Esame di Fondamenti di Informatica T-1/T-A
Ingegneria Gestionale (A-K)

Appello del 18/10/2013

Compito A

NOTA: Per il superamento dell'esame è **necessario** ottenere la sufficienza nello svolgimento dell'Esercizio 1.

Esercizio 1 (4 punti)

1. Si illustri il concetto di ricorsione *tail*, indicando quali siano le differenze nell'utilizzo della memoria rispetto ad una normale ricorsione lineare.
2. Definire il concetto di complessità asintotica di un algoritmo.

Esercizio 2 (2 punti)

Convertire in binario i numeri **85** e **–113**, supponendo di utilizzare una rappresentazione a 8 bit in complemento a 2. Si esegua infine la somma dei due numeri, riconvertendo il risultato in base 10, motivando eventuali differenze tra il risultato ottenuto e quello atteso.

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

```
public static int f(int V[], int N) {
  int i=0, sum=0;
  while(i++<N)
    sum+=V[++i];
  return sum;
}

public static int g(int V[], int N) {
  int j=0, sum=0;
  for (; j<N; ++j)
    sum+=f(V, ++j);
  return sum;
}
```

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `N` (si distinguano i casi in cui `N` assume valori pari da quelli in cui assume valori dispari).
2. Calcolare la complessità in passi base del metodo `g` nei termini del parametro `N` (si supponga `N` pari).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 4 (5 punti)

La Litorale Crociere è una società che organizza crociere nel Mar Mediterraneo. Di recente, ha deciso di informatizzare la gestione del proprio staff, composto da personale altamente professionale, utile a rendere possibile la navigazione. A tal scopo, per ogni membro dello staff vengono memorizzati nome e cognome, ruolo (es. "pilota", "nostromo", "mozzo"), stipendio mensile e numero di matricola. Si scriva una classe `Staff` per la Litorale Crociere che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presenti il metodo `toString` che fornisca una descrizione del membro dello staff.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Staff` (la verifica va fatta sul numero di matricola).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Staff` passato come parametro (in ordine alfabetico per cognome e nome).

Esercizio 5 (8 punti)

Si scriva una classe `Nave` che memorizzi le informazioni relative al personale in viaggio su una determinata nave da crociera. Per ciascuna nave occorre memorizzare il nome (es. "Intesa", "Limpida", ecc.) e il numero di passeggeri a bordo, mentre il personale va memorizzato all'interno di una lista. La classe `Nave` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, la lista dello staff è vuota).
2. Possedere opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione della nave (inclusa la descrizione di tutto il personale in viaggio).
4. Possedere il metodo `ruolo` che, dato un ruolo, restituisca un insieme contenente tutto il personale che a bordo ricopre tale ruolo.
5. Presentare il metodo `aggiungi` che, dato un oggetto `Staff`, lo inserisca all'interno della lista, mantenendo la lista ordinata secondo il punto 5. dell'esercizio 4.
6. Possedere il metodo `quanti` che restituisca il numero di persone dello staff presente a bordo.
7. Presentare il metodo `maxStipendio` che restituisca il membro dello staff avente stipendio massimo tra quelli presenti a bordo.

Esercizio 6 (8 punti)

Si scriva un'applicazione per la gestione della flotta della Litorale Crociere che:

1. Crei un insieme di oggetti `Nave`.
2. Crei un oggetto `Nave`, lette da tastiera le informazioni necessarie.
3. Inserisca l'oggetto di cui al punto 2. all'interno dell'insieme di cui al punto 1., controllando che tale inserimento sia possibile.
4. Crei un oggetto `Staff`, lette da tastiera le informazioni necessarie.
5. Inserisca il membro dello staff creato al punto 4. tra quelli a bordo della nave di cui al punto 2.
6. Stampi a video il numero totale di membri dello staff a bordo delle navi della flotta.
7. Stampi a video il numero di matricola del membro dello staff che percepisce lo stipendio maggiore tra quelli a bordo della flotta.
8. Letto da tastiera il nome di una nave, stampi a video, se esiste, la matricola del comandante di tale nave (su ogni nave della Litorale Crociere c'è sempre uno e un solo comandante).

Soluzione Esercizio 2

$85_{10} = 01010101_2$
$-113_{10} = 10001111_2$

$11100100_2 = -28_{10}$

01010101+
10001111
11100100

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`i++<N`	N/2 + 1	o (N + 1)/2 + 1
`sum+=V[++i]`	N/2	o (N + 1)/2
Totale	N + 3	o N + 4

Domanda 2:

2 assegnamenti	2
`j<N`	N/2 + 1
`sum+=f(V, ++j)`	N/2
`++j`	N/2
complessità di `f`	$N^2/4 + 2N$
Totale	$N^2/4 + 7N/2 + 3$

complessità di `f`: $\sum_{\substack{j=1 \\ (j \text{ dispari})}}^{N-1} (j+4) = \sum_{i=0}^{N/2-1} (2i+5) = \frac{N^2}{4} + 2N$

Domanda 3:

Complessità asintotica: $O(N^2)$

Soluzione Esercizio 4

```
class Staff implements Comparable<Staff> {
  private String nome, cognome, ruolo;
  private int stipendio, matricola;

  public Staff(String nome, String cognome, String ruolo, int stipendio,
       int matricola) {
    this.nome = nome;
    this.cognome = cognome;
    this.ruolo = ruolo;
    this.stipendio = stipendio;
    this.matricola = matricola;
  }

  public String getRuolo() { return ruolo; }
  public String getNome() { return nome + " " + cognome; }
  public int getStipendio() { return stipendio; }
  public int getMatricola() { return matricola; }

  public String toString() {
    return getNome() + " (" + ruolo + "-" + matricola + ":" + stipendio;
  }

  public boolean equals(Object o) { return equals((Staff) o); }
  public boolean equals(Staff s) {
    return matricola==s.matricola;
  }

  public int compareTo(Staff s) {
    int ret = cognome.compareTo(s.cognome);
    if(ret==0) ret = nome.compareTo(s.nome);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
class Nave {
  private List<Staff> l;
  private String nome;
  private int passeggeri;
  public Nave(String nome, int passeggeri) {
    this.nome = nome; this.passeggeri = passeggeri;
    l=new LinkedList<Staff>();
  }
  public String getNome() { return nome; }
  public int getPasseggeri() { return passeggeri; }
  public String toString() {
    return nome + "( " + passeggeri + "):" + l.toString();
  }
  public Set<Staff> ruolo(String ruolo) {
    Set<Staff> s = new HashSet<Staff>();
    for(Staff x:l) if(x.getRuolo().equals(ruolo)) s.add(x);
    return s;
  }
  public void aggiungi(Staff s) {
    int i=0;
    while((i<l.size())&&(l.get(i).compareTo(s)<0)) i++;
    l.add(i, s);
  }
  public int quanti() { return l.size(); }
  public Staff maxStipendio() {
    Staff max=null;
    for(Staff x:l)
      if(max==null||x.getStipendio()>max.getStipendio()) max=x;
    return max;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Nave> s=new TreeSet<Nave>();
    Scanner scanner=new Scanner(System.in);
    Nave n=new Nave(scanner.nextLine(), scanner.nextInt());
    if(!s.add(n)) System.out.println("Nave già esistente!");
    Staff x=new Staff(scanner.nextLine(), scanner.nextLine(),
      scanner.nextLine(), scanner.nextInt(), scanner.nextInt());
    n.aggiungi(x);
    int totale = 0;
    for(Nave z: s) totale += z.quanti();
    System.out.println(totale);
    Staff max=null;
    for(Nave z: s) {
      Staff m=z.maxStipendio();
      if(max==null||m.getStipendio()>max.getStipendio()) max=m; }
    System.out.println(max.getMatricola());
    String nome = scanner.nextLine();
    for(Nave z: s) if(z.getNome().equals(nome))
    for(Staff x: z.ruolo("Comandante"))
      System.out.println(x.getMatricola());
  }
}
```

Esame di Fondamenti di Informatica T-1/T-A
Ingegneria Gestionale (A-K)

Appello del 18/10/2013

Compito B

NOTA: Per il superamento dell'esame è **necessario** ottenere la sufficienza nello svolgimento dell'Esercizio 1.

Esercizio 1 (4 punti)

1. Definire il concetto di complessità asintotica di un algoritmo.
2. Si illustri il concetto di ricorsione *tail*, indicando quali siano le differenze nell'utilizzo della memoria rispetto ad una normale ricorsione lineare.

Esercizio 2 (2 punti)

Convertire in binario i numeri **75** e **–123**, supponendo di utilizzare una rappresentazione a 8 bit in complemento a 2. Si esegua infine la somma dei due numeri, riconvertendo il risultato in base 10, motivando eventuali differenze tra il risultato ottenuto e quello atteso.

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

```
public static int f(int V[], int N) {
  int i=0, sum=0;
  while(++i<N)
    sum+=V[i++];
  return sum;
}

public static int g(int V[], int N) {
  int j=0, sum=0;
  for (; j<N; j++)
    sum+=f(V, j++);
  return sum;
}
```

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `N` (si distinguano i casi in cui `N` assume valori pari da quelli in cui assume valori dispari).
2. Calcolare la complessità in passi base del metodo `g` nei termini del parametro `N` (si supponga `N` pari).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 4 (5 punti)

La Air Carnia è una società che organizza voli nei paesi mitteleuropei. Di recente, ha deciso di informatizzare la gestione del proprio personale di volo, composto da personale altamente professionale, utile a rendere possibile le trasvolate. A tal scopo, per ogni membro del personale vengono memorizzati nome e cognome, incarico (es. "pilota", "hostess", "steward"), codice personale e ore di volo totali. Si scriva una classe `Personale` per la Air Carnia che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presenti il metodo `toString` che fornisca una descrizione del membro del personale.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Personale` (la verifica va fatta sul numero di codice personale).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Personale` passato come parametro (in ordine alfabetico per cognome e nome).

Esercizio 5 (8 punti)

Si scriva una classe `Aereo` che memorizzi le informazioni relative al personale in viaggio su un determinato aereo. Per ciascun aereo occorre memorizzare il codice (es. "NE143", "NE751", ecc.), il numero di passeggeri a bordo, mentre il personale va memorizzato all'interno di una lista. La classe `Aereo` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, la lista del personale è vuota).
2. Possedere opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione dell'aereo (inclusa la descrizione di tutto il personale in viaggio).
4. Possedere il metodo `incarico` che, dato un incarico, restituisca un insieme contenente tutto il personale che a bordo ricopre tale incarico.
5. Presentare il metodo `aggiungi` che, dato un oggetto `Personale`, lo inserisca all'interno della lista, mantenendo la lista ordinata secondo il punto 5. dell'esercizio 4.
6. Possedere il metodo `quanti` che restituisca il numero di membri del personale presenti a bordo.
7. Presentare il metodo `maxOre` che restituisca il membro del personale avente il maggior numero di ore di volo tra quelli presenti a bordo.

Esercizio 6 (8 punti)

Si scriva un'applicazione per la gestione della flotta della Air Carnia che:

1. Crei un insieme di oggetti `Aereo`.
2. Crei un oggetto `Aereo`, lette da tastiera le informazioni necessarie.
3. Inserisca l'oggetto di cui al punto 2. all'interno dell'insieme di cui al punto 1., controllando che tale inserimento sia possibile.
4. Crei un oggetto `Personale`, lette da tastiera le informazioni necessarie.
5. Inserisca il membro del personale creato al punto 4. tra quelli a bordo dell'aereo di cui al punto 2.
6. Stampi a video il numero totale di membri del personale a bordo degli aerei della flotta.
7. Stampi a video codice del membro del personale che ha il massimo numero di ore di volo tra quelli a bordo della flotta.
8. Letto da tastiera il codice di un aereo, stampi a video, se esiste, il codice personale del comandante di tale aereo (su ogni aereo della Air Carnia c'è sempre uno e un solo comandante).

Soluzione Esercizio 2

$75_{10} = 01001011_2$
$-123_{10} = 10000101_2$

$11100100_2 = -48_{10}$

01001011+
10000101
11010000

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`++i<N`	`N`/2 + 1	o (`N` – 1)/2 + 1
`sum+=V[i++]`	`N`/2	o (`N` – 1)/2
Totale	`N` + 3	o `N` + 2

Domanda 2:

2 assegnamenti	2
`j<N`	`N`/2 + 1
`sum+=f(V, ++j)`	`N`/2
`++j`	`N`/2
complessità di `f`	$N^2/4 + 2N$
Totale	$N^2/4 + 5N/2 + 3$

complessità di `f`: $\sum_{\substack{j=0 \\ (j \text{ pari})}}^{N-2} (j+3) = \sum_{i=0}^{N/2-1} (2i+3) = \frac{N^2}{4} + N$

Domanda 3:
Complessità asintotica: $O(N^2)$

Soluzione Esercizio 4

```
class Personale implements Comparable<Personale> {
  private String nome, cognome, incarico;
  private int volo, codice;

  public Personale(String nome, String cognome, String incarico, int volo,
      int codice) {
    this.nome = nome;
    this.cognome = cognome;
    this.incarico = incarico;
    this.volo = volo;
    this.codice = codice;
  }

  public String getIncarico () { return incarico; }
  public String getNome() { return nome + " " + cognome; }
  public int getVolo() { return volo; }
  public int getCodice() { return codice; }

  public String toString() {
    return getNome() + " (" + incarico + "-" + codice + ":" + volo;
  }

  public boolean equals(Object o) { return equals((Personale) o); }
  public boolean equals(Personale p) {
    return codice==p.codice;
  }

  public int compareTo(Personale p) {
    int ret = cognome.compareTo(p.cognome);
    if(ret==0) ret = nome.compareTo(p.nome);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
class Aereo {
  private List<Personale> l;
  private String nome;
  private int passeggeri;
  public Aereo(String nome, int passeggeri) {
    this.nome = nome; this.passeggeri = passeggeri;
    l=new ArrayList<Personale>();
  }
  public String getNome() { return nome; }
  public int getPasseggeri() { return passeggeri; }
  public String toString() {
    return nome + "( " + passeggeri + "):" + l.toString();
  }
  public Set<Personale> incarico(String incarico) {
    Set<Personale> s = new TreeSet<Personale>();
    for(Personale x:l) if(x.getIncarico().equals(incarico)) s.add(x);
    return s;
  }
  public void aggiungi(Personale p) {
    int i=0;
    while((i<l.size())&&(l.get(i).compareTo(p)<0)) i++;
    l.add(i, p);
  }
  public int quanti() { return l.size(); }
  public Personale maxOre() {
    Personale max=null;
    for(Personale x:l)
      if(max==null||x.getVolo()>max.getVolo()) max=x;
    return max;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Aereo> s=new HashSet<Aereo>();
    Scanner scanner=new Scanner(System.in);
    Aereo a=new Aereo (scanner.nextLine(), scanner.nextInt());
    if(!s.add(a)) System.out.println("Aereo già esistente!");
    Personale p=new Personale(scanner.nextLine(), scanner.nextLine(),
      scanner.nextLine(), scanner.nextInt(), scanner.nextInt());
    a.aggiungi(p);
    int totale = 0;
    for(Aereo z: s) totale += z.quanti();
    System.out.println(totale);
    Personale max=null;
    for(Aereo z: s) {
      Personale m=z.maxOre();
      if(max==null||m.getVolo()>max.getVolo()) max=m;
    }
    System.out.println(max.getCodice());
    String nome = scanner.nextLine();
    for(Aereo z: s) if(z.getNome().equals(nome))
      for(Personale x: z.incarico("Comandante"))
        System.out.println(x.getCodice());
  }
}
```