Informatica Generale II - Prova teorica

A.A. 2005/2006

Esame: 19 aprile 2006
Codice: IDHK

1. Quali degli alberi in figura 1 *non sono* alberi binari di ricerca, mentre gli altri lo sono?

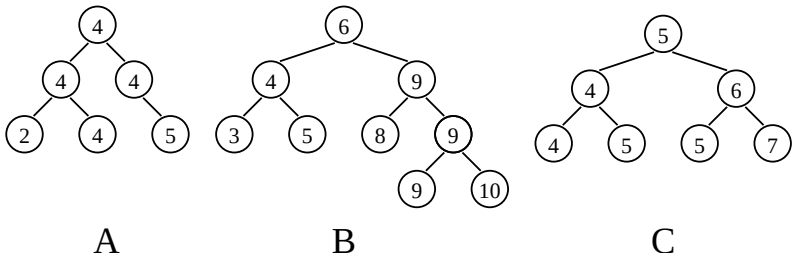

Figura 1:

   (a) B
   (b) A,C
   (c) A
   (d) nessuno
   (e) non rispondo

2. Si analizzi il seguente codice:

```
void foo(int A[], int N) {
  for (int i=0; i < N-1; ++i) {
    int min = i;
    for (int j=i+1; j < N; ++j)
      if (A[j] < A[min]) min = j;
    swap(A[i], A[min]);
  }
}
```

   Esso implementa l'algoritmo:

   (a) Mergesort
   (b) Selection sort
   (c) Bubblesort
   (d) Quicksort
   (e) non rispondo

3. Quale tipo di attraversamento d'albero implementa il seguente codice?

```
void foo-order(Node* l, void visit(Node*))
  StackPtr s = new Stack();

  Push(s, i);
  while(! StackIsEmpty(s)) {
```

```
    Node * h = Pop(s);

    visit(h);
    if (h->right != NULL) Push(s, h->right);
    if (h->left != NULL) Push(s, h->left);
  }
  delete s;
}
```

(a) inorder
(b) level-order
(c) preorder
(d) postorder
(e) non rispondo

4. Un albero N-ario è:

(a) nessuna delle altre risposte proposte è accettabile
(b) un albero con esattamente N nodi
(c) un albero i cui nodi hanno N figli
(d) un albero con cammino minimo N dalla radice alle foglie
(e) non rispondo

5. Si consideri il seguente frammento di codice:

```
{
  double x=10;
  double *j;
  j = new double[x];
  ...
}
```

Quale delle seguenti affermazioni è corretta:

(a) le espressioni `j[i]` e `*(i + j)` sono espressioni equivalenti per indicare l'(i+1)-esimo elemento dell'array
(b) le espressioni `j[i]` e `*(i + j)` sono espressioni *non* equivalenti
(c) le espressioni `j[i]` e `*(i + j)` sono espressioni equivalenti per indicare l'(i)-esimo elemento dell'array
(d) le espressioni `j[i]` e `*(i + j)` sono espressioni equivalenti per indicare l'indirizzo dell'i-esimo elemento dell'array
(e) non rispondo

6. Una pila è:

(a) un multiinsieme di elementi in cui ogni eliminazione ha per oggetto l'elemento inserito per primo;
(b) un multiinsieme di elementi gestiti secondo la politica *fifo* (first in first out);
(c) un insieme di elementi gestiti secondo la politica *lifo* (last in first out);
(d) un multiinsieme di elementi gestiti secondo la politica *lifo* (last in first out);
(e) non rispondo

7. Si consideri il seguente frammento di codice:

```
struct Tipo1 {
 int a;

 Tipo1() { a = 0; }
 Tipo1(int _a) { a = _a; }
};

struct Tipo2 {
```

```
 Tipo1 a;

 Tipo2(int _a)  {
   a = Tipo1(_a);
 }
};
```

(a) è corretto e compilabile.
(b) è errato perché Tipo1 e Tipo2 contengono due campi con lo stesso nome.
(c) è corretto, ma Tipo2 non è istanziabile.
(d) è errato perché Tipo2 non ha un costruttore di default.
(e) non rispondo

8. Si consideri l'algoritmo di fusione (merge) tra due array ordinati aventi n elementi ciascuno:

(a) si implementa sempre tramite programmazione ricorsiva;
(b) ha complessità lineare poiché ogni volta che si inserisce un elemento nell'array risultato si esegue un numero costante di operazioni;
(c) ha complessità $\mathcal{O}(n^2)$;
(d) si avvale di questo algoritmo l'ordinamento per fusione che funziona nel seguente modo: si sceglie a caso un elemento di pivot sulla base del cui valore si divide in due parti l'array, di seguito ciascuna parte viene ricorsivamente ordinata e infine i due array vengono fusi;
(e) non rispondo

9. Quale tipo di attraversamento d'albero implementa il seguente codice?

```
void foo-order(Node* l, void visit(Node*))
  StackPtr s = new Stack();

  Push(s, i);
  while(! StackIsEmpty(s)) {
    Node * h = Pop(s);

    if ((h->left == NULL) && (h->right == NULL)) visit(h);
    else Push(s, new Node(h->data, true));
    if (h->right != NULL) Push(s, h->right);
    if (h->left != NULL) Push(s, h->left);
    if (h->flag == true) delete h;
  }

  delete s;
}
```

(a) level-order
(b) postorder
(c) preorder
(d) inorder
(e) non rispondo

10. Il seguente codice:

```
int a[5];
++(*((&(*(a+2))) + 2));
```

(a) è sbagliato perché `a` è un array e non un puntatore;
(b) ha l'effetto di incrementare di 1 il quarto elemento dell'array `a`
(c) ha l'effetto di incrementare di 1 il quinto elemento dell'array `a`
(d) ha l'effetto di incrementare di 1 il terzo elemento dell'array `a`
(e) non rispondo