Prova scritta di Algoritmi e Strutture Dati (1^o anno)

Ottobre 2000

L'ordine in cui vengono svolti gli esercizi non è rilevante (in altre parole: se avete problemi a svolgerne uno potete passare tranquillamente ai successivi e farlo dopo).

I punti previsti per ogni esercizio si riferiscono ad uno svolgimento completamente corretto.

NOME:

COGNOME:

MATRICOLA:

CORSO:

==

Non scrivere qui sotto

Esercizio	Punti previsti	Punti assegnati
1	5	
2	9	
3	10	
4	6	
Totale	30	

Esercizio 1 (punti 5)

Consideriamo il seguente codice C:

```
#include <stdio.h>

int x = 50;

void pp (int b[]);
int ff (int *b);
void qq (int *n);
int gg(int n);
void rr (int n);

main()
{
  int x = 20;   int y;   int a[10]={0,0,0,0,0,0,0,0,0,0};

  pp(a);
  x = ff(a);
  qq(&y);
  rr(x);
  printf("%d, %d\n ",x, y);
}

void pp(int b[])
{ int i; for (i=0;i<10;i++) scanf("%d",&b[i]); }

int ff(int *b)
{ int i;
  for (i=1;i<10;i++) b[i]=b[i]+b[i-1];
  return(b[9]);
}

void qq(int *n)   { *n = gg(x);}

int gg(int n)     { n = n/2; return(n);}

void rr(int n)    { n = n+x; printf("%d, ",n);}
```

Domanda: Dire qual'è l'output del programma fornendo in input la successione 1 2 3 4 5 6 7 8 9 10 (spiegare **brevemente** come cambiano i valori delle variabili durante il programma).

Esercizio 2 (punti 9)

Consideriamo il tipo di dato *Insieme* di caratteri implementato con alberi binari di ricerca.

Domande:

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato;
2. Definire in pseudo-codice la procedura **Empty** che crea un insieme vuoto;
3. Definire in pseudo-codice la procedura **Insert** che inserisce un nuovo elemento in un insieme (non si ammettono duplicati);
4. Definire in pseudo-codice una procedura **Intersection** che prende in input due insiemi e ne restituisce uno corrispondente alla loro intersezione **cancellando gli alberi in input e senza allocare nuove celle**.

Esercizio 3 (punti 10)

Consideriamo il tipo di dato *Successione* di numeri interi implementato con array.

Domande:

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato;
2. Definire in pseudo-codice una procedura che, mediante un algoritmo a piacere, ordina l'array in input **senza usare al suo interno altri array o liste dinamiche ausiliarie**;
3. Stimare la complessità della procedura implementata in $\Theta()$ in funzione della dimensione n dell'array in input.

Esercizio 4 (punti 6)

Consideriamo la seguente procedura ricorsiva in C.

```
int fff(int a[]; int inf; int sup)
{
  int k; int m;
  if (inf>=sup)
     return(a[inf]);
  else
     { m = (sup + 2 * inf)/ 3;
       k = fff(a,m,sup);
       return(k);
     }
}
```

Domanda: Stimare la complessità in $\Theta()$ della chiamata `fff(A,1,N)` in funzione di `N`.

SUGGERIMENTO: il valore di `m` ad ogni chiamata corrisponde all'indice che si trova ad **un terzo** della porzione di array che va da `inf` a `sup`.

NOTE:

- I passaggi di parametro si intendono secondo le regole del C;
- Non è necessario fare conti molto dettagliati esplicitando tutte le costanti. Basta considerare il costo di tutte le parti di programma mettendo eventualmente in evidenza le operazioni dominanti (una volta che si sono individuate si possono ignorare le altre).
- Non limitarsi a dare solo il risultato: bisogna spiegare come ci si arriva.