Esercitazione
12 gennaio 2021

Problem Set 1

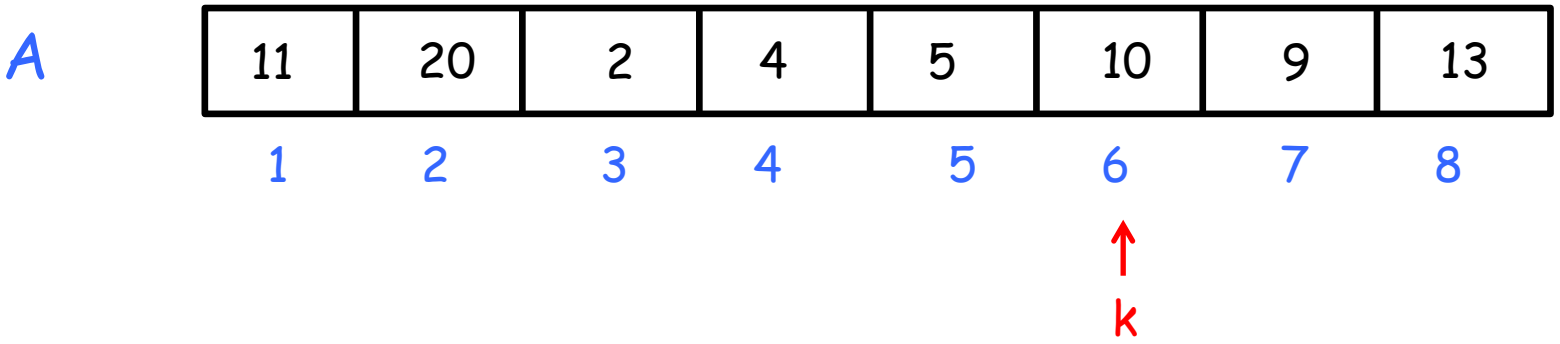
Esercizio 3:

trovare un picco in una dimensione

Progettare un algoritmo efficiente per il seguente problema.

Input: vettore A[1:n] di n numeri

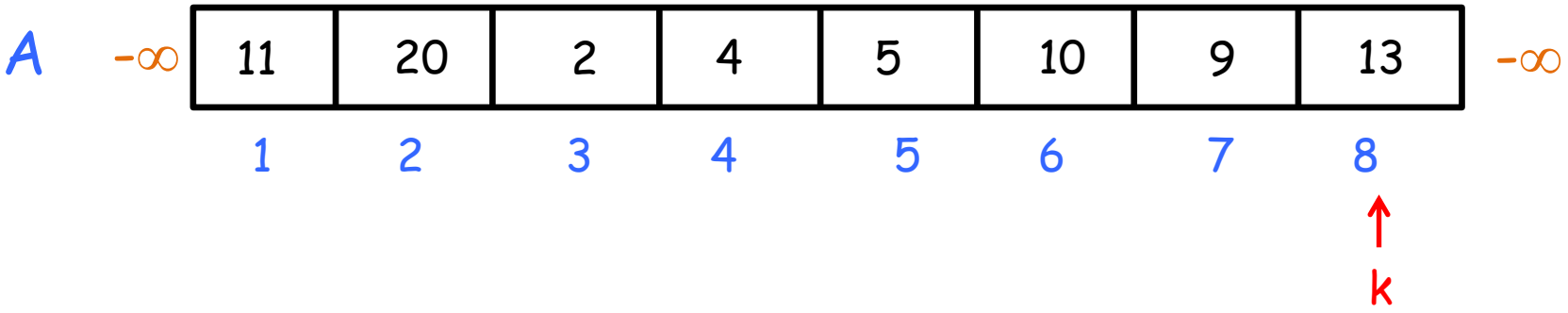
Output: l'indice k di un *picco*, ovvero un elemento tale che A[k] è maggiore o uguale dei suoi (al più) due elementi adiacenti.



Progettare un algoritmo efficiente per il seguente problema.

Input: vettore A[1:n] di n numeri

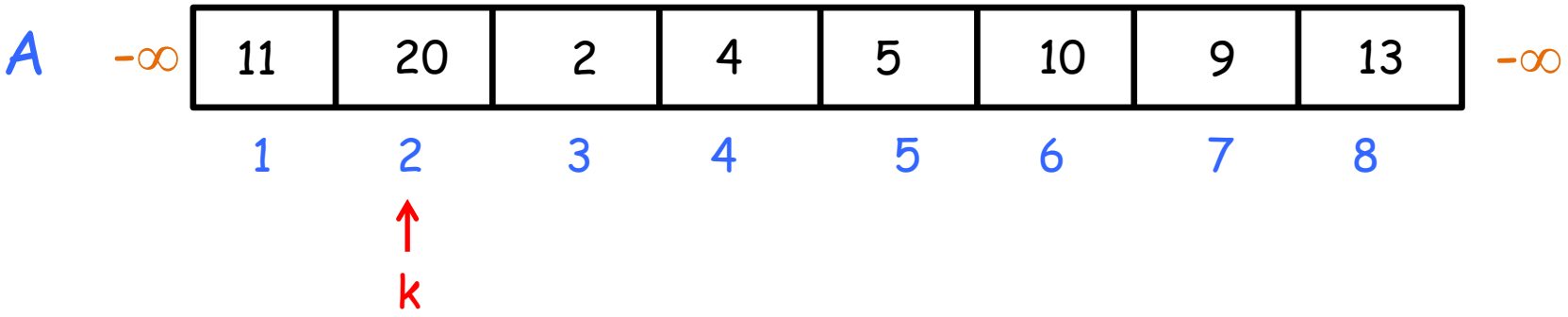
Output: l'indice k di un *picco*, ovvero un elemento tale che A[k] è maggiore o uguale dei suoi (al più) due elementi adiacenti.



Progettare un algoritmo efficiente per il seguente problema.

Input: vettore A[1:n] di n numeri

Output: l'indice k di un *picco*, ovvero un elemento tale che A[k] è maggiore o uguale dei suoi (al più) due elementi adiacenti.



osservazione: il massimo è sempre un picco

goal: $O(\log n)$

idea: uso l'approccio delle ricerca binaria.

```
1D-PiccoR(A, i, j)
1.   m=⌊ (i+j)/2⌋
2.   if  (A[m]≥ A[m+1] & A[m]≥ A[m-1]) then return m
3.   if  (A[m-1]> A[m]) then return 1D-PiccoR(A, i, m-1)
4.                          else   return 1D-PiccoR(A, m+1,j)
```

```
1D-Picco(A)
1.   if (A[1] ≥ A[2]) then return 1
2.   if (A[n] ≥ A[n-1]) then return n
3.   return 1D-PiccoR(A, 2, n-1)
```

correttezza?

il massimo della "metà" in cui ricorro è un picco

complessità?

$O(\log n)$

Esercizio 4:
trovare un picco in due dimensioni

Sia $M[1:n, 1:m]$ una matrice con n righe, m colonne, tale che $M[i,j]$ è un numero non negativo. Un *picco* in M è una posizione (p,q), $p \in \{1,\ldots,n\}$ e $q \in \{1,\ldots,m\}$ il cui elemento ha a sinistra, a destra, sopra e sotto elementi non più grandi di lui, ovvero $M[p,q] \geq \max\{M[p-1,q], M[p+1,q], M[p,q-1], M[p,q+1]\}$. Quando (p,q) è sul "bordo", la condizione è da considerarsi relativa solo agli elementi esistenti adiacenti a (p,q). Pertanto, per semplicità e in modo equivalente considereremo $M[i,j] = -\infty$, quando uno dei due indici i o j è minore di 1 o maggiore di n.

- Si consideri il seguente algoritmo che essenzialmente esegue una ricerca di un picco spostandosi sempre su elementi più grandi. Più in dettaglio, l'algoritmo parte da una data posizione (i,j). Se tale elemento non è un picco si controlla gli (al più 4) elementi adiacenti a (i,j), si sposta sull'elemento di valore massimo, e procede ricorsivamente. Si dimostri se tale algoritmo è corretto e se ne studi la complessità asintotica nel caso peggiore.

- Si progetti un algoritmo che trovi un picco in M con complessità temporale $o(nm)$ e che usi la tecnica *divide et impera*.

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

osservazione: il massimo è sempre un picco

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

picco!

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

Analisi algoritmo greedy (goloso) – punto uno

picco!

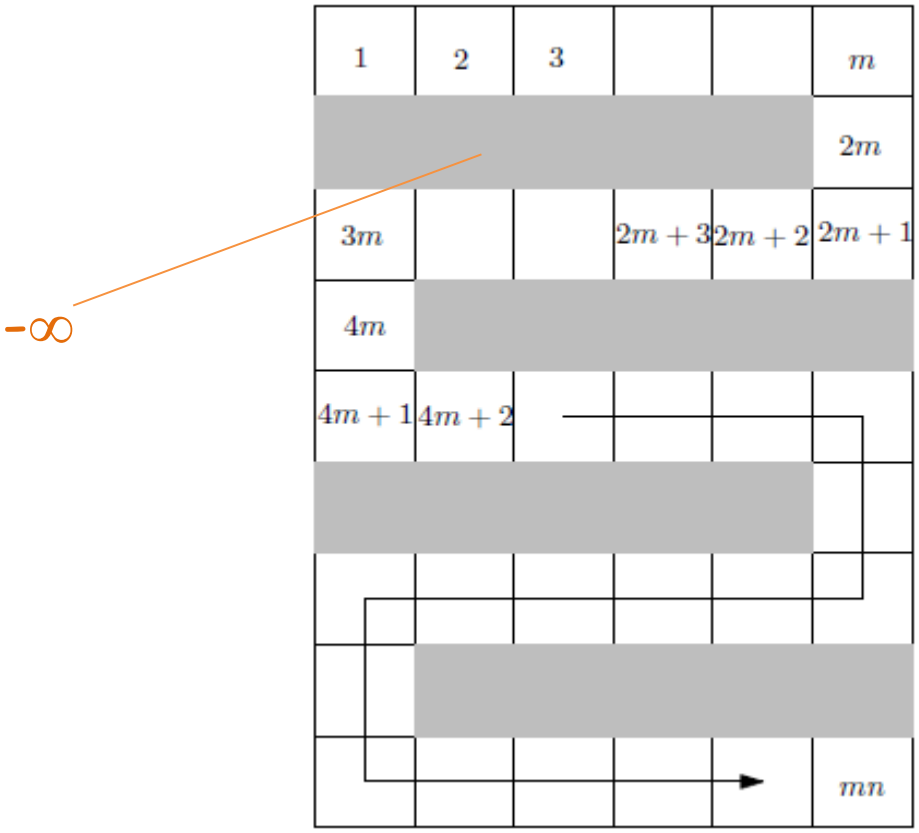
2	5	6	16	15
3	7	9	10	14
5	6	10	12	13
10	5	9	9	12
4	7	8	10	11

correttezza?

-ad ogni passo vedo un elemento strettamente più grande del precedente;
-non posso tornare su un elemento già visto (tutte elementi distinti);
-alla fine arrivo su un picco.

complessità?

Analisi algoritmo greedy (goloso) – punto uno

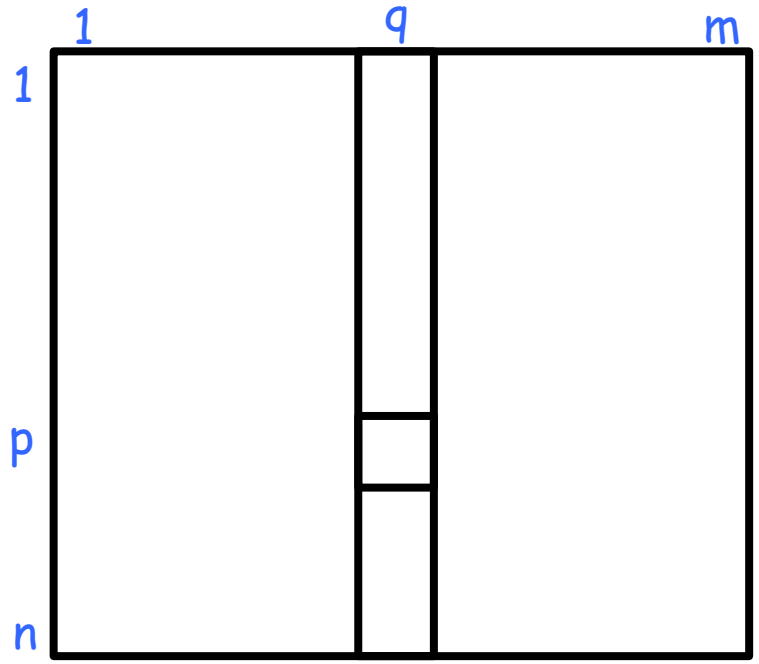


complessità?

$\Theta(nm)$

idea: ancora ricerca binaria!

-guardo la colonna in posizione centrale q
-cerco in tempo O(n) il massimo della colonna. Sia esso in posizione (riga) p
-se l'elemento M[p,q] è un picco lo restituisco, altrimenti c'è almeno uno dei due elementi M[p,q-1] o M[p,q+1] che è strettamente più grande di M[p,q].
-se M[p,q-1] > M[p,q] chiamo l'algoritmo ricorsivamente sulla sottomatrice formata dalle prime q-1 colonne; altrimenti lo chiamo sulla sottomatrice formata dalle colonne da q+1 a m

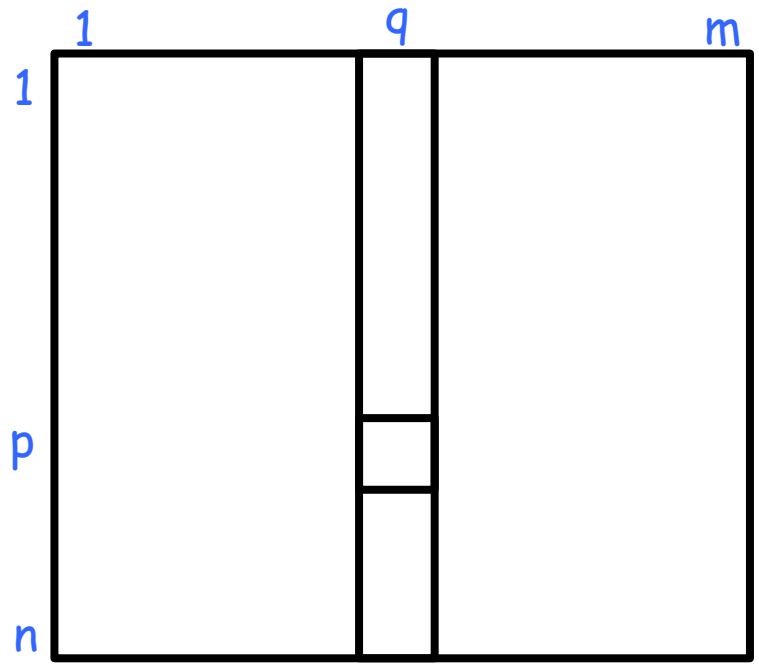


correttezza: dobbiamo solo argomentare che nella sottomatrice in cui si ricorre c'è un picco.

Due argomentazioni diverse:

- il massimo della sottomatrice in cui si ricorre è un picco.

- se si lanciasse l'algoritmo greedy a partire dall'elemento adiacente orizzontalmente a M[p,q] che è strettamente più grande di M[p,q] si fermerebbe su un picco nella sottomatrice in cui si ricorre.



complessità?

$T(n,m)=T(n,m/2)+O(n)$

$T(m)=T(m/2)+O(n)$

$O(n \log m)$

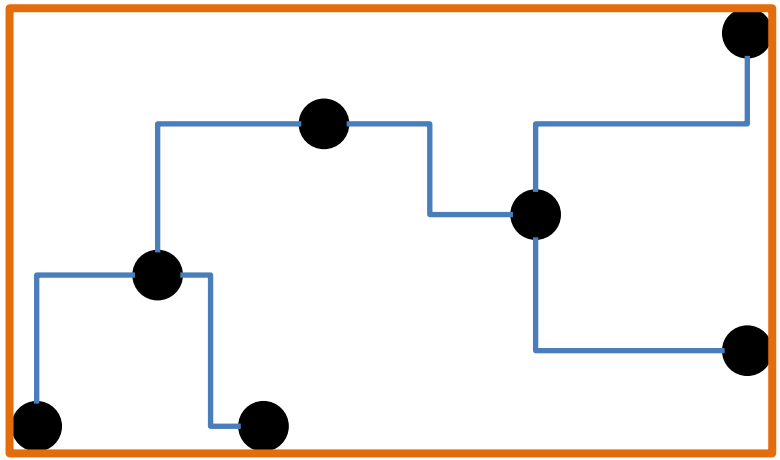
eventualmente scambiando i ruoli di n e m

$O(\min\{n,m\} \log (\max\{n,m\}))$

Esercizio 5:
embedding di un albero binario completo su una griglia bidimensionale

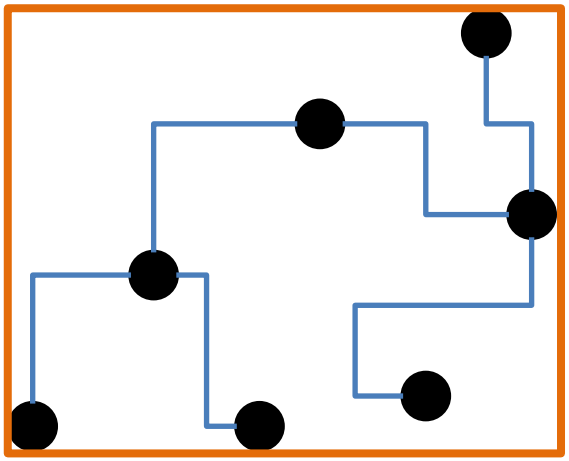
Sia dato un albero binario completo T con n foglie e una griglia bidimensionale (foglio a quadretti). Si vuole disegnare T sulla griglia in modo da non far intrecciare gli archi e minimizzando lo spazio utilizzato. Più precisamente, un *embedding* di T è un disegno di T sul foglio tale che: (i) i nodi di T sono disegnati sulle intersezioni della griglia (ovvero come cerchi centrati su un qualche spigolo di un qualche quadratino), (ii) gli archi sono delle linee "seghettate" che seguono le linee del foglio, (iii) le linee che rappresentano gli archi non possono incrociarsi reciprocamente. Dato un embedding di T il suo *bounding box* è il rettangolo più piccolo che contiene l'intero embedding. Trovare un embedding di T che minimizza (asintoticamente) l'area del bounding box.

n=4



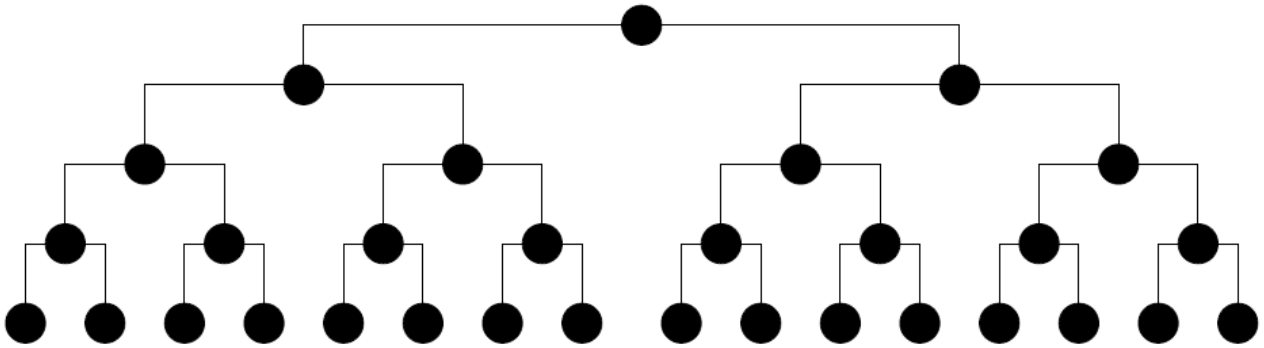
Sia dato un albero binario completo T con n foglie e una griglia bidimensionale (foglio a quadretti). Si vuole disegnare T sulla griglia in modo da non far intrecciare gli archi e minimizzando lo spazio utilizzato. Più precisamente, un *embedding* di T è un disegno di T sul foglio tale che: (i) i nodi di T sono disegnati sulle intersezioni della griglia (ovvero come cerchi centrati su un qualche spigolo di un qualche quadratino), (ii) gli archi sono delle linee "seghettate" che seguono le linee del foglio, (iii) le linee che rappresentano gli archi non possono incrociarsi reciprocamente. Dato un embedding di T il suo *bounding box* è il rettangolo più piccolo che contiene l'intero embedding. Trovare un embedding di T che minimizza (asintoticamente) l'area del bounding box.

n=4

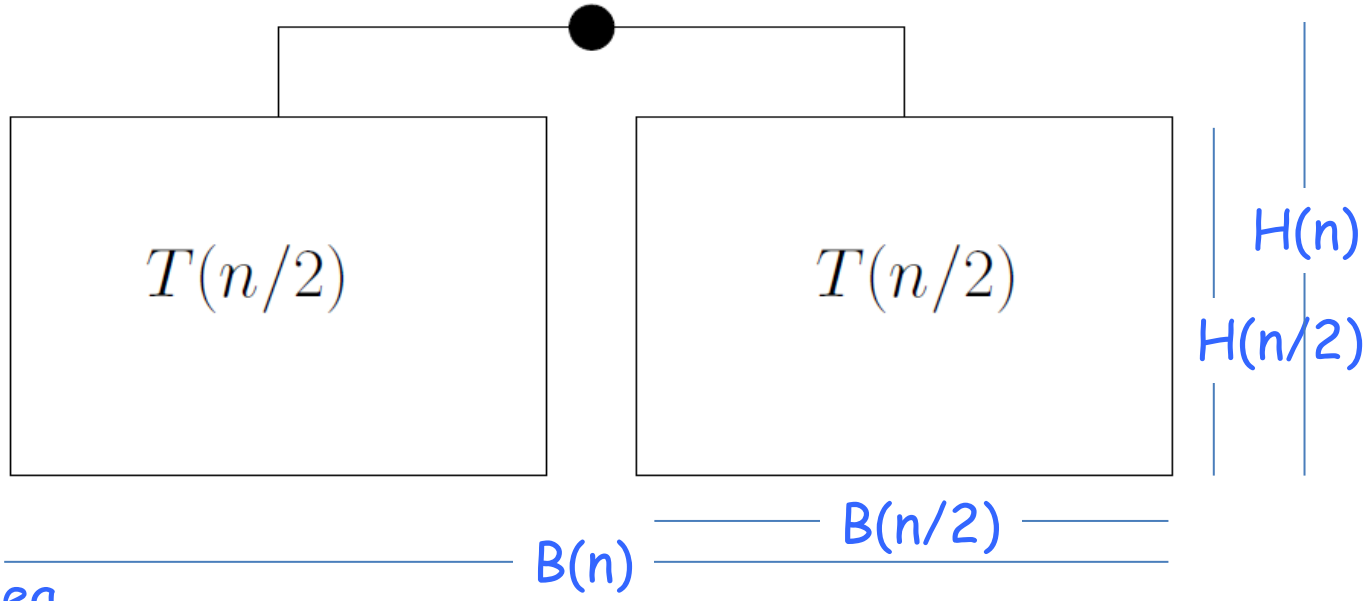


Una soluzione naturale

n=16



schema:



quanta area
(in funzione di n)?

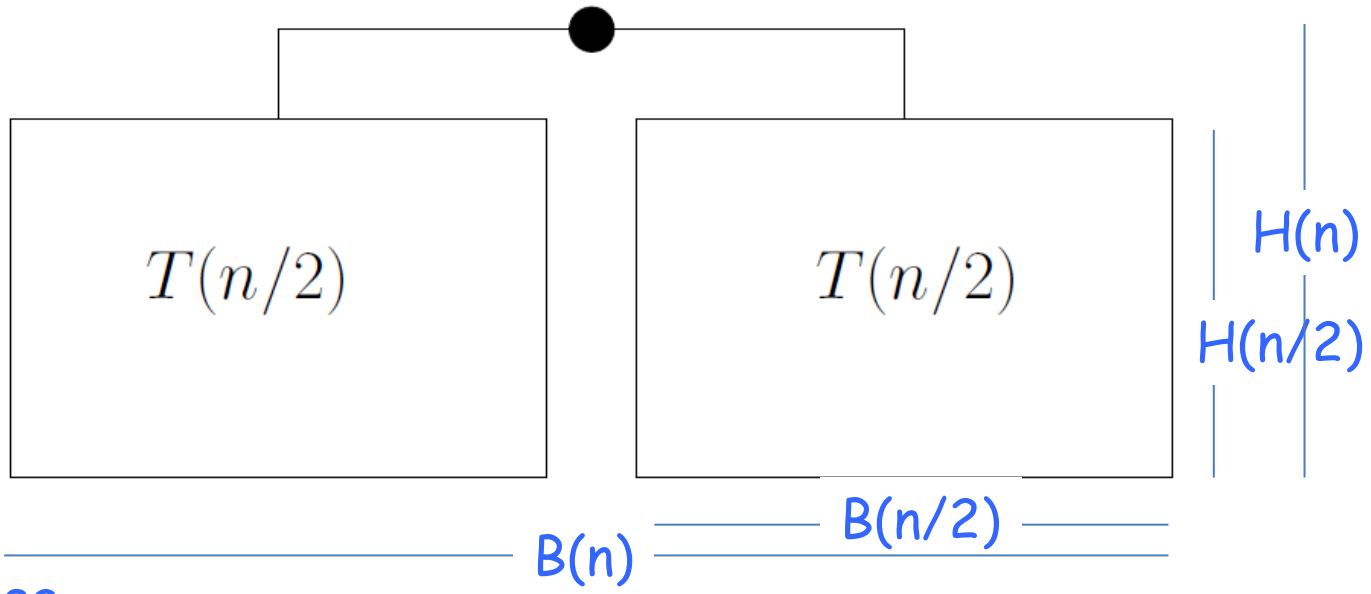
Una soluzione naturale

$H(n)=H(n/2)+O(1) \Rightarrow H(n)=\Theta(\log n)$

$B(n)=2B(n/2)+O(1) \Rightarrow B(n)=\Theta(n)$

$\Rightarrow$ area $\Theta(n \log n)$

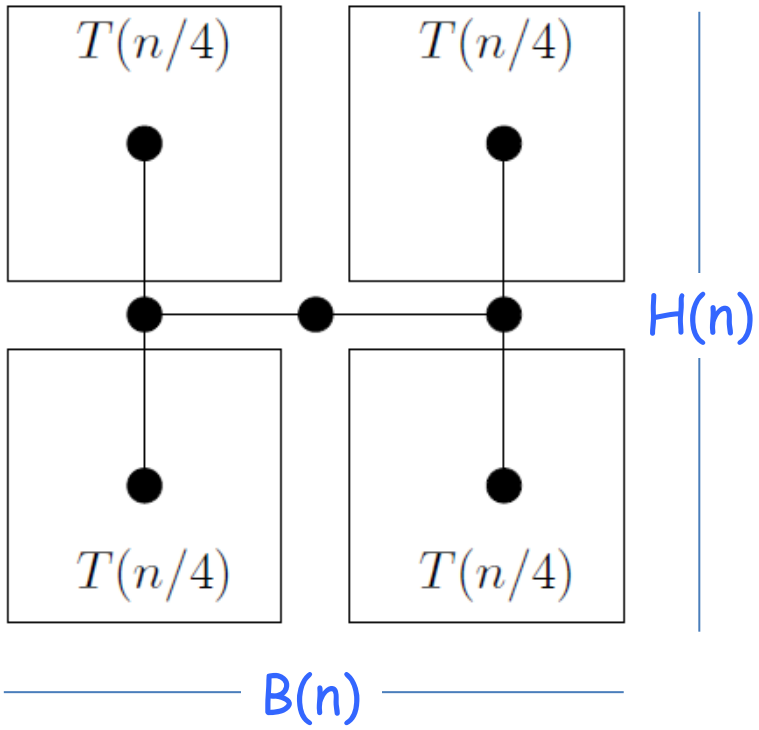
schema:



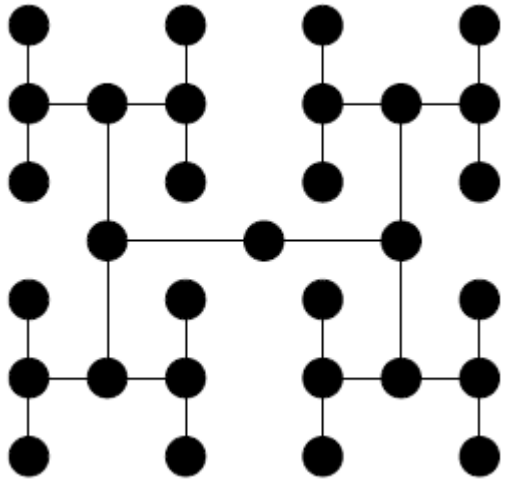
quanta area (in funzione di n)?

Una soluzione più furba

schema:



n=16



$H(n)=2H(n/4)+O(1) \Rightarrow H(n)=\Theta(\sqrt{n})$

$B(n)=2B(n/4)+O(1) \Rightarrow B(n)=\Theta(\sqrt{n})$

$\Rightarrow$ area $\Theta(n)$

quanta area (in funzione di n)?