**Fondamenti di Informatica II (V.O.) – Ingegneria Informatica**
Compito, 19/3/2007 (Durata: 3h)

**Nome, Cognome, Matricola:** _____

# Prima Parte: domande a risposta unica

Si forniscano negli appositi spazi **solo le risposte** alle seguenti domande:

1. Utilizzando il Master Theorem, si determini l'ordine di grandezza associato alla ricorrenza $T(n) = 27T(n/3) + n^2 \log^2 n$

   $T(n) = \Theta(n^3)$

   (**Solution strategy:**) In the above recurrence, we have $a = 27$ and $b = 3$, hence $\log_b a = 3$. For $\epsilon = 1/2$, we have $n^2 \log^2 n = O(n^{3-\epsilon}) = O(n^2 n^{1/2})$, hence we are in the first case of the Master Theorem. Therefore $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$, and the answer follows.

2. Si consideri la seguente doppia sommatoria:

   $$S(n) = \sum_{i=4}^{n-3} \sum_{j=1}^{n-2-i} 1.$$

   Si valuti $S(12)$.

   $S(12) = 21$

   (**Solution strategy:**) We have:

   $$\begin{aligned} S(n) &= \sum_{i=4}^{n-3} \sum_{j=1}^{n-2-i} 1 \\ &= \sum_{i=4}^{n-3} (n-2-i) \\ &= \sum_{k=1}^{n-6} k \\ &= (n-6)(n-5)/2 \end{aligned}$$

   For $n = 12$, we obtain $S(12) = 6 \cdot 7/2 = 21$.

3. Sia $\boldsymbol{a} = (3, 0, 0, 0, -1, 0, 0, 0)$ e sia $\boldsymbol{b} = \boldsymbol{a} \circledast \boldsymbol{a} \circledast \boldsymbol{a}$. Si calcoli $DFT_8(\boldsymbol{b})$.

   $DFT_8(\boldsymbol{b}) = (8, 64, 8, 64, 8, 64, 8, 64)$

(**Solution strategy:**) By applying the cyclic convolution theorem twice, it follows that $DFT_8(\boldsymbol{b}) = DFT_8(\boldsymbol{a}) \odot DFT_8(\boldsymbol{a}) \odot DFT_8(\boldsymbol{a})$, where $\odot$ denotes componentwise product. Since $\boldsymbol{a}$ is an $(8,4)$-sparse vector, its $DFT_8$ is obtained by concatenating $DFT_2(3,-1) = (2,4)$ four times. Hence

$$\begin{aligned} DFT_8(\boldsymbol{a}) &= (2,4,2,4,2,4,2,4), \text{ and} \\ DFT_8(\boldsymbol{b}) &= (8,64,8,64,8,64,8,64) \end{aligned}$$

4. Si consideri la seguente matrice di pesi $W$ relativa a un grafo diretto con quattro nodi:

$$W = \begin{bmatrix} 0 & 1 & 7 & 6 \\ 4 & 0 & 2 & \infty \\ 1 & \infty & 0 & 4 \\ 5 & \infty & \infty & 0 \end{bmatrix}$$

Se determinini costo$(\pi^{(4)}_{1,3})$ secondo la strategia APSP di Floyd-Warshall.

| costo$(\pi^{(4)}_{1,3}) = 3$ |
|---|

(**Solution strategy:**) Since $n = 4$, ($\pi^{(4)}_{1,3}$ is the shortest path in the graph between nodes 1 and 3. The only two such paths are $\langle 1,3\rangle$ of cost 7, or $\langle 1,2,3\rangle$ of cost 3, and the answer follows.

5. Per $n, m > 0$, una formula $\Phi(x_1, x_2, \ldots, x_n)$ è in forma **3-normale disgiuntiva** (3-DNF) se $\Phi = C_1 \vee C_2 \vee \cdots C_m$, e $C_i = y^i_1 \wedge y^i_2 \wedge y^i_3$, con $y^i_j$ letterale su $\{x_1, x_2, \ldots, x_n\}$. Si indichi se la seguente affermazione è **vera** o **falsa**: "Il problema di determinare se una formula booleana $\Phi(x_1, x_2, \ldots, x_n)$ in forma 3-DNF è soddisfacibile è NP-completo".

| L'affermazione è **falsa**. |
|---|

(**Solution strategy:**) The statement is clearly false. Indeed, each such formula is satisfiable if and only if there exists a clause $C_i$ which does not contain both a literal and its negation. This is easily checkable in linear time.

# Seconda Parte: risoluzione di problemi

**Esercizio 1 [15 punti]**

- **Punto 1 [7 punti]** Sia $n > 0$. Si dimostri rigorosamente che per ogni vettore $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1}) \in \boldsymbol{C}^n$ si ha:

$$(F_n)^2\boldsymbol{x} = n \cdot \boldsymbol{x}^R,$$

  dove $(F_n)^2 = F_n \times F_n$ denota il quadrato della matrice di Fourier di ordine $n$ e $\boldsymbol{x}^R = (x_0, x_{n-1}, x_{n-2} \ldots, x_1)$ denota il *reverse* del vettore $\boldsymbol{x}$.

- **Punto 2. [8 punti]** Utilizzando la relazione provata al Punto 1, si fornisca lo pseudocodice e si analizzi un algoritmo *divide-and-conquer* KFT$(\boldsymbol{x}, k)$ che, dati in ingresso un vettore complesso $\boldsymbol{x} \in \boldsymbol{C}^n$, con $n$ potenza di due e $k$ un generico intero positivo o nullo, restituisca $\boldsymbol{y} = (F_n)^k\boldsymbol{x}$ eseguendo $T(n, k) = O(n(k + \log n))$ operazioni aritmetiche tra scalari complessi. (*Suggerimento:* si osservi che per $k \geq 2$ vale $(F_n)^k\boldsymbol{x} = (F_n)^2((F_n)^{k-2}\boldsymbol{x})$ ...)

**Answer: Point 1** In order to prove the stated relation, let $\boldsymbol{y} = F_n\boldsymbol{x}$. Then we have:

$$(F_n)^2\boldsymbol{x} = n\boldsymbol{x}^R \Leftrightarrow F_n\boldsymbol{y} = n\boldsymbol{x}^R \Leftrightarrow \boldsymbol{x}^R = \frac{1}{n}F_n\boldsymbol{y} \Leftrightarrow \boldsymbol{x} = \frac{1}{n}(F_n\boldsymbol{y})^R \Leftrightarrow \boldsymbol{x} = (F_n)^{-1}\boldsymbol{y},$$

where the (trivially true) last relation follows from the reduction of $DFT_n^{-1}$ to $DFT_n$ discussed in class.

**Point 2** Let $\boldsymbol{x} \in \boldsymbol{C}^n$ and $k \geq 0$. In order to derive a divide-and-conquer algorithm, let us first discuss the base cases. If $k = 0$, we have $(F_n)^0 = I_n$, hence we return $\boldsymbol{x}$. If $k = 1$, we have $(F_n)^1 = F_n$, hence we return $F_n\boldsymbol{x}$ by calling FFT$(\boldsymbol{x})$. When $k \geq 2$, we observe that $(F_n)^k\boldsymbol{x} = (F_n)^2\boldsymbol{z}$, where $\boldsymbol{z} = (F_n)^{k-2}\boldsymbol{x}$. We can then obtain $\boldsymbol{z}$ recursively and then obtain $(F_n)^k\boldsymbol{x}$ by applying the relation proved in Point 1, by first reversing $\boldsymbol{z}$ and then multiplying each of its components by $n$. The pseudocode follows.

KFT$(\boldsymbol{x}, k)$
$n \leftarrow$ length$(\boldsymbol{x})$
**if** $(k = 0)$ **then return** $\boldsymbol{x}$
**if** $(k = 1)$ **then return** FFT$(\boldsymbol{x})$
$\boldsymbol{z} \leftarrow$ KFT$(\boldsymbol{x}, k - 2)$
$y_0 \leftarrow n \cdot z_0$
**for** $i \leftarrow 1$ **to** $n - 1$ **do** $y_i \leftarrow n \cdot z_{n-i}$
**return** $\boldsymbol{y}$

The above algorithm makes use of the routine FFT developed in class, and its correctness follows from the previous discussion. The recurrence relation on the number of arithmetic operations between complex scalars performed by KFT$(\boldsymbol{x}, k)$ is:

$$T(n,k) = \begin{cases} 0 & k = 0, \\ cn \log n & k = 1 \\ T(n, k-2) + n & k > 1, \end{cases}$$

for some constant $c$ (determined by the call to FFT). The above relation unfolds to $T(n,k) = T(n, k-2i) + ni$, for $1 \leq i \leq \lfloor k/2 \rfloor$. For even $k$, we then obtain $T(n,k) = nk/2 + T(n,0) = nk/2$, while for odd $k$ (the worst case) we get $T(n,k) = n\lfloor k/2 \rfloor + T(n,1) = n\lfloor k/2 \rfloor + cn \log n$. Therefore $T(n,k) = O(n(k + \log n))$.

Indeed, this is not the best algorithm for computing $(F_n)^k \boldsymbol{x}$, since it is not difficult to prove (reasoning along similar lines) that the following closed formula holds. Let $\boldsymbol{y} = F_n \boldsymbol{x}$. Then:

$$(F_n)^k \boldsymbol{x} = \begin{cases} n^{k/2} \cdot \boldsymbol{x} & \text{if } k \bmod 4 = 0, \\ n^{(k-1)/2} \cdot \boldsymbol{y} & \text{if } k \bmod 4 = 1, \\ n^{k/2} \cdot \boldsymbol{x}^R & \text{if } k \bmod 4 = 2, \\ n^{(k-1)/2} \cdot \boldsymbol{y}^R & \text{if } k \bmod 4 = 3. \end{cases}$$

Implementing the above formula, in the worst case ($k$ mod 4 an odd number), we would only need to compute $n^{(k-1)/2}$ in $O(\log k)$ time, transform $\boldsymbol{x}$ in $O(n \log n)$ time and perform $n$ additional scalar multiplications, for a total of $O(n \log n + \log k)$ time. □

**Esercizio 2 [15 punti]** Data una stringa $X = \langle x_1, x_2, \ldots, x_n \rangle$, si consideri la seguente ricorrenza $\ell(i,j)$, definita per $1 \leq i \leq j \leq n$:

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

- **Punto 1 [10 punti]** Si fornisca lo pseudocodice della coppia di procedure INIT_L$(X)$ e REC_L$(X,i,j)$ che, data in ingresso una stringa $X$ di lunghezza $n$, restituiscano in uscita $\ell(1,n)$ utilizzando la memoizzazione.

- **Punto 2 [5 punti]** Si analizzi l'albero associato alla chiamata REC_L$(X,1,n)$ e si determini la complessità **al caso migliore** $T_B(n)$, supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.

**Answer:**

**Point 1** Recall that INIT_L($X$) must return directly on base cases ($n = 1$ or $n = 2$ in this case), otherwise it must initialize a look-up table with base and default values and then invoke REC_L($1, n$), which computes recursively all non-base values, storing them in the table to avoid repeated calls. Observe that 0 is a suitable default value, since $\ell(i, j) > 0$ for $1 \leq i \leq j \leq n$. The code follows.

```
INIT_L(X, n)
if n = 1 then return 1
    {ℓ(1,1) = 1, base case}
if n = 2 then return 2
    {ℓ(1,2) = 2, base case}
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
    forj ← i + 2 to n do
        L[i, j] ← 0
return REC_L(X, 1, n)
```

```
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]
```

**Point 2** First, observe that INIT_L($X$) does not perform any character comparisons. Let us now analyze the recursion tree associated to the call REC_L($X, 1, n$). The best case is clearly the one where all characters are the same, since the recursion tree in that case is unary and its internal nodes correspond to the calls with indices $(1, n), (2, n-1), \ldots (k, n-k+1)$, each associated to a cost of 1 (one comparison). There is one such call for every $1 \leq k \leq n - k - 1$ (nonbase cases), whence $1 \leq k \leq (n-1)/2$, for a running time $T_B(n) \leq \lfloor n/2 \rfloor = O(n)$.

For completeness, let us also consider the worst case. If $x_i \neq x_j$ for $1 \leq i < j \leq n$ all subproblems are eventually solved. In this case, thanks to memoization, there is exactly one internal node for each call with indices $(i, j)$, with $1 \leq i < j - 1$, each associated to a cost of 1 (again, one comparison only), for a total time

$$T_W(n) = \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} 1 = \sum_{i=1}^{n-2} n - 1 - i = \sum_{k=1}^{n-2} k = \frac{(n-2)(n-1)}{2} = \Theta(n^2).$$

Finally, note that if we considered additions as operations also requiring unit cost, the running time in the best case would stay linear in $n$, while the running time in the worst case would become cubic in $n$. □