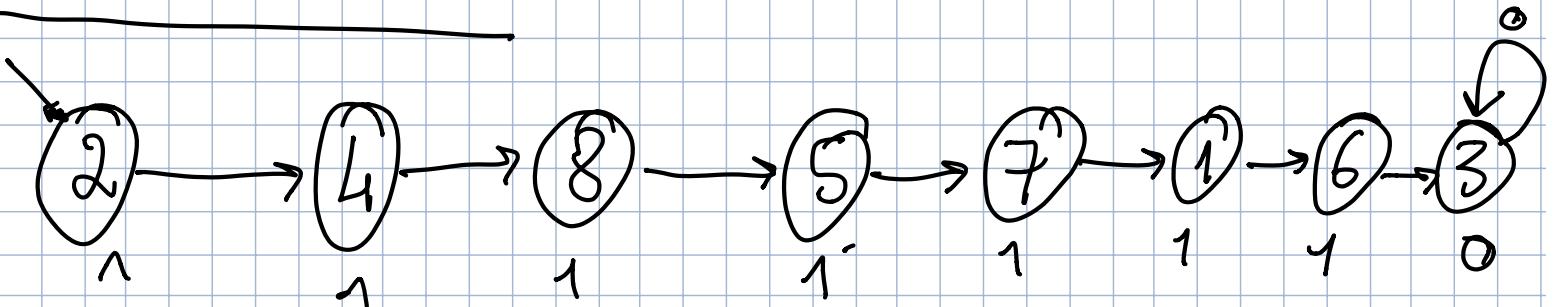
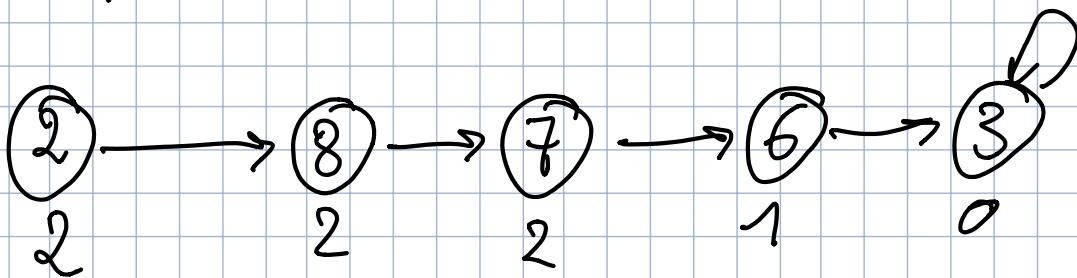


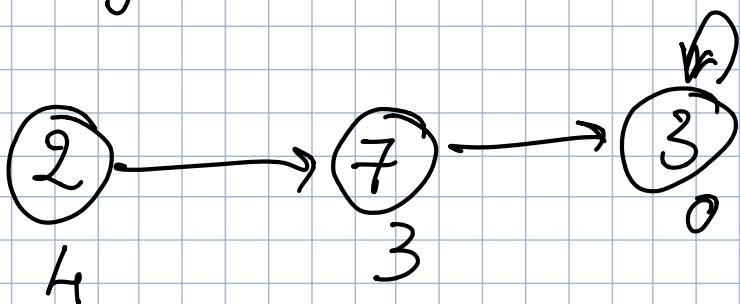
Esercizio 1



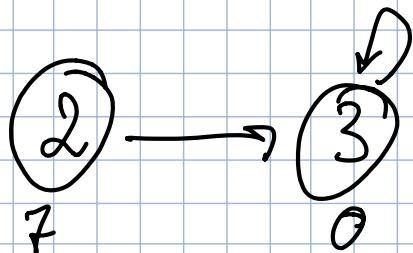
$$I = \{4, 5, 1\}$$



$$I = \{8, 6\}$$



$$I = \{7\}$$



$$7 \rightarrow 3$$

$$8 \rightarrow 5$$

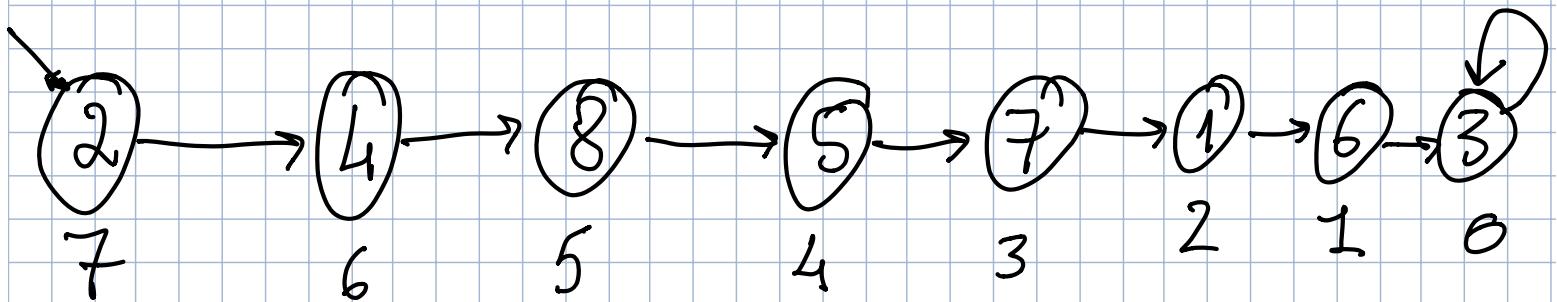
$$6 \rightarrow 1$$

$$\dots$$

$$1 \rightarrow 6$$

$$5 \rightarrow 4$$

$$1 \rightarrow 2$$



Complessità I/O di questo algoritmo:

Dipende dal modo in cui si crea la I:

$$T(n) = I(n) + O\left(\frac{n}{B}\right) + T\left(\left(1 - \frac{1}{C}\right)n\right)$$

\downarrow divide \downarrow ricombina \downarrow conquer

→ Con un turing random ho $\frac{1}{2}$ possibilità di prendere un intero. $\frac{n}{h}$ è la dimensione massima di I. Quindi $C = h$ e abbiamo

Queste procedure mi costo in media

$$O\left(\frac{n}{B}\right)$$

$$T(n) = O\left(\frac{n}{B}\right) + T\left(\frac{3}{4}n\right)$$

Anche il list ranking costa $O\left(\frac{n}{B}\right)$

→ Caso con turing deterministico:

Il costo è sempre $O\left(\frac{n}{B}\right)$ però al caso pessimo
è con algoritmo deterministico.

Per il list ranking abbiamo sempre
un numero di I/O pari a $O\left(\frac{n}{B}\right)$

Esercizio 2

A: 1, 2, 4, 9, 10, 11, 15, 19, 20, 24
↑

B: 3, 20

① Doubling: cerco 3 in A, parto

dai i con $i = 1$

mi fermo con $i + 2^k$ con $i = 1$ e $k = 1$

perché ottengo 3 e in posizione 3 abbiamo

$4 > 3$.

Cerco 3 nel range $[1 + i, 1 + 2^k]$. non lo
trovo quindi 3 non ve in output.

Ripeto da 3, mi fermo a

$3 + 2^k$ con $k = 3$ perché

loce dell'array. Quindi

abbiamo che 20 è nell'range

$[3, 3+8]$ e lo troviamo

Questo range quindi
nell'intersezione abbiamo
soltamente {20}.

② Shuffling algorithm:

A: 1, 2, 4, 3, 10, 11, 15, 19, 20, 24

B: 3, 20

$$h(x) = 2x + 3 \bmod 29$$

A: 5, 7, 11, 21, 23, 25, 1, 12, 14, 22

B: 9, 14

Sono di A e di B

A: 4, 5, 7, 11, 12, 14, 21, 22, 23, 25

B: 9, 14

Scrivere in binario ogni numero:

1010110101010101

4	00100
5	00101
7	00111
11	01011
12	01100
14	01110
21	10101
22	10110
23	10111
25	11001
9	01001
14	01110

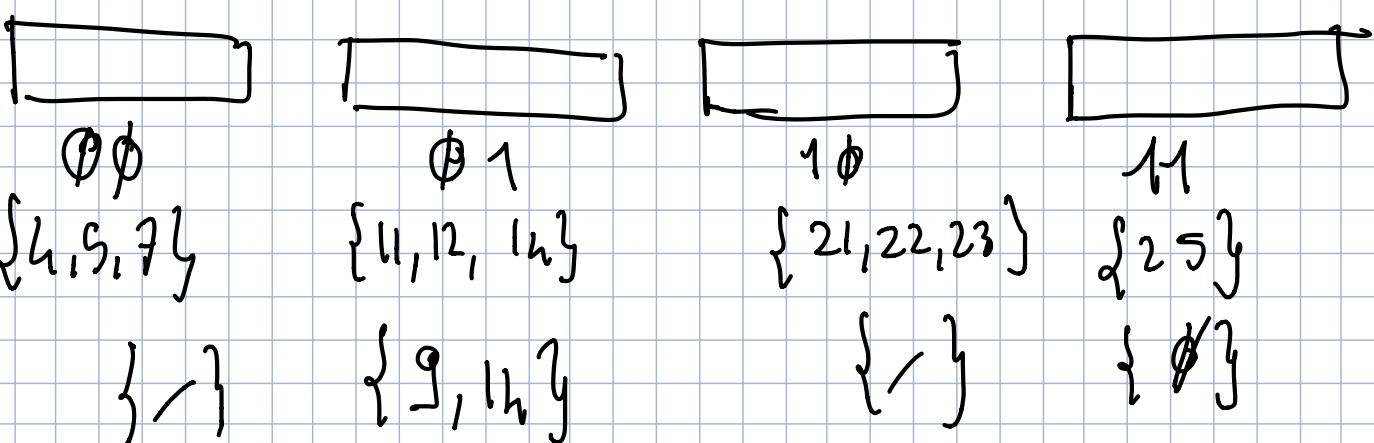
dim: $L=3$ $n=10$
 Creiamo $\frac{n}{L}$ blocchi = 4

gli elementi vanno
 nei blocchi

guardano i primi

$$\lg_2 \frac{n}{L} \text{ bit} =$$

$$\lg_2 \left\lceil \frac{10}{3} \right\rceil = 2$$



\rightarrow Merge the $\{11, 12, 14\}$ e $\{21, 22, 23\}$
 lasci fuori da lk è nell'intersezione

Con π^{-1} trovo che lk è insieme a 20.

Esercizio 3

La Completé in tempo deve rimanere lo stesso ma lo spazio deve diventare $O(\log n)$

while ($j - i > 0$):

$r = \text{random item da } S$

$S_{<} = \text{elementi di } S < S[r]$

$S_{>} = \text{elementi di } S > S[r]$

$n_{<} = |S_{<}|$

$n_{\geq} = |S| - (|S_{<}| + |S_{>}|)$

if ($k \leq n_{<}$):

 if ($n_{<} > \frac{j+i}{2}$):

$j = i + n_{<}$

 else if ($n_{<} < \frac{j+i}{2}$)

 RanSelect($S_{<}, k$)

else if ($k \leq (n_{<} + n_{\geq})$)

 return $S[r]$

else if ($n_j > \frac{i+j}{2}$)

$$i = j - n_j$$

else RandSelect (S_2, k)

Esercizio 4

① Il deterministic coin tossing assegna ad ogni elemento un valore $\text{Cov}(i) = i - 1$.

Vogliamo ridurre questo valore a 6, per ridurlo consideriamo la rappresentazione in bit di ogni $\text{Cov}(i)$.

Questo allora è $b = \log x + 1$

Ad ogni iterazione confrontiamo

bit i con bit $i+1$ e vediamo dove differiscono

quindi abbiamo che $\text{Cov}(i) = 2\pi_i + z_i$

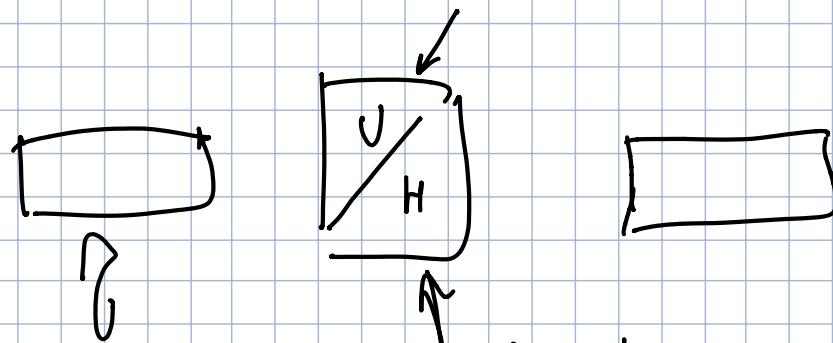
Ved dire che questo $\text{Cov}(i)$ lo esprimiamo con $\log b$ bit.

Dato che la procedura viene ripetuta più volte e che ogni volta diminuisce le dimensione di $\text{Cov}(i)$ di un

fattore logaritmico abbasso che
 l'algoritmo richiede $\lg(\lg(\lg \dots))$
 step over $\lg^{\star} n$ step

$$② \text{ heap} = \frac{2}{3} \text{ e non } \frac{1}{2}$$

$$\text{array} = \frac{1}{3}$$



n elementi

T in ingresso:

M vanno in U

$G - M$ vanno in H

output: $T - M + M = T$

$$\frac{1}{3} T = M \rightarrow \boxed{T = 3M}$$

Le lunghezze medie del sorted run
 possibile da $2M$ a $3M$

3) Il lower bound per il sorting delle stringhe è $\Omega(d + n \lg n)$

Dim: Assumiamo di prendere S stringhe binarie tali che i bit le distinguono e $\lg n$ sono gli altri dobbiamo

$$N = n(l + \lg n) \quad \text{e} \quad d = \Theta(N)$$

Il lower bound in questo caso è:

$$\Omega(d + n \lg n) = \Omega(N + n \lg n) = \Omega(N)$$

Con i classici algoritmi è:

$$\Omega((l + \lg_2 n)n \lg n) = \Omega(N \lg n)$$

quindi siamo distanti dall'ottimo per un fattore $\lg n$.
