Manipolazione di strutture dati

1.1

Esercizio tratto da *Programmazione Scientifica - Barone, Marinari, Organtini, Ricci-Tersenghi*

Viene data la seguente struttura

```
struct dataStruct {
    int *data;
    int size, numData;
};
```

utilizzata per gestire un array dinamico `data` di interi con il supporto di `size` – che indica la dimensione massima di `data` – e `numData` – che invece indica il numero attuale di elementi presenti in `data`.
Scrivere una funzione che ridimensioni l'array `data`, del tipo:

```
void ResizeArray(int **orig, int size);
```

1.2

Scrivere due funzioni per inserire e rimuovere un elemento dalla struttura. I prototipi delle funzioni sono:

```
void insertInBucket(dataStruct bucket, int newItem);
int removeFromBucket(dataStruct bucket, int index);
```

La prima deve ricevere un nuovo elemento e modificare opportunamente i campi `numData` e `size`. La seconda deve ricevere l'indice di un elemento da rimuovere, modificare opportunamente le dimensioni e ritornare il valore rimosso. Attenzione alla gestione dei valori `size` e `numData`!

Consiglio: un modo efficiente per allocare nuova memoria è quello di raddoppiare `size` ogni qual volta `numData == size`. In questo modo si è sicuri che il numero di operazioni per l'allocazione di nuova memoria cresca logaritmicamente con la lunghezza della struttura che è quindi un'operazione efficiente.

2

Scrivere un programma che ordini gli item di una linked list usando l'algoritmo di sorting `BubbleSort`.

3

Scrivere un programma che data una Linked list, permetta di spostare una sezione arbitraria di items in fondo alla lista.

La funzione dovrebbe avere questa signature:

```
LinkedListNode* moveChunk(LinkedListNode* head, int startIndex, int lenOfChunk);
```

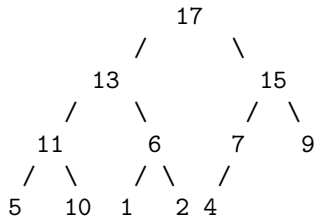
4

Esercizio tratto da *Programmazione Scientifica - Barone, Marinari, Organtini, Ricci-Tersenghi*

Uno heap (binario) è un vettore `v[]` i cui elementi hanno tra di essi relazioni di parentela definite come segue:

- `v[i]` ha come "genitore" l'elemento `v[(int)(i/2)]`
- `v[i]` ha come "figli" gli elementi `v[2*i]` e `v[2*i+1]`

Le relazioni di parentela si capiscono più chiaramente rappresentando l'array come un albero binario (attenzione, non è veramente un albero!).
Ad esempio l'array `[0, 17, 13, 15, 11, 6, 7, 9, 5, 10, 1, 2, 4]` si rappresenta come



dove si preferisce sprecare qualche byte per il valore `0` e semplificare la gestione degli indici nel codice.

Proprietà fondamentale dello heap: *la priorità di un elemento deve essere sempre non superiore a quella dell'elemento genitore*, in questo caso `v[i] <= v[(int)(i/2)]`.
Dato un array di interi, la cui dimensione deve poter crescere dinamicamente, si chiede di implementare:

- la funzione di inserimento di un elemento
- la funzione di rimozione di un elemento
- una funzione di ordinamento che rispetti la proprietà fondamentale e consenta di creare uno heap a partire da un array generato randomicamente

Consigli:

- sfruttare l'implementazione della struttura all'esercizio 1
- è molto utile ragionare graficamente con l'ausilio della rappresentazione ad albero.