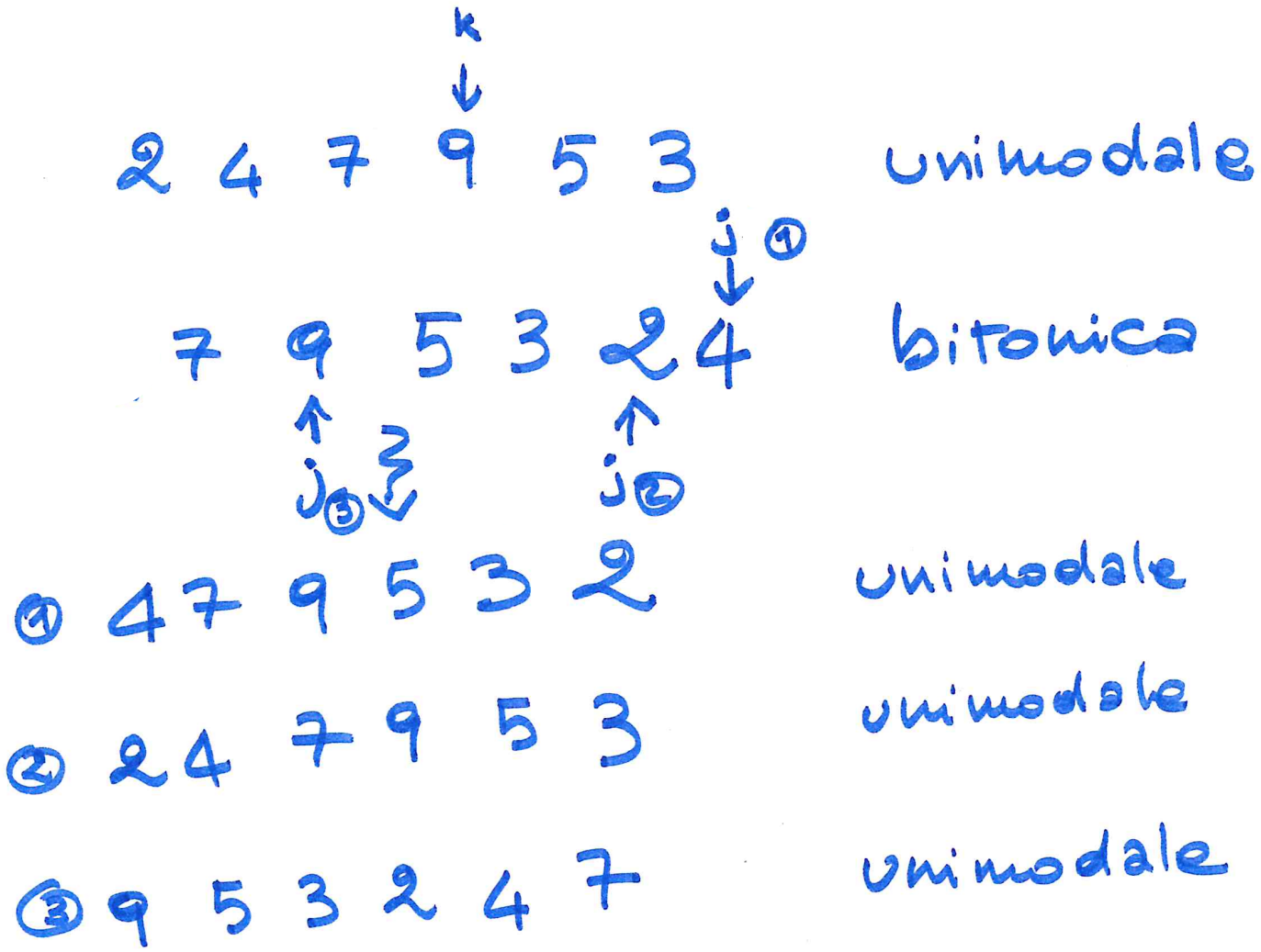
- BITONICA: A è bitonica sse $\exists$ una permutaz. ciclica di A che mi dà una seq. unimodale:

$\exists$ j t.c.

$A[j], \ldots, A[n], A[1], \ldots, A[j-1]$

è unimodale

Esempi:


k
2 4 7 9 5 3
unimodale
j ①
7 9 5 3 2 4
bitonica
j ③
k
j ②
① 4 7 9 5 3 2
unimodale
② 2 4 7 9 5 3
unimodale
③ 9 5 3 2 4 7
unimodale

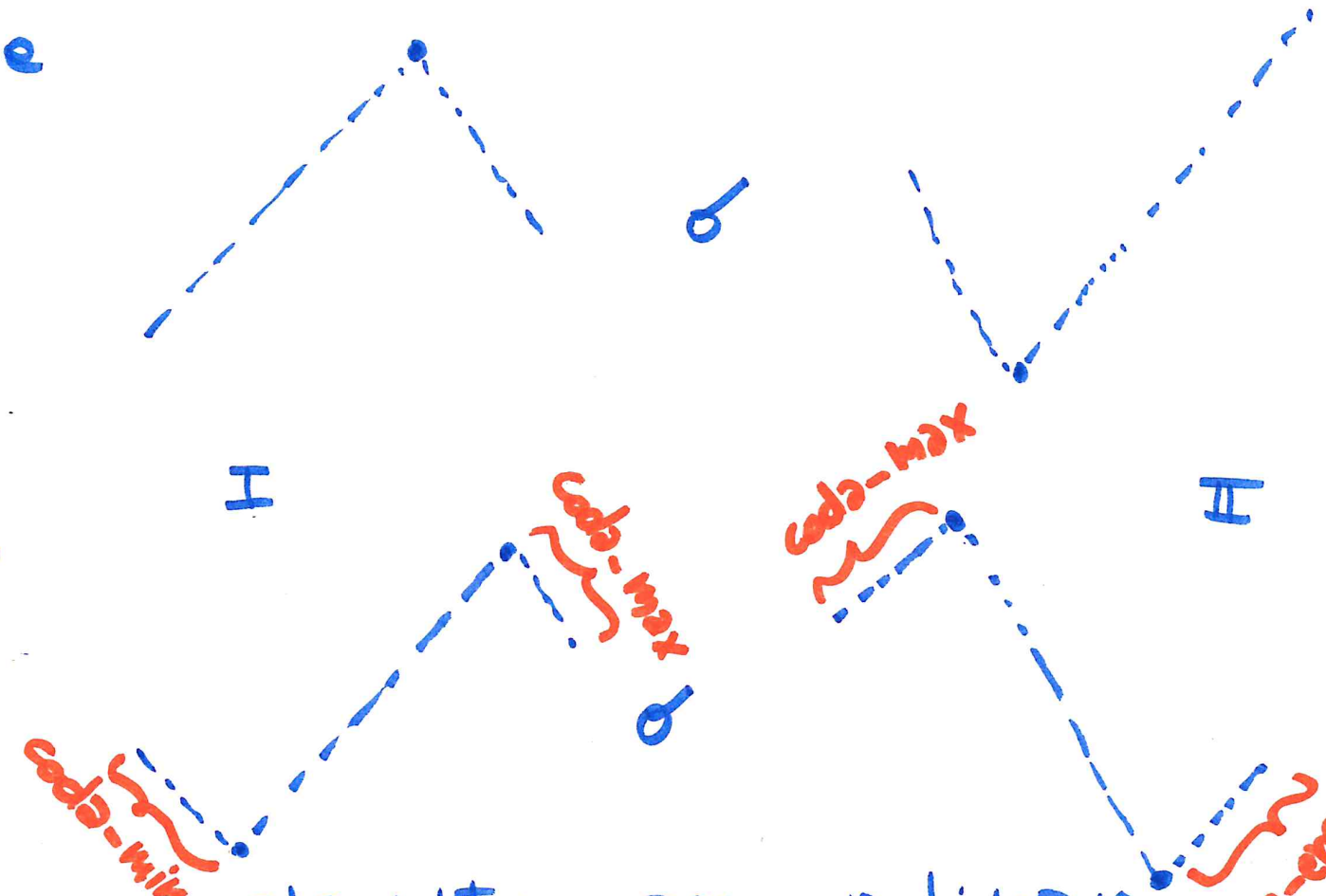
GRAFICAMENTE:

Unimodale

un picco

bitonica

due picchi



BIT-MERGE

Diamo ora un algoritmo per ordinare sequenze BITONICHE Batcher (1968)

Osservazioni:

① unimodale ⇒ bitonica
grazie alla permutazione identità

② Gli elementi di fine array devono essere maggiori degli elementi di inizio array (Forma I) - Viceversa minori nella Forma II.

③ siano A, B due sequenze ordinate crescenti (decrescenti) la sequenza

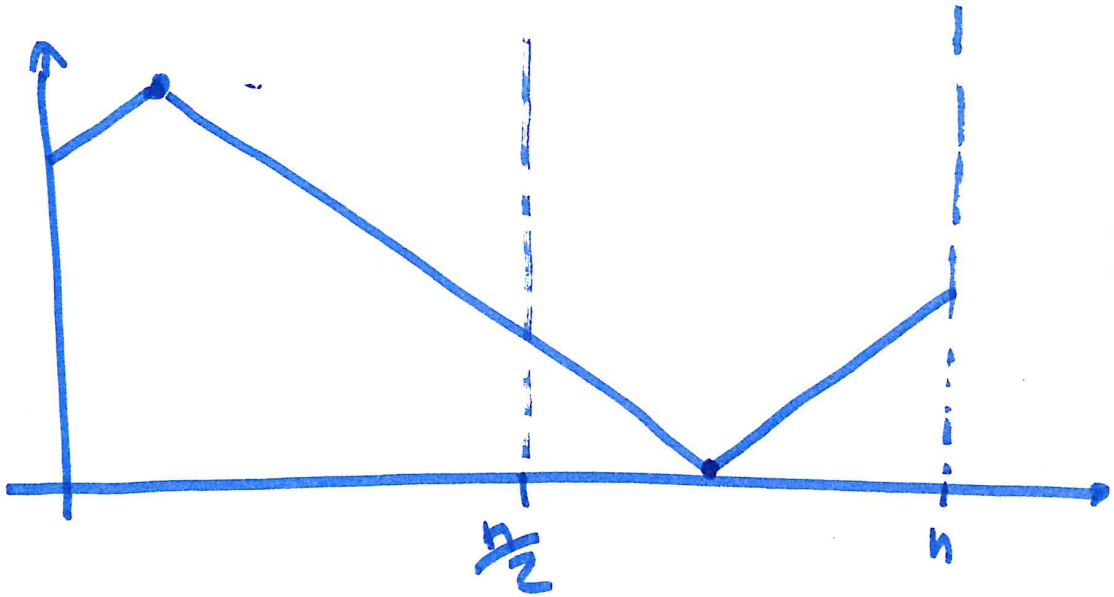
A · REV(B) è unimodale

Proposizione su sequenze BITONICHE:

Sia A bitonica, eseguo MINMAX(A) ottengo:

① Amin e Amax sono bitoniche

② Ogni elemento di Amin è minore di ogni elemento di Amax



A = sequenza BITONICA