Esercizio 35 Progettare una struttura dati per la gestione di un insieme dinamico di interi, con operazioni

- $New(S)$ crea un insieme vuoto;
- $Ins(S, x)$ inserisce l'elemento x nell'insieme S;
- $Half(S)$ cancella da S i $\lceil |S|/2 \rceil$ elementi più piccoli.

Si richiede che una qualsiasi sequenza di n operazioni venga eseguita in tempo $O(n \log n)$.

i. Specificare le strutture dati di supporto utilite e lo pseudo-codice delle operazioni suddette (questo può ridursi ad una chiamata di un'operazione della struttura scelta).

ii. Dimostrare, mediante un'analisi ammortizzata della complessità, che una sequenza di n operazioni costa $O(n \log n)$.

Soluzione: La struttura S può essere semplicemente un min-heap, con operazioni

```
New(S)
    crea un min-heap S
    S.size = 0

Ins(S,x)
    Heap-Insert(S,x)

Half(S,x)
    k = S.heapsize
    for i = 1 to k/2
        Heap-Extract-Min(S)
```

L'idea è che ogni operazione di inserimento paga in anticipo il costo dell'estrazione dell'elemento ($\log |S|$). Questo da' una complessità ammortizzata di $2 \log |S|$ per l'inserimento e complessità costante per *Half*, dato che le estrazioni sono già pagate. Si noti che in realtà un elemento non paga esattamente per la sua estrazione, ma per quella di un qualche altro elemento, a seconda dell'ordine relativo. Si può formalizzare con una funzione potenziale che può essere

$$\Phi(S) = \Sigma_{j=1}^{|S|} \log j = \log |S|!.$$

In questo modo, il costo delle operazioni è

	c	$\Delta\Phi$	$\hat{c}$
New	1	0	1
Ins	$1+ \log \|S\|$	$\log \|S\|$	$1+ 2 \log \|S\|$
Extract	$1+ \log \|S\|$	$- \log \|S\|$	1
Half	$1 + \Sigma_{j=1}^{\|S\|/2} log j$	$-\Sigma_{j=1}^{\|S\|/2} log j$	1

Quando la struttura è vuota, $\Phi(S) = 0$ e, per ogni S, vale $\Phi(S) \geq 0$. Pertanto per una sequenza di n operazioni, certamente $\Phi(S_n) - \Phi(S_0) \geq 0$, per cui i costi ammortizzati forniscono un upperbound per i costi reali. Una sequenza di n operazioni costa al più $\Sigma_{j=1}^{n} \log j \leq n \log n$, come desiderato.

Esercizio 2 (9 punti) L'ufficio postale offre un servizio di ritiro pacchi in sede su prenotazione. Il destinatario, avvisato della presenza del pacco, deve comunicare l'orario preciso al quale si recherà allo sportello. Sapendo che gli impiegati dedicano a questa mansione turni di un'ora, con inizio in un momento qualsiasi, si chiede di scrivere un algoritmo che individui l'insieme minimo di turni di un'ora sufficienti a soddisfare tutte le richieste. Più in dettaglio, data una sequenza $\vec{r} = r_1, \ldots, r_n$ di richieste, dove r_i è l'orario della i-ma prenotazione, si vuole determinare una sequenza di turni $\vec{t} = t_1, \ldots, t_k$, con t_j orario di inizio del j-mo turno, che abbia dimensione minima e tale che i turni coprano tutte le richieste.

i. Formalizzare la nozione di soluzione per il problema e il relativo costo. Mostrare che vale la proprietà della sottostruttura ottima e individuare una scelta che gode della proprietà della scelta greedy.

ii. Sulla base della scelta greedy individuata al passo precedente, fornire un algoritmo greedy `time(R,n)` che dato in input l'array delle richieste `r[1..n]` restituisce una soluzione ottima.

iii. Valutare la complessità dell'algoritmo.

Soluzione: Una soluzione è una sequenza $\vec{t} = t_1, \ldots, t_k$ tale che per ogni $i = 1, \ldots, n$ l'istante $r_i \in I(t_j)$ per qualche $j = 1, \ldots, k$, dove $I(t_j)$ indica l'intervallo di un'ora con inizio in t_j, Il costo è k.

Sottostruttura ottima Vale la sottostruttura ottima. Infatti se $\vec{t} = t_1 q \ldots, t_k$ è ottima per le richieste $\vec{r} = r_1, \ldots, r_n$ allora $t_2, \ldots, t_k$ è ottima per il sottoproblema $\vec{r'}$ che comprende le richieste r_i tali che $r_i \notin I(t_1)$. Infatti, se ci fosse un insieme di turni di dimensione minore di $k-1$ per servire le richieste in $\vec{r'}$, aggiungendo t_1 otterremmo una soluzione del problema originale $\vec{r}$ migliore di $\vec{t}$

Scelta greedy Assumiamo che la lista di richieste $\vec{r} = r_1, \ldots, r_n$ sia ordinata in modo crescente. La scelta greedy consiste nel considerare come primo turno $t_1 = r_1$.

Esiste sempre una soluzione ottima che la contiene. Infatti se $\vec{t'} = t'_1 \ldots t'_k$ è una qualsiasi soluzione ottima e supponiamo che i turni siano ordinati anch'essi in senso crescente, allora certamente $t'_1 \leq r_1$, dato che la prima richiesta deve essere servita. Quindi se sostituiamo t'_1 con $t_1 = r_1$, otteniamo una nuova soluzione (tutte le richieste servite da t'_1 sono anche servite da t_1!), anch'essa ottima.

Ne segue l'algoritmo che riceve in input l'array delle richieste $r[1..n]$ (che si assume non vuoto), e fornisce in uscita $t[1, k]$.

```
time(r,n)
  t[1] = r[1]
  turni=1
  for i = 2 to n
    if t[turni] < r[i]
       turni++
       t[turni] = r[i]
  return t
```

La complessità è $\Theta(n)$.