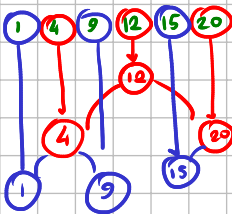


Esercizio 1 (Per fare l'albero ci vuole...). Sia *BinSearchTree* una classe descrivente un nodo di un albero binario di ricerca (ABR), dotata di un campo *key* e due puntatori *sx*, *dx* al figlio sinistro e destro del nodo.

Progettare un algoritmo di complessità $O(n)$ che dato in input un array *A* di *n* interi distinti ordinati in modo crescente restituisca un ABR di altezza minima contenente gli interi di *A*.

```
ric(A:array) {
    n=A.length()
    if(n==1){ return A[0] }
    c=⌈n/2⌋
    u=crea_nodo(A[c-1])
    u.sx=ric(A[0:c-1])
    u.dx=ric(A[c:n])
    return u
}
```



Esercizio 2 (MST senza arco specifico). Sia *G* un grafo non diretto, pesato e connesso. Progettare un algoritmo che dato in input il grafo *G* e un arco $(x,y) \in E(G)$ restituisca *True* se esiste un albero $T \subseteq G$ tale che *T* sia un MST di *G* e che $(x,y) \notin E(T)$ oppure *False* se tale *T* non esiste.

```
ES2 ( G:grafo, (x,y):arco) {
    G'=G \ (x,y)
    T=Kruskal(G)
    T'=Kruskal(G')
    if(ω(T)==ω(T')){ return True }
    return False
}
```

Esercizio 3 (5-copertura di costo minimo). Data una sequenza *X* di *n* interi positivi, definiamo come 5-copertura di *X* una sotto-sequenza *A* di suoi elementi se, presi 5 elementi consecutivi qualsiasi della sequenza di *X*, almeno uno di questi è in *A*.

Dare lo pseudocodice di un algoritmo di complessità $O(n)$ che data una sequenza *X* in input restituisca una 5-copertura di costo minimo e il suo costo. Ad esempio, per *X* = [2, 4, 1, 2, 6, 4, 8, 3, 5, 1] una 5-copertura di costo minimo è [→, →, 1, →, →, →, 3, →, →], avente costo pari a 4

$T[i]$ = 5-cop minimo con $X[i]$ incluso

$T[i] = X[i] + \min(\bigcup_{j=i-1}^{i-5} T[j])$

```
5Cop(X:array) {
    n=X.length()
    T[n]:array
    for(i=0,...,4) {
        T[i]=X[i]
    }
    for(i=5,...,n-1) {
        m=∞
        for(j=i-1,...,i-5) { m=min(m, T[j]) }
        T[i]=X[i]+m
    }
    m=∞
    for(i=n-1,...,n-5) {
        m=min(m, T[i])
    }
    return m
}
```

```
5Cop_Array(X:array, T:array) {
    Sol[n]={_,_,_,...,_}
    ind_m=n
    while(true) {
        for(i=ind_m-1 ..., ind_m-5) {
            if(T[i]< T[ind_m]) {
                ind_m=i
            }
        }
        if(i==0) {
            Sol[ind_m]=X[ind_m]
            return Sol
        }
    }
    Sol[ind_m]=X[ind_m]
}
```