1. Qual è l'output del seguente programma?

```c
#include<stdio.h>
void fun(int arr[])
{
  int i;
  int arr_size = sizeof(arr)/sizeof(arr[0]);
  for (i = 0; i < arr_size; i++)
      printf("%d ", arr[i]);
}

int main()
{
  int i;
  int arr[4] = {10, 20 ,30, 40};
  fun(arr);
  return 0;
}
```

- **10 20 30 40**
- **dipende dalla macchina**
- **10 20**
- **nessun output**
- **10 20 30**
- **10**

2. Collegate ciascuna strutture dati (a sinistra) e con l'implementazione opportuna (a destra) in modo che le operazioni fondamentali siano eseguite in tempo costante. A destra devono avanzare due implementazioni, meno efficienti.

coda senza limiti di capienza	lista concatenata con due puntatori al primo e all'ultimo elemento, con inserimento in fondo e cancellazione all'inizio
pila con capienza limitata	array allocato in fase di compilazione, con indice dell'elemento in cima
	array circolare allocato in fase di compilazione
	lista concatenata semplice con puntatore alla testa, con

coda con capienza limitata

pila senza limiti di capienza

inserimento e cancellazione all'inizio

lista concatenata con due puntatori al primo e all'ultimo elemento, con inserimento e cancellazione in fondo

lista concatenata con due puntatori al primo e all'ultimo elemento, con inserimento all'inizio e cancellazione in fondo

3. a) Considerate la seguente funzione, che deve calcolare il valore massimo contenuto nel vettore numbers. La funzione max calcola il massimo tra i suoi due argomenti.

```
1 int largest(int numbers[], int index) {
2    if ( ---------- )
3       return numbers[0];
4
5    return max( numbers[index], largest(numbers, index-1));
6 }
```

Come deve essere completato il caso base?

- **index == 0**
- **index == n**
- **index < n**
- **index == 0 || index == 1**
- **numbers[index] > numbers[index - 1]**
- **numbers[0] < numbers[1]**
- **index == n-1**

b) Come deve essere invocata la funzione se si vuole calcolare l'elemento massimo in un vettore v di n elementi? Completate l'invocazione:

largest([ ], [ ])

c) Assumete d'ora in poi che il vettore v contenga i valori [1, 2, 5, 7, -2, 10, 9, 21, 3, 8].

Durante l'esecuzione della chiamata specificata al punto precedente, considerate la chiamata ricorsiva che termina per prima. Qual è il secondo argomento passato in questa chiamata e quale valore restituisce la funzione?

Il secondo argomento è [ ] e il valore restituito è [ ]

d) Con quali argomenti viene eseguita per la prima volta la funzione max (sempre considerando la chiamata e il vettore v specificati sopra)?

primo argomento: [ ]
secondo argomento: [ ]

e) E con quali argomenti viene eseguita l'ultima volta la funzione max?

Primo argomento: [ ]
Secondo argomento: [ ]

4. Dato un puntatore p al primo nodo di una lista concatenata, e un intero k, la funzione ricorsiva kLastCount stampa il valore del k-ultimo elemento della lista e restituisce la lunghezza della lista.

Ad esempio, sia p il puntatore alla testa della lista 3 --> 2 --> 5 --> 1 --> 7.
Se k vale 1, allora f(p,k) stampa "7" e restituisce 5;
se k vale 3, allora f(p,k) stampa "5" e restituisce 5;
se k vale 10, allora f(p,k) non stampa nulla e restituisce 5.

Qui sotto sono riportate le righe che formano la funzione kLastCount, messe in disordine, con 3 righe in più che non servono.

```
if ( count == k )
}
kLastCount( p -> next, k, count );
return count;
int count;
count = 1 + kLastCount( p -> next, k );
int kLastCount( Node p, int k ){
if ( *count == k )
count++;
if ( p == NULL )
printf( "%d\n", p -> val );
return 0;
```

Risposta:

5. Considerate le seguenti definizioni. Il tipo `Graph` è usato per rappresentare un grafo orientato i cui i vertici sono interi e implementato mediante un array (allocato dinamicamente) di liste di adiacenza.

```
struct listnode {
   struct listnode *next;
   int v;
};

typedef struct graph {
   int n;  /* numero di vertici */
   struct listnode **a;
} Graph;
```

a) Scrivete una funzione `Graph *new( int n )` che alloca lo spazio per un nuovo grafo di n vertici e lo inizializza opportunamente.

Risposta:

b) Cosa rappresenta `*(a + i)` ?

- **un vertice adiacente al vertice i**
- **l'array dei vertici adiacenti al vertice i**
- **l'inizio della lista di adiacenza del vertice i**
- **l'insieme dei vertici da cui parte un arco verso il vertice i**
- **l'array della lista di adiacenza del vertice i**
- **un arco uscente dal vertice i**
- **un arco entrante nel vertice i**

c) Come è rappresentato l'arco dal vertice v al vertice w?

- **L'arco è rappresentato da un elemento nella lista di adiacenza a[v]**
- **L'arco è rappresentato da a[v]**
- **L'arco è rappresentato da un elemento nella lista di adiacenza a[w]**
- **L'arco è rappresentato da una coppia di elementi, uno nella lista di adiacenza a[v] e l'altro nella lista di adiacenza a[w]**

L'arco è rappresentato da a[w]

d) Scrivete una funzione `inverti(G, v, w)` che modifica il grafo nel modo seguente: se nel grafo c'è l'arco dal vertice v al vertice w, la direzione dell'arco viene invertita, altrimenti il grafo resta invariato.

Risposta:

e) Descrivete un algoritmo che stampa, in ordine crescente, l'elenco dei gradi dei suoi vertici (il grado di un vertice v è il numero degli archi che escono da v).

Potete spiegare la vostra soluzione a parole (siate brevi), con pseudocodice, con degli schemi, oppure potete scriverla in C.

Risposta:

f) Stimate la complessità in tempo dell'algoritmo descritto al punto precedente. Giustificate la risposta.

Risposta:

g) Stimate la complessità in spazio dell'algoritmo descritto sopra. Giustificate la risposta.

Risposta: