ESERCIZIO 1

Si può utilizzare il Counting Sort, opportunamente modificato.

```
Sort(S)
B = nuovo Array di dim n;
C = nuovo Array di dim 50;
   // 50 = 382 - 333 + 1, numero di possibili
      valori diversi da ordinare.
for (i=0; i<50; i++) C[i]=0;
for (j=0; j<n; j++) {
      C[S[j]-333] ++;
}
for (i=1; i<50; i++) {
      C[i] = C[i] + C[i-1];
}
for (j=n-1; j>0; j--) {
      B[C[S[j]-333]] = S[j];
      C[S[j]-333] --;
}
return B;
```

$T(n) = O(n)$

$S(n) = O(1)$ (si utilizza un array di dim 50)

ESERCIZIO 2

1) La soluzione del problema si può rappresentare con un array binario B tale che

$$B[i] = 0 \iff i \in S_1$$
$$B[i] = 1 \iff i \in S_2,$$

Con $0 \le i < n$;

2)

```
Verifica (G, B, k)
ctr=0;
for (u=0; u<n; u++) {
  for (x = Adj[u].inizio; x≠null; x=x.next) {
      v = x.dato;
      if (B[u] ≠ B[v]) ctr++;
      if (ctr/2 > k) return <FALSE, ctr/2>;
  }
}
return <TRUE, CTR/2>;
```

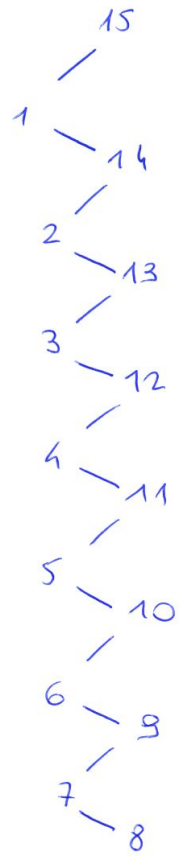
L'algoritmo scorre le liste di adiacenza di tutti i vertici del grafo, dunque la sua complessità è lineare nella dimensione dell'istanza di input:

$$T(V,E) = O(|V| + |E|).$$

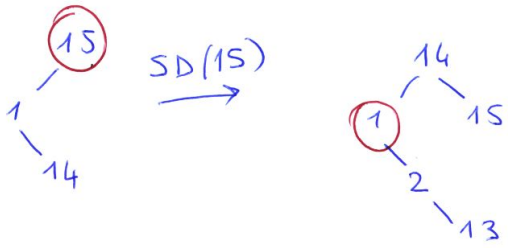
Questo dimostra che MAX CUT $\in$ NP

ESERCIZIO 3

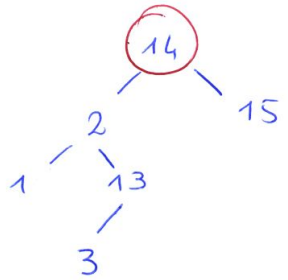
1)



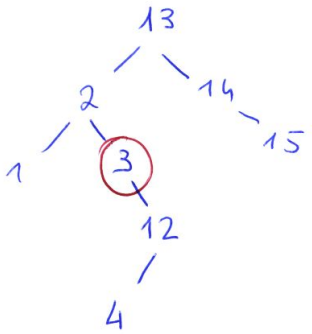
2)



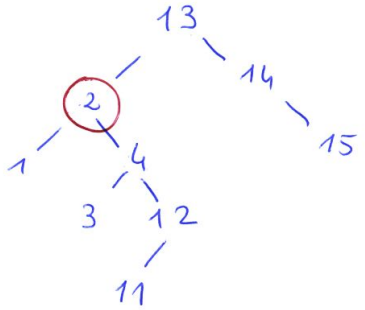
DD(1)



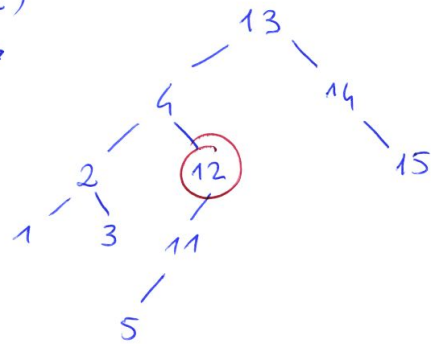
SD(14)



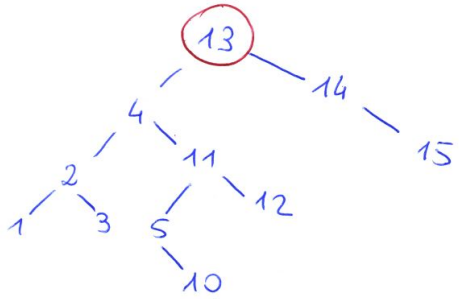
DS(3)



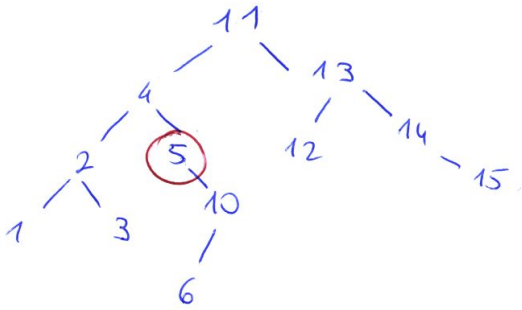
DD(2)



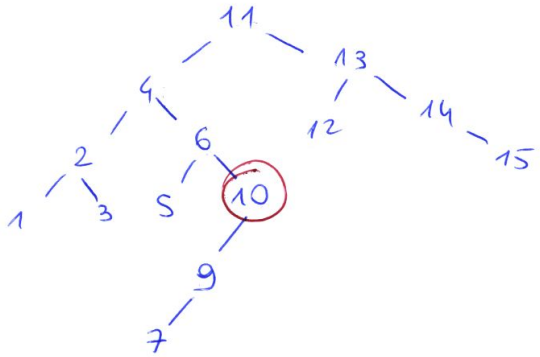
SS(12)



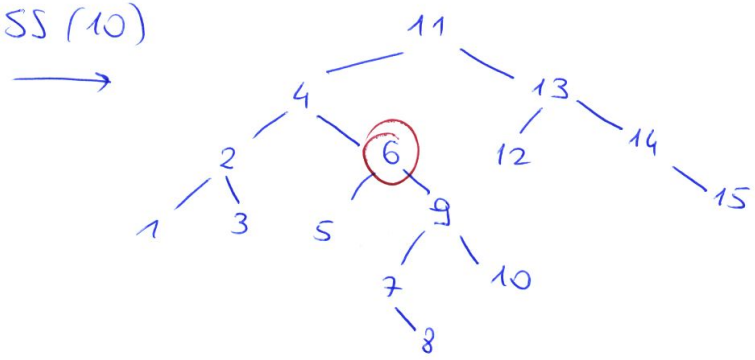
SD(13)



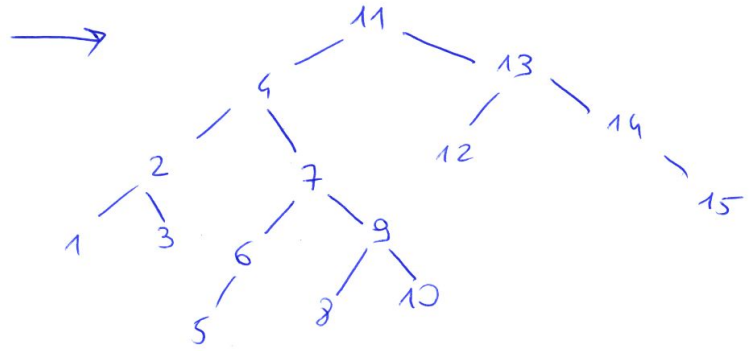
DS(5)



SS (10)



DS(6)



3) ABR2AVL(u)

```
A = nuovoArray();
A = VisitaSimmetrica(u);
return Array2AVL(A, 0, n-1);
```

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

Array2AVL(A, sx, dx)

```
if (sx > dx) return null;
cx = (sx + dx) / 2;
u = nuovoNodo();
u.chiave = A(cx);
u.sx = Array2AVL(A, sx, cx-1);
u.dx = Array2AVL(A, cx+1, dx);
return u;
```

Dato che l'algoritmo Array2AVL è eseguito su un array ordinato, l'albero risultante è bilanciato

$T(n) = 2T\left(\frac{n}{2}\right) + O(1) = O(n)$

ESERCIZIO 4

```
SoluzioneOttima (G, k)
min = |E| ;        // globale
B = nuovo Array() // globale
GeneraBinario (A, n, k);
return B;

GeneraBinario (A, b, k)
if (b==0) {
    <sol, c> = Verifica (G, A, k);
    if (sol == true) {
        if (c < min) {
            min = c;
            Copia A in B;
        }
    }
}
else {
    A[b-1] = 0;
    GeneraBinario (A, b-1, k);
    A[b-1] = 1;
    GeneraBinario (A, b-1, k);
}
```