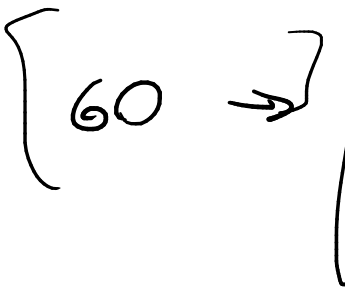
07/12

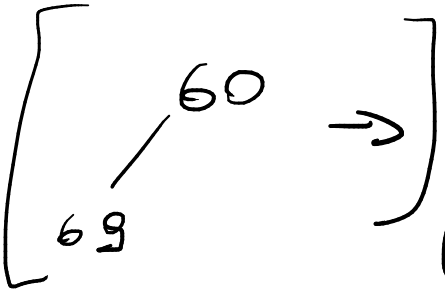
[MIN-HEAP] → 60, 69, 26, 95, 51, 24, 46

80, 60, 38, 12, 70

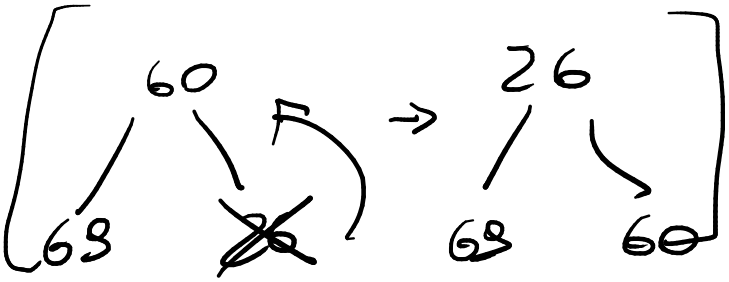
1°



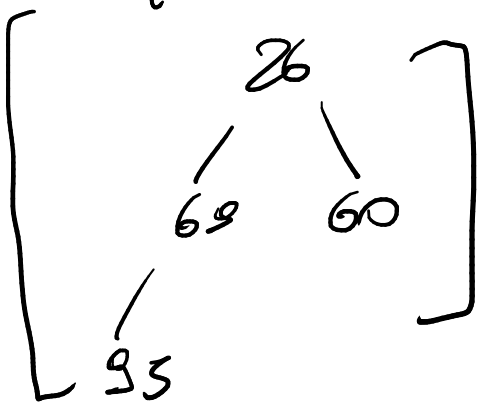

2°



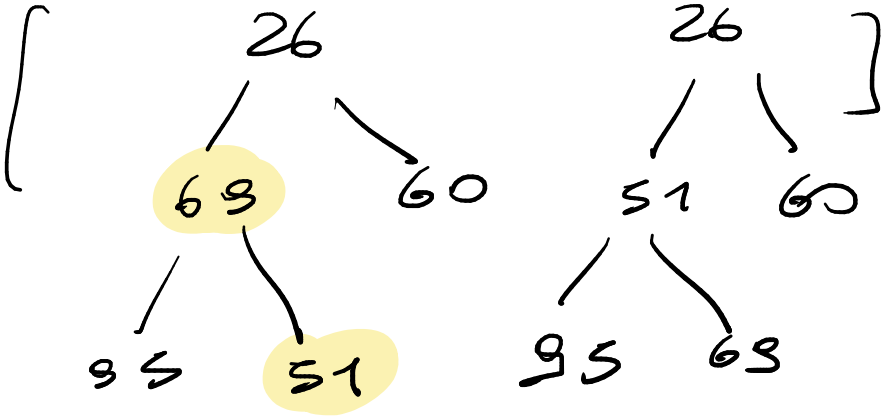

3°



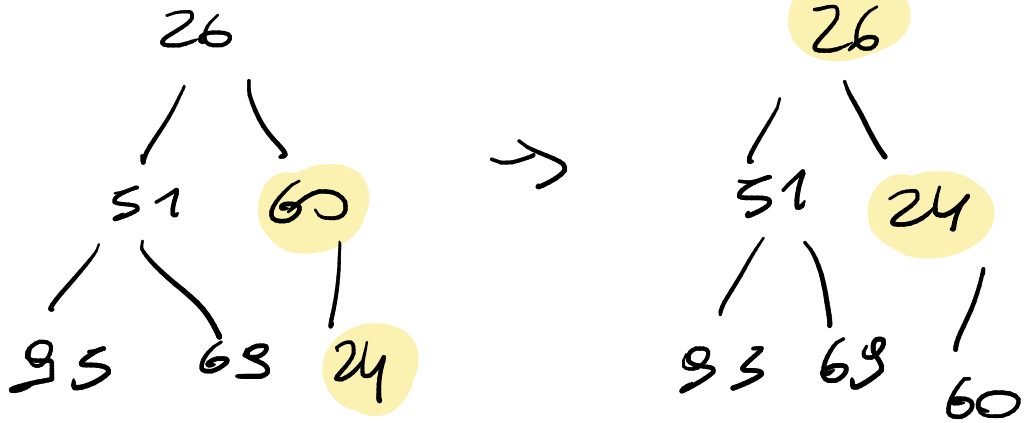

4°

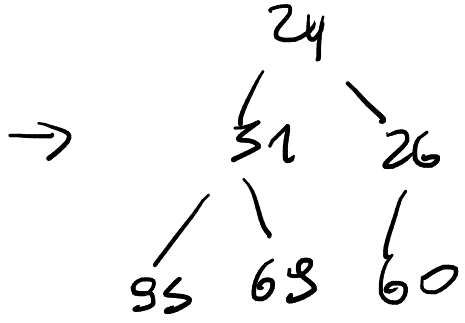



5°



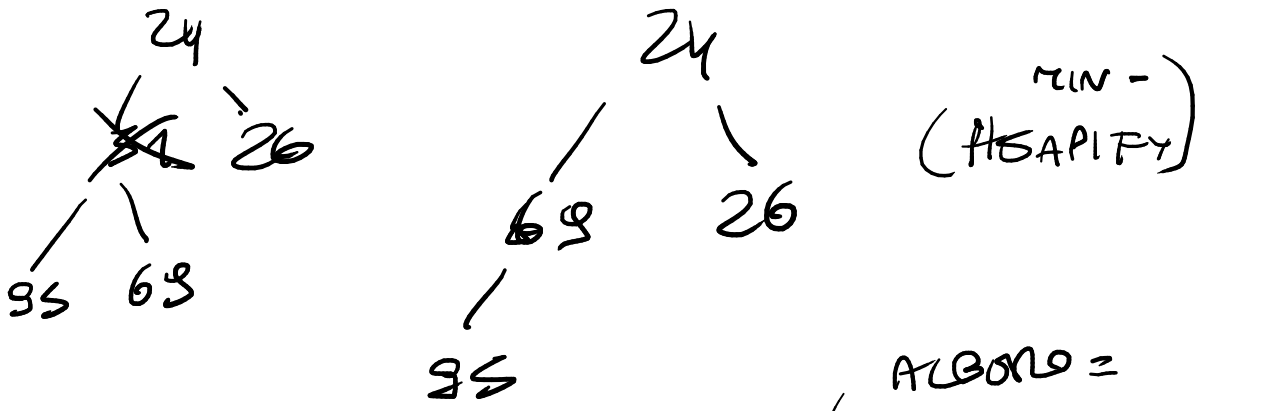

24




... (MIN-HEAPIFY)

DATO UNO HEAP, CANCELLA (S1)




SINTESI → HEAP (B.T. QUASI COMPLETO) ↓ ALBERO = BINARY TREE

---

MT
- CASO 1 → 0
- CASO 2 (Θ) → k
- CASO 3 → ∞

↑

RICORR. → $AT(\frac{n}{b}) \leq k\, f(n)$ ↓

COND. DI REGOLARITÀ

RISOLTO PER k, n

∀k, ∀n

$2T\left(\frac{n}{2}\right) + c \rightarrow$ ALBERO

$O[\log_2(n) + 1] \rightarrow \Theta(\log(n))$

```
REC_FUN(A)
IF A==1
RETURN REC_FUN(A, P, R) + REC_FUN(A, P, Q)
```

n/2

n/2

DIVIDE ET IMPERA (?)

ARRAY ORDINATO CIN 2000 CRESCENTE)

**Esercizio 1** (10 punti) Diciamo che un array senza ripetizioni $A[1, n]$ è *semi-ordinato* se esiste un indice $k$, con $1 \leq k < n$, tale che $A[k + 1..n]A[1..k]$ sia ordinato, ovvero i sottoarray $A[k + 1..n]$ e $A[1..k]$ sono ordinati e $A[n] < A[1]$. In questo caso l'indice $k$ viene detto il centro dell'array. Ad esempio l'array che segue è semi-ordinato con centro $k = 4$.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 4 | 9 | 12 | 18 | -1 | 1 | 2 |

Scrivere una funzione `centre(A)` che dato un array $A$ semi-ordinato ne restituisce il centro. Giustificare la correttezza dell'algoritmo e valutarne la complessità.

DIVIDE - ET - IMPERA

USO - CASO → DIVIDE ET IMPERA

```
CENTRE(A)
RETURN CENTRE_REC(A, 1, N)
```

→ A, P, Q

A, Q, R

DIVIDE ET IMPERA

**Esercizio 1** (10 punti) Realizzare una funzione `Prod(A,k)` che dato un array `A` di interi $\geq 0$ ordinato in senso crescente e un valore intero `k` $\geq 0$ verifica se esistono due indici `i` e `j` tali che `k = A[i]*A[j]`. Valutarne la complessità. Adattare la soluzione al caso in cui i valori nell'array possono essere negativi (assumendo ancora `k` $\geq 0$).

SE VALE UNA SPECIFICA CON SUGLI INDICI

-> WHILE ( I ∈ N, J ∈ N, K <> A[I] * A[J])

IF ( < I++ )  IF ( > ) J++

**Esercizio 2** (10 punti) Si consideri il problema di selezione di attività compatibili, con $n$ attività $a_1, \ldots, a_n$ che ci vengono date attraverso due vettori **s** e **f** di tempi di inizio e fine, e ordinate per tempo di *inizio* (cioè $0 < s_1 \leq s_2 \leq \cdots \leq s_n$).

(a) Scrivere un algoritmo greedy iterativo che implementa la scelta greedy di selezionare l'attività che inizia per ultima.

(b) Determinare l'insieme di attività restituito dall'algoritmo al punto (a) quando eseguito sul seguente insieme di 6 attività, caratterizzate dai seguenti vettori **s** e **f** di tempi di inizio e fine:

$$\mathbf{s} = (1, 2, 3, 5, 7, 10) \qquad \mathbf{f} = (3, 9, 10, 7, 11, 12)$$

(c) Dimostrare la proprietà di scelta greedy, cioè che esiste soluzione ottima che contiene l'attività che inizia per ultima.

ALGORITMI (PSEUDOCODICE)

ALGORITMI GREEDY (= SEL-ATTIVITÀ)

$A_{OPT} = \{A_1\}$

WHILE (i ∈ N)

IF ( $A_i \cup A_{OPT} \neq \emptyset$ )

$A_{OPT} = A_{OPT} \cup A_i$

**Esercizio 2** (9 punti) Data una stringa $X = x_1, x_2, \ldots, x_n$, si consideri la seguente quantità $\ell(i, j)$, definita per $1 \leq i \leq j \leq n$:

$$\ell(i,j) = \begin{cases} 1 & \text{se } i = j \\ 2 & \text{se } i = j-1 \\ 2 + \ell(i+1, j-1) & \text{se } (i < j-1) \text{ e } (x_i = x_j) \\ \sum_{k=i}^{j-1} (\ell(i,k) + \ell(k+1,j)) & \text{se } (i < j-1) \text{ e } (x_i \neq x_j). \end{cases}$$

-> LEGGO E IMPARO LA LOGICA DELLA SOLUZIONE

1. Scrivere una coppia di algoritmi INIT_L($X$) e REC_L($X, i, j$) per il calcolo memoizzato di $\ell(1, n)$.
2. Si determini la complessità *al caso migliore* $T_{\text{best}}(n)$, supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.

MEMOIZZAZIONE → RICORSIVO ⟨ INIT / REC

BOTTOM-UP → ITERATIVO

```
INIT_L(X)
 n <- length(X)
 if n = 1 then return 1
 if n = 2 then return 2
 for i=1 to n-1 do
   L[i,i] <- 1
   L[i,i+1] <- 2
 L[n,n] <- 1
 for i=1 to n-2 do
   for j=i+2 to n do
     L[i,j] <- 0
 return REC_L(X,1,n)
```

→ i = 3, i = j + 1

→

LEGGI L'ALGORITMO ....

```
REC_L(X,i,j)
 if L[i,j] = 0 then
   if x_i = x_j then L[i,j] <- 2 + REC_L(X,i+1,j-1)
   else for k=i to j-1 do
          L[i,j] <- L[i,j] + REC_L(X,i,k) + REC_L(X,k+1,j)
 return L[i,j]
```