ESERCIZI SULLA TECNICA *Divide et impera*

1. [**GAP**] In un vettore V di n interi si dice *gap* un indice i, $1 \leq i < n$, tale che $V[i] < V[i+1]$.
   - Dato un vettore V di $n \geq 2$ interi tale che $V[1] < V[n]$, provare che V ha almeno un *gap*.
   - Progettare un algoritmo che, dato un vettore V di $n \geq 2$ interi tale che $V[1] < V[n]$, trovi un *gap* in $O(\log n)$ tempo.
2. [**DOUBLE GAP**] In un vettore V di n interi si dice *double-gap* un indice i, $1 \leq i < n$, tale che $V[i+1] - V[i] \geq 2$.
   - Dato un vettore V di $n \geq 2$ interi tale che $V[n] - V[1] \geq n$, provare che V ha almeno un *double-gap*.
   - Progettare un algoritmo che, dato un vettore V di $n \geq 2$ interi tale che $V[n] - V[1] \geq n$, trovi un *double-gap* in $O(\log n)$ tempo.
3. [**PUNTO FISSO**] Progettare un algoritmo che, preso un vettore ordinato V di n interi distinti, determini se esiste un indice i tale che $V[i] = i$ in $O(\log n)$ tempo.
4. [**ZERI DI VETTORI CONTINUI**] Sia V un vettore di n interi. Si dice che V è *continuo* se per ogni $i = 1, 2, \ldots, n-1$, $|V[i+1] - V[i]| \leq 1$. Si dice *zero* del vettore un indice k tale che $V[k] = 0$.
   - Dato un vettore V di $n \geq 2$ interi continuo tale che $V[1] < 0$ e $V[n] > 0$, provare che V ha almeno uno *zero*.
   - Progettare un algoritmo che, dato un vettore V di $n \geq 2$ interi continuo e tale che $V[1] < 0$ e $V[n] > 0$, trovi uno *zero* in $O(\log n)$ tempo.
5. [**MASSIMA SOMMA DI SOTTOVETTORI**] Progettare un algoritmo che, preso un vettore V di n interi, trovi un sottovettore (una sequenza di elementi consecutivi del vettore) la somma dei cui elementi sia massima. La complessità dell'algoritmo deve essere $O(n \log n)$.
6. [**SOTTOVETTORI SOTTILI**] In un vettore V di interi si dice *spessore* del vettore la differenza tra il massimo e il minimo del vettore. Progettare un algoritmo che, preso un vettore V di n interi ed un intero C, trovi un sottovettore (una sequenza di elementi consecutivi del vettore) di, lunghezza massima tra quelli di spessore al più C. La complessità dell'algoritmo deve essere $O(n \log n)$.
7. [**MASSIMO PRODOTTO DI SOTTOVETTORI**] Progettare un algoritmo che, preso un vettore V di n reali, trovi un sottovettore (una sequenza di elementi consecutivi del vettore) il prodotto dei cui elementi sia massimo. La complessità dell'algoritmo deve essere $O(n \log n)$.

8. [**SOTTOVETTORI EVANESCENTI**] Progettare un algoritmo che, preso un vettore V di n reali, trovi un sottovettore (una sequenza di elementi consecutivi del vettore) la somma dei cui elementi sia la più vicina a zero. La complessità dell'algoritmo dovrebbe essere $O(n \log n)$, una soluzione con complessità $O(n \log^2 n)$ è comunque non banale.

9. [**MASSIMA POTENZA DI UNA SOTTOSEQUENZA**] Progettare un algoritmo che, prese in input due sequenze A e B di caratteri trova il massimo intero k tale che B^k è una sottosequenza di A. La sequenza B^k è quella che si ottiene ripetendo k volte ogni caratteri di B, ad esempio se $B = aab$ allora $B^3 = aaaaaabbb$. Una sequenza B è una sottosequenza di una sequenza A se B si può ottenere da A eliminando eventualmente dei caratteri. L'algoritmo deve avere complessità $O(n \log n)$ dove n è la lunghezza di A.

10. [**MINIMO IN VETTORI UNIMODULARI**] Un vettore di interi V è detto *unimodulare* se ha tutti valori distinti ed esiste un indice i tale che $V[1] > V[2] > \cdots V[i-1] > V[i]$ e $V[i] < V[i+1] < V[i+2] < \cdots < V[n]$, dove n è la dimensione del vettore. Progettare un algoritmo che dato un vettore unimodulare restituisce il valore minimo del vettore. La complessità dell'algoritmo deve essere $O(n \log n)$.

11. [**NUMERO DI INVERSIONI**] Progettare un algoritmo che dato un vettore V di n interi calcola il numero di inversioni. Un inversione è una coppia di indici i, j tali che $i < j$ e $V[i] > V[j]$. L'algoritmo deve avere complessità $O(n \log n)$.

12. [**POTENZA MODULARE**] Considera il problema di calcolare l'espressione $(a^n) \bmod r$ assumendo come input tre interi positivi a, n ed r. Progettare un algoritmo che risolve il problema in $O(\log n)$ tempo.

13. [**RADICE**] Progettare un algoritmo che, dato un intero n, calcoli il valore $\lfloor \sqrt{n} \rfloor$ in $O(\log n)$ tempo, usando solo operazioni aritmetiche.

14. [**ROTAZIONE**] Si supponga di avere in input un vettore ordinato di n interi il cui contenuto è stato ruotato di k posizioni.
    - Supponendo di conoscere solo n, progettare un algoritmo che restituisca l'elemento massimo del vettore in $O(\log n)$ tempo.
    - Supponendo di conoscere n e k, progettare un algoritmo che restituisca l'elemento massimo del vettore in $O(1)$ tempo.

15. [**SCOMPOSIZIONE ADDITTIVA**] Siano X e Y due vettori di n interi, ordinati in senso non decrescente ed x un intero. Progettare un algoritmo che trovi, se esiste, una coppia di indici i, j tali che $X[i] + Y[j] = x$. L'algoritmo deve avere complessità $O(n \log n)$.

16. [**MEDIANO DI DUE VETTORI**] Siano X e Y due vettori di n interi, ordinati in senso non decrescente. Progettare un algoritmo che trovi il mediano dei $2n$ elementi in $O(\log n)$ tempo.

17. [**MEDIANO DI TRE VETTORI**] Siano X, Y e Z tre vettori di n interi, ordinati in senso non decrescente. Progettare un algoritmo che trovi il mediano dei $3n$ elementi in $O(\log n)$ tempo.

18. [**RICERCA TERNARIA**] Si consideri un algoritmo di ricerca "ternaria", che prima confronta l'elemento x da trovare con quello in posizione $\frac{n}{3}$ e poi, eventualmente con quello in posizione $\frac{2n}{3}$; in questo modo, o si trova x o si riduce lo spazio di ricerca ad $\frac{1}{3}$ rispetto all'originale. Valutare la complessità dell'algoritmo e confrontare l'algoritmo con la ricerca binaria.

19. [**PRODOTTO DI POLINOMI**] Progettare un algoritmo che presi in input due polinomi dello stesso grado

$$A = \sum_{i=0}^{n-1} a_i x^i \quad \text{e} \quad B = \sum_{i=0}^{n-1} b_i x^i$$

calcola i coefficienti del polinomio prodotto C. Chiaramente,

$$C = \sum_{k=0}^{2n-2} \left(\sum_{i+j=k} a_i b_j \right) x^k.$$

Però l'algoritmo deve avere complessità strettamente inferiore ad $O(n^2)$. Per semplicità si può assumere che n sia una potenza di 2.

20. [**BULLONI E DADI**] Abbiamo una collezione di n bulloni di differenti dimensioni ed una collezione di n dadi. Sappiamo che ad ogni bullone corrisponde un dado.

    - Progettare un algoritmo che assegni ad ogni bullone il dado corrispondente. L'algoritmo deve avere tempo di esecuzione $O(n \log n)$.
    - Assumendo che non è possibile confrontare tra loro i bulloni ne confrontare tra loro i dadi, progettare un algoritmo che assegni ad ogni bullone il dado corrispondente. L'algoritmo deve avere tempo di esecuzione $O(n \log n)$ nel caso medio.

21. [**FIBONACCI**] I numeri di Fibonacci determinano una famosa sequenza

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

che si definisce tramite la ricorrenza

$$F(n) = F(n-1) + F(n-2) \quad \text{For } n > 1$$

con le due condizioni iniziali

$$F(0) = 0, \quad F(1) = 1.$$

   - Prova la seguente equivalenza

$$\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \quad \text{for } n \geq 1.$$

   - Progettare un algoritmo che calcola l'n-esimo numero di Fibonacci in $O(\log n)$ tempo.