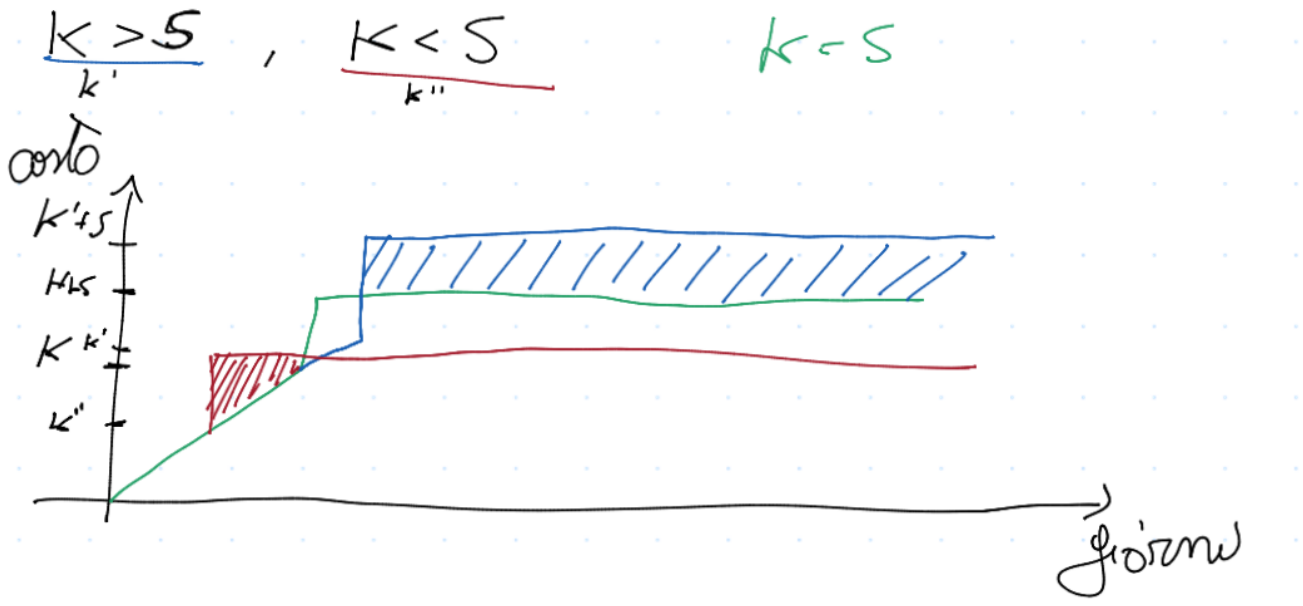
Esercizi5

Es. 1



Es. 2

Creiamo un algoritmo con un parametro i t.c. $0 \leq r < n$
Attendiamo per i giorni e la lista $V = v_1, \ldots, v_r$ saranno i giorni di cui conosciamo il valore perché non vendo niente, analizzo solo la sequenza dei valori per i giorni.
Dal giorno $r + 1$ fino ad $n - 1$, controllo ogni giorno se $v_r > Max(V)$. Se si verifica, vendo tutto altrimenti vendo tutto al giorno v_n.

```
V=v_1,...v_r
finché r<n-1:
```

```
        i++
        Se v_r > Max(v):
                vendi tutto
vendi tutto
```

La probabilità che dato r, l'algoritmo calcoli il massimo è:
$\frac{r-1}{n} \sum_{i=r}^{n} \frac{1}{i-1}$
oppure
$\frac{1}{n} \sum_{i=r}^{n} \frac{r-1}{i-1}$

La funzione è massimizzata per $r = \frac{n}{e}$

Es. 3

ALGORITMO LRU.

Ci sono 2 casi da analizzare:

a.

P è duplicato. ($\exists i | P = A_i$)
Affinché ci sia un page-fault su P bisogna che A_i sia il più vecchio valore ad essere letto. Ciò porta a dover effettuare $k-1$ page-fault. Poi una richiesta di una pagina ulteriore non presente in memoria affinché P venga rimossa dalla memoria e si può avere nuovamente un page-fault su P.
In totale ci vogliono k page-fault con pagine diverse da P.

Es.
$P = 4, k = 4$
$\delta = 1_p, 2_p, 3_p, 4_p, 1, 3, 7_p, 8_p, 4, 9_p, \dots$

dopo il 9 possiamo avere un page fault sul 4.
I page-fault sono indicati dal pedice p.

b.

$P \neq Q = A_i = A_j | i < j$

Tra A_i e A_j devono esserci almeno k page-fault da P.

Es.
$P = 4, Q = 5$
$\delta = 1_p, 2_p, 3_p, 4_p, 5_p, 6_p, 7_p, 8_p, 1_p, 5_p, \dots$

Es. 4

FwF è brutto ed inefficiente. Viene utilizzato per determinare se un algoritmo è addirittura meno competitivo di quello FwF
.

Affinché sia k-competitivo deve verificarsi:
$FwF(\delta) \leq k * OPT(\delta)$

Dove δ è la sequenza di richieste

$\delta = F_0, F_1, \dots, F_n$

Dove F_0 ha al più k fault e $\forall i | i \geq 1$ F_i contiene k page-fault.

OPT deve avere almeno un page-fault $\forall F_i$, quindi:

$FwF(\delta) \leq k(n+1) \leq k * OPT(n+1)$

Sia P una pagina per cui si verifica un page-fault in $F_i | i \geq 1$.
Se si è verificato un page-faut significa che in memoria ci sono k elementi diversi da P.
In seguito al page-fault, ci saranno altri $k-1$ page-fault che concluderanno la fase F_i.

Nel caso pessimo, ad ogni fase vengono rimosse $k-1$ pagine presenti nella fase precedente (almeno una per l'algoritmo OPT).

Es.

$P = 4, k = 3$
$\delta = 1_p, 2_p, 3_p, 4_p, 2_p, 3_p, \dots$

Ma 2 e 3 erano presenti in memoria nella fase precedente.
Un algoritmo LRU avrebbe consentito alla fase F_1 di essere più grande.

Es. 5

Provare che l'algoritmo $FIFO$ è k-competitivo bisogna dimostrare che $Costo_{FIFO}(\delta) \leq k * Costo_{OPT}(\delta) + a$ dove OPT è un algoritmo ottimo per il problema del page-fault. Sia $\delta = F_1, \dots, F_s$ dove F_i è una sequenza di pagine e n la dimensione della memoria. In F_0 ci sono al più n page-fault

mentre per le altre F_i ci sono esattamente n page-fault.
Nell'algoritmo ottimo ci sarà almeno un page-fault per ogni F_i quindi:

$Costo_{FIFO}(\delta) =$ Numero di page-fault con $FIFO$
$\leq k(s+1) = k*$ numero di fasi
$\leq k * Costo_{OPT}(\delta)$

Spiegare perché in OPT ci sarà almeno un page-fault.

Es. 6

$1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$

Es. 7

$Bubble - Sort$

Es. 8

È possibile creare una sequenza di richieste in cui ogni pagina è un page-fault.

Es.
$\delta = 1, 2, \ldots, k, k-1, k, k-1, k, \ldots$

Ci sono infiniti page-fault dopo la k-esima richiesta.

Poi un algoritmo ottimo OPT non è detto che in ogni finestra $F_i \mid i \geq 1$ per l'algoritmo $LIFO$, l'ottimo produca almeno un page-fault.

Affinché ci sia almeno un page-fault nell'algoritmo ottimo c'è bisogno di almeno k richieste a k pagine non presenti nella fase precedente.

Es. 9

Perché così simula una LRU che per gli algoritmi online è una buona approssimazione, mentre se spostiamo gli elementi alla coda, allora dovremmo scorrere tutto l'array prima di essere trovati nuovamente.

Ci salviamo la posizione di un elemento che sta in testa alla coda x e appena arriviamo al nodo che dobbiamo scambiare y siccome già lo avevamo esplorato in precedenza, dobbiamo fare uno swap senza costo.
Mentre se vogliamo scambiare un nodo della coda, dovremmo continuare ad avanzare lungo la lista fino a trovare la coda e successivamente fare lo swap.
Verranno quindi esplorati n elementi invece si $posizione(y)$ ogni volta che viene effettuata una add.

Es. 10

$\sum_{i=1}^{n} f_i = m = kn$
$\sum_{i=1}^{n} i f_i \leq \frac{kn(n+1)}{2}$

https://en.wikipedia.org/wiki/Rearrangement_inequality

$\sum_{i=1}^{n} i f_i \geq \sum_{i=1}^{n} i f_n$

$\sum_{i=1}^{n} i f_n \leq \frac{kn(n+1)}{2}$

$\frac{n(n+1)}{2} f_n \leq \frac{kn(n+1)}{2}$

$f_n \leq k$

Questo è vero sempre.
Supponiamo per assurdo che $f_n = k + 1$

Allora $\forall f_i \mid f_i \geq k + 1$
Ma allora $\sum_{i=1}^{n} f_i \geq n(k+1)$

Questo è assurdo perché $\sum_{i=1}^{n} f_i = nk$

Es. 11

```
b=1 // num. di camion disponibili
for i<-1 to m do //Per ogni a
        for j<-1 to b do //Per ogni camion
disponibile
                if a può essere inserito in Bj
                        inserisci ai in Bj
                        flag_inserito = TRUE
        if not flag_inserito:
                b=b+1
                insrisci ai in Bb
```

Il numero di camion b viene incrementato solo se non è stato possibile inserire l'articolo a_i in nessuno dei camion

disponibili.