ESERCIZIO 1

1) Chiavi Distinte (S)

```
D = nuovo Dizionario;
ctr = 0;
for (i = 0; i < n; i++) {
    if (! Appartiene (D, S(i))) {
        Inserisci (D, S(i));
        ctr++;
    }
}
return ctr;
```

2) D contiene solo gli r elementi distinti.

Se D è implementato con un array ordinato, ~~[illegible]~~ ogni verifica di appartenenza costa $O(\log r)$ [→ Ricerca Binaria] e ogni inserimento costa $O(r)$ [→ per mantenere l'array ordinato]

Dato che le verifiche di appartenenza sono n, e gli inserimenti r, il costo complessivo è $O(n \log r + r^2)$

ESERCIZIO 2

```
RicercaRandom ( a, sx, dx, k)
if (sx > dx) return -1;
if (sx == dx) {
    if (a(sx) == k) return sx;
    else return -1;
}
cx = Random (sx, dx);
if (a(cx) == k) return cx;
if (a(cx) > k)
    return RicercaRandom (a, sx, cx-1, k)
else return RicercaRandom (a, cx+1, dx, k)
```

Analisi

CASO PESSIMO:

$$T(n) = T(n-1) + O(1) = O(n)$$

CASO MEDIO: (analisi quicksort/quickselect)

$$T(n) \leq T\left(\frac{3}{4}n\right) + O(1)$$

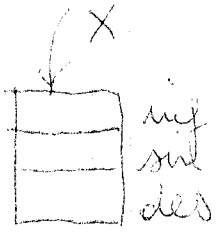
(th. principale 2° caso)

$a = 1 \quad b = 4/3 \quad e = 0$

$a = 1 = b^e = 1 \quad T(n) = O(n^0 \log n) = O(\log n)$

Es. 3 (G, s) pre: G è un grafo di radice s, Raggiunto array di n puntatori;

```
for (u=0; u<n; u++)
  Raggiunto[u] = Null;
Q.enqueue(s);
while (!Q.empty()) {
    u = Q.dequeue();
    if (Raggiunto[u] == NULL) {
        x = nuovonodo();
        Raggiunto[u] = x;
        x.inf = u.dato; x.des = NULL; x.sin = NULL;
    } else x = Raggiunto[u];
    v = listaAdj[u].inizio; primo = true;
    if (v != NULL) {
        if (Raggiunto[v] == NULL) {
            y = nuovonodo();
            Raggiunto[v] = y;
            Q.Enqueue(v);
            y.inf = v.dato; y.sin = y.des = NULL;
            if primo {
                x.sin = y;
                primo = false;
            } else x.des = y;
        }
        x = y; v = v.next;
    }
}
```



Lo schema è di una BFS. Un nodo raggiunto è già stato creato nell'albero e ci si ricorda il puntatore.

ESERCIZIO 4

Certificato: $F' \subseteq F$,
composto da al più k
sottoinsiemi di S.

Algoritmo di Verifica (S, F')

// si ripropone che ogni insieme di F'
sia rappresentato con una lista

* Unione: unione delle liste che
rappresentano gli insiemi di F'
(si può fare concatenando le liste)

* for (i = 0; i < n; i++) {
  if (~~Unione contiene~~ ! Appartiene (Unione, S(i)))
    return false;
  }

* return true;