ASD (FI2 2013-14 12 CFU), ASD (5 CFU), ASD Nettuno (3, 6 CFU)
Appello del 12-6-2015 – a.a. 2014-15 – Tempo a disposizione: *120 minuti*

Problema 1

[Punti: (a) 4/30; (b) 4/30]
Si considerino i metodi Java di seguito illustrati.

```
static void merge3(int a[], int h, int i, int j, int k) {
  int[] temp = new int[k-h+1];
  final int NULL = -1, UNO = 0, DUE = 1, TRE = 2;
  int indice[] = new int[3];
  indice[UNO] = h; indice[DUE] = i; indice[TRE] = j;
  int p = 0;
  while((indice[UNO]<i)||(indice[DUE]<j)||(indice[TRE]<=k)) {
    int min = NULL; if(indice[UNO] < i) min=UNO;
    if(indice[DUE] < j)
      if((min==NULL)||(a[indice[DUE]]<a[indice[min]])) min=DUE;
    if(indice[TRE]<=k)
      if((min==NULL)||(a[indice[TRE]]<a[indice[min]])) min=TRE;
        temp[p] = a[indice[min]];
    p++; (indice[min])++;
  }
  while(p > 0) a[k--] = temp[--p];
}
```

```
static void mergeSort3(int a[], int i, int j) {
  if(i >= j) return;
  int m1, m2;
  if(j - i >= 2) {
    int trz = (j - i + 1) / 3;
    m1 = i+trz; m2 = i+2*trz;
  } else { // nel caso trz == 0
    m1 = i; m2 = j;
  }
  mergeSort3(a, i, m1-1);
  mergeSort3(a, m1, m2-1);
  mergeSort3(a, m2, j);
  merge3(a, i, m1, m2, j);
}
```

Sviluppare, *argomentando adeguatamente* (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

(a) Determinare l'upper bound dell'algoritmo `mergeSort3`, in funzione del numero di celle n dell'array in input (assumere che la prima chiamata sia `mergeSort3(a, 0,` $n-1$), per un qualche array `a` di n celle).

(b) Confrontare l'efficienza temporale e spaziale di `mergeSort3` con quella del *MergeSort* tradizionale.

Problema 2

[Punti: 8/30]
Si consideri un albero binario i cui nodi sono etichettati attraverso un `char`. Scrivere un metodo Java che stampi a schermo le etichette del ramo (percorso radice-foglia) più lungo (a parità di lunghezza, quello più a sinistra) e quelle del ramo più corto (a parità di lunghezza, quello più a destra). L'algoritmo deve avere costo computazionale $\Theta(n)$. Che tipo di visita è stata eseguita?

Problema 3

[Punti: 8/30]
Descrivere una struttura dati efficiente per la gestione di insiemi disgiunti, che supporti le operazioni di creazione di un singleton (cioè un insieme contenente un solo elemento), unione di due insiemi e individuazione dell'insieme a cui appartiene un dato elemento. Presentarne i relativi algoritmi in pseudo-codice e discuterne la complessità computazionale.

Problema 4

[Punti: 9/30]
Alice è un'utente di Twitter e intende postare un tweet. In cuor suo Alice desidera che l'utente Bob – che non è follower di Alice – legga questo tweet. Piuttosto che menzionare direttamente Bob nel suo tweet, Alice spera che si verifichino alcuni retweet, fino ad arrivare al retweet di un qualche utente effettivamente seguito da Bob. Alice si domanda quale sia il minimo numero di retweet affinché ciò possa accadere.

Si richiede di supportare Alice, svolgendo i seguenti punti, nell'ipotesi semplificativa che per ogni utente Twitter sia pubblico l'insieme di follower e di following.

(a) Riformulare il problema come problema su grafi. A tal fine occorre dapprima definire il grafo a cui si farà riferimento, chiarendo come esso sia collegato al problema in esame; successivamente va definito il quesito che si pone Alice come quesito sul grafo definito, utilizzando una terminologia appropriata.

(b) Definire un algoritmo (pseudo-codice) per la soluzione generale del problema, di cui Alice e Bob costituiscono l'input (oltre alla rete sociale Twitter, naturalmente) e r (minimo numero di retweet) costituisce l'output.

(c) Con riferimento a una specifica modalità di rappresentazione del grafo, determinare i costi computazionali dell'algoritmo (tempo e spazio). La modalità di rappresentazione scelta sarà oggetto di valutazione.

ASD (FI2 2014-15 12 CFU) – solo ultimo ordinamneto
Appello del 12-6-2015 – a.a. 2014-15 – Tempo a disposizione: *120 minuti*

Problema 1

[Punti: (a) 4/30; (b) 4/30]
Si considerino le funzioni C di seguito illustrate.

```
void merge3(int a[], int h, int i, int j, int k) {
  int *temp = malloc((k-h+1)*sizeof(int));
  int hh = h, ii = i, jj = j, p = 0;
  while((hh < i) || (ii < j) || (jj <= k)) {
    int *min = NULL;
    if(hh < i) min = &hh;
    if(ii < j)
      if((min == NULL) || (a[ii] < a[*min])) min = &ii;
    if(jj <= k)
      if((min == NULL) || (a[jj] < a[*min])) min = &jj;
    temp[p] = a[*min];
    p++; (*min)++;
  }
  while(p > 0) a[k--] = temp[--p];
  free(temp);
}
```

```
void mergeSort3(int a[], int i, int j) {
  if(i >= j) return;
  int m1, m2;
  if(j - i >= 2) {
    int trz = (j - i + 1) / 3;
    m1 = i+trz;
    m2 = i+2*trz;
  } else { /* nel caso trz == 0 */
    m1 = i;
    m2 = j;
  }
  mergeSort3(a, i, m1-1);
  mergeSort3(a, m1, m2-1);
  mergeSort3(a, m2, j);
  merge3(a, i, m1, m2, j);
}
```

Sviluppare, *argomentando adeguatamente* (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

(a) Determinare l'upper bound dell'algoritmo `mergeSort3`, in funzione del numero di celle n dell'array in input (assumere che la prima chiamata sia `mergeSort3(a, 0,` $n-1$), per un qualche array `a` di n celle).

(b) Confrontare l'efficienza temporale e spaziale di `mergeSort3` con quella del *MergeSort* tradizionale.

Problema 2

[Punti: 8/30]
Si consideri un albero binario i cui nodi sono etichettati attraverso un `char`. Scrivere una funzione C che stampi a schermo le etichette del ramo (percorso radice-foglia) più lungo (a parità di lunghezza, quello più a sinistra) e quelle del ramo più corto (a parità di lunghezza, quello più a destra). L'algoritmo deve avere costo computazionale $\Theta(n)$. Che tipo di visita è stata eseguita?

Problema 3

[Punti: 8/30]
Descrivere una struttura dati efficiente per la gestione di insiemi disgiunti, che supporti le operazioni di creazione di un singleton (cioè un insieme contenente un solo elemento), unione di due insiemi e individuazione dell'insieme a cui appartiene un dato elemento. Presentarne i relativi algoritmi in pseudo-codice e discuterne la complessità computazionale.

Problema 4

[Punti: 9/30]
Alice è un'utente di Twitter e intende postare un tweet. In cuor suo Alice desidera che l'utente Bob – che non è follower di Alice – legga questo tweet. Piuttosto che menzionare direttamente Bob nel suo tweet, Alice spera che si verifichino alcuni retweet, fino ad arrivare al retweet di un qualche utente effettivamente seguito da Bob. Alice si domanda quale sia il minimo numero di retweet affinché ciò possa accadere.

Si richiede di supportare Alice, svolgendo i seguenti punti, nell'ipotesi semplificativa che per ogni utente Twitter sia pubblico l'insieme di follower e di following.

(a) Riformulare il problema come problema su grafi. A tal fine occorre dapprima definire il grafo a cui si farà riferimento, chiarendo come esso sia collegato al problema in esame; successivamente va definito il quesito che si pone Alice come quesito sul grafo definito, utilizzando una terminologia appropriata.

(b) Definire un algoritmo (pseudo-codice) per la soluzione generale del problema, di cui Alice e Bob costituiscono l'input (oltre alla rete sociale Twitter, naturalmente) e r (minimo numero di retweet) costituisce l'output.

(c) Con riferimento a una specifica modalità di rappresentazione del grafo, determinare i costi computazionali dell'algoritmo (tempo e spazio). La modalità di rappresentazione scelta sarà oggetto di valutazione.