* Dato un grafo G la chiusura transitiva di G contiene un arco fra due nodi i,j se e solo se in G vi è un arco fra i nodi i, j  V
* Dijkstra ha complessità THETA(m log n)V.
* Esiste un algoritmo avente complessità spaziale θ(n) e complessità temporale θ(log n)  F
* Il probela della ricerca di un elemento in un array non ordinato ha complessita intrinseca Ω(n)V
* Il problema della ricerca del minimo elemento in un array ordinato ha complessità intrinseca omega(n) . dove n è il numero di elementi nell’array.  F
* In un albero esiste sempre un cammino tra due nodi che non passa dalla radice  F
* In un grafo non orientato ed aciclico il numero di nodi è uguale al numero di archi meno uno (n= m-1)  V
* In un grafo orientato aciclico tutti i nodi hanno grado di entrata maggiore di zero  F
* In un grafo orientato con n nodi il numero di componenti connesse massimali è al più n  F
* In un grafo orientato non contenente cicli, esiste almeno un nodo con grado di entrata uguale a 0.V
* In un grafo pesato non orientato e connesso esiste sempre almeno due alberi ricoprenti aventi lo stesso costo  F
* In un grafo pesato non orientato esiste sempre un unico minimo albero ricoprente  F
* L’algoritmo di Floyd calcola, dati un grafo orientato e pesato e un nodo x di tale grafo, le distanze minime fra x e i nodi da esso raggiungibili  V
* L’algoritmo di Floyd ha complessità spaziale θ(n^2)  V
* L’algoritmo di Floyd ha complessità temporale θ/O(n^3)  V
* L’algoritmo di Floyd ha complessità θ(n^3) dove con n si intende il numero di archi presenti nel grafo  V la spaziale è θ(n^2)
* L’algoritmo di kruscal ha complessità, m numero di archi e n nodi, O(m log n) V
* L**’**algoritmo di Kruskal ha complessità θ(n^4) nel caso peggiore F
* L’algoritmo di Prim ha complessità θ(m) con m numero di archi F è O(m\*n) uguale a Ford
* L’algoritmo di Prim permette di calcolare il peso dei cammini minimi fra ogni coppia di nodi di un grafo orientato e pesato  F
* L’algoritmo di visita in ampiezza di un grafo con n nodi ed m archi è θ(m) nel caso di rappresentazione con liste di adiacenza  V
* L’algoritmo MergeSort ha complessità temporale O(n^3) nel caso peggiore  F
* L’algoritmo QuickSort ha complessità temporale θ(n log n) nel caso peggiore  F
* L’algoritmo QuickSort ha complessità temporale θ(n^2) nel caso peggiore  V
* L’inserimento di un elemento in un heap binario ha complessità temporale temporale theta(log n) nel caso peggiore (dove n è il numero dei nodi)  V
* L’inserimento di un elemento in un heap binario ha complessità temporale omega(log n) nel caso peggiore, dove n è il numero dei nodi )  V
* L’inserimento di un elemento in un Heap ha complessità θ( lg n) nel caso peggiore dove n è il numero dei nodi  V
* L’inserimento di un elemento in un Heap ha complessità θ(1) nel caso migliore  V
* L’inserimento di un elemento in una Hash table ha complessità O(n) nel caso miglioreF
* L’inserimento di un valore in una tabella hash contenente n valori ha complessità temporale theta(log n) nel caso peggiore  F
* La cancellazione d un elemento da un Heap ha complessità θ(lg n^2) nel caso peggiore  F
* La complessità dell’algoritmo di Floyd è O(n^5) , dove n è il numero di nodi del grafo in input  F
* La complessità dell’inserimento di un elemento in una tabella Hash in cui sono presenti n elementi è nel caso peggiore θ(n) V
* La complessità dell’inserimento di un elemento in una tebella hash in cui sono presenti n elementi è nel caso peggiore O(n) )  F
* La complessità di accedere all’elemento con priorità massima in una coda di priorità realizzata tramite heap è theta(log n), essendo n il numero di elementi della coda.  F
* La complessità intrinseca del problema di calcolare il valore massimo di un array di interi è Ω(n) dove n è il numero di interi presenti nell’array  V
* La complessità intrinseca del problema di calcolare la somma di un array di interi è Ω(n) dove n è il numero di interi presenti nell’array  V
* La complessità spaziale della visita anticipata di un albero binario con n nodi è O(log n) nel caso peggiore F
* La complessità spaziale della visita infissa di un albero è Ω(n^2), dove n è il numero di nodi presenti nell’albero  F
* La complessità spaziale della visita per livelli di un albero è Ω(n) dove n è il numero di nodi presenti nell’albero  V
* La complessità temporale del problema dell’inserimento in un array ordinato di interi è Ω(log n)  V
* La complessità temporale del problema di decidere se un intero è presente in una lista ordinata di interi è Ω(n) V
* La complessità temporale dell’algoritmo di Dijkstra è θ(n^2) dove n è il numero di nodi presenti nel grafo nel caso di rappresentazione con liste di adiacenza  V
* La funzione F(n) = 2n log n^2 è O(n log n)  V
* La funzione f(n) = 2n^2 è Ω(n)V
* La funzione F(n) = n + 2^n è O(n)  F
* La funzione f(n) = n è Ω(log (n^2))  V
* La funzione F(n) = n log n è O(n^2)  V
* La funzione f(n)= n^2 è Omega(n log(n^n))  F
* La funzione f(n)=2n log n è omega(n log log n) )  V
* La funzione f(n)=n^2 è omega(n log n)  V
* La ricerca di un elemento in un albero AVL ha complessità O(log n) dove n sono i nodi dell’albero  V
* La ricerca di un elemento in un array ordinato ha complessità intrinseca Ω(log n) oppure Ω(n^3)  F
* La ricerca in un array di interi è omega(n) )  V
* La somma di interi su un array ha complessità omega(n) )  V
* La visita per livelli ha complessità O(1) )  V
* Nel caso peggiore, ricercare/rimuovere/inserire un elemento in albero AVL con n nodi ha complessità O(log n)  V
* Per ogni coppia di nodi u,v appartenenti ad un grafo orientato debolmente connesso esiste sempre un cammino dal nodo u al nodo v e dal nodo v al nodo u  F
* Può esistere un algoritmo che abbia complessità spaziale theta(n^2) e complessità temporale theta(n) dove n indica la dimensione dell’input.  F
* Se un algoritmo ha complessità θ(2^2n) allora la sua complessità è O(2^n)  V
* Se un algoritmo ha complessità θ(2^n) allora la sua complessità non è Ω(3^n)  V
* Se un problema P ha complessità intrinseca Ω(2^n) allora esiste almeno un algoritmo risolutore di P che ha complessità O(2^n)  V
* Se un problema P ha complessità intrinseca Ω(n^3) allora esiste almeno un algoritmo risolutore di P che ha complessità O(n)  F
* Se una funzione f è O(n) allora è anche Ω(n^2)F
* Sia dato un algoritmo A risolutore del problema P, se la complessità di A è O(f(n)) e la complessità di A è anche Ω(f(n)) allora A è ottimale per P  V
* Sia dato un algoritmo A risolutore del problema P. Se la complessità di A è omega(f(n)) allora la complessità intrinseca di P è omega(f(n)) V
* Sia G un grafo connesso, orientato ed aciclico i cui pesi sono numeri positivi. L’algoritmo di Floyd potrebbe non essere in grado di calcolare le distanze minime tra tutte le coppie di nodi di G F
* Sia G un grafo non orientato che contiene almeno un ciclo. Il numero di componenti connesse massimali è minore o uguale a n-2, dove n è il numero di nodi di G  F
* Sia G un grafo non orientato che non contiene cicli. Il numero di componenti connesse massimali è minore o uguale a n-m, dove n è il numero di nodi e m il numero di archi in G  V
* Sia G un grafo non orientato e aciclico. Il grafo G è un albero. F, DEVE ESSERE ANCHE CONNESSO
* Sia *G* un grafo orientato. Se *G* contiene cicli allora non vi è nessun nodo in *G* con grado di entrata 0.F
* Tutte le visite hanno complessità O(n).  V
* Un albero binario è bilanciato se la differenza tra l’altezza del sottoalbero sinistro della radice e l’altezza del sottoalbero destro della radice è minore o uguale a 1  V
* Un albero binario è detto di ricerca se, per ognuno dei suoi nodi u, la radice del figlio sinistro di u contiene un valore minore di quello contenuto in u e la radice del figlio destro di u contiene un valore maggiore o uguale di quello contenuto in u.V
* Un grafo connesso contenente un ciclo di *m* archi ammette almeno *m* alberi ricoprenti. V
* Un grafo connesso ed orientato in cui esiste almeno un nodo con grado di entrata uguale a 0 può contenere un ciclo.V
* Un grafo G non orientato e aciclico ha come numero di componenti connesse ( non considerando gli insiemi massimali) <= a n )  V
* Un grafo non orientato connesso e pesato (sugli archi) ammette sempre un unico albero ricoprenti di costo minimo  V
* Un grafo non orientato è connesso se e solo se ogni nodo è raggiungibile da ogni altro tramite un cammino  V
* Un grafo non orientato è connesso se e solo se vi è un cammino senza cicli che passa per tutti i nodi  F
* Un grafo non orientato è connesso se e solo se vi è un cammino senza cicli che passa per tutti i nodi  F
* Un grafo non orientato è fortemente connesso se e solo se per ogni nodo (u,v) si verifica: u è raggiungibile da v e v è raggiungibile da u )  V
* In un grafo non orientato connesso con n nodi ed n-1 archi, il numero di componenti connesse (anche non massimali) è uguale ad n  V secondo MaxL
* Un albero binario a è completo se e solo se per tutti i nodi x di a, la differenza tra l’altezza del sott’albero sinistro di x e l’altezza tra il sottalbero destro di x è minore uguale a 1  F
* La complessità intrinseca del problema della ricerca del secondo minimo in una sequenza disordinata è omega(log n)  F