Coda con priorita’

- A.K.A. Priority Queue
- Coda da cui gli elementi vengono estratti in base alla loro “priorita’”
- Che cos’e’ la priorita’?
  - Un parametro qualsiasi capace di definire un ordinamento degli elementi
  - E.g. Quanto pagano come canone di servizio

Nota Bene

- La coda FIFO e’ un caso particolare della coda con priorita’
  - Parametro di priorita’ la “data” di arrivo in coda

```
public class PQueue {
    LinkedList coda = new LinkedList();
    public void add(Object o) {...}
    public Object get() {...}
    public int size() {...}
}
```

Una coda con priorita' generica implementa l'interfaccia FIFO?

- In generale no!
- la coda FIFO e' solo un caso ***speciale*** di coda con priorita'
- come ottengo l'ordine degli oggetto recuperati tramite la ***get()***?

Possibili Implementazioni

- Scelta su quando ordinare
- Ordino in ingresso
  - approccio eager
- Ordino in uscita
  - approccio lazy

Ordino in ingresso

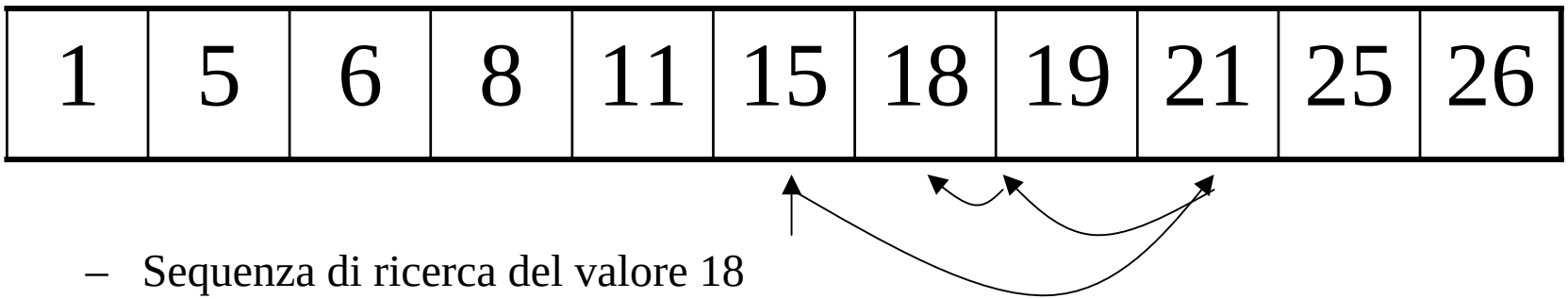
- Ogni elemento inserito richiede un ordinamento della coda
- Inserire un singolo elemento in una coda ordinata
  - Insertion sort

Algoritmo Insertion Sort Lineare

```
if(lista vuota) {
    inserisci corrente e termina
} else {
    while(next != null && next < corrente) {
        <avanza nella lista>
    }
    <inserisci corrente prima di next>
}
```

Complessita'?

- L'algoritmo tende ad eseguire un numero di attraversamenti e confronti pari alla lunghezza della lista
  - O(n)
- Posso migliorare?
  - Certo! Ricerca binaria.



  - Sequenza di ricerca del valore 18
  - L'algoritmo tende a fare un numero di attraversamenti e confronti pari a logaritmo in base 2 del numero di elementi
    - $O(\log_2 n)$

Esempio di Implementazione

```
int binarySearch( Object key, Object[] r ){
    int high, i, low;
    for ( low=(-1), high=r.length; high-low > 1; ) {
        i = (high+low) / 2;
        if ( key.compareTo(r[i]) <= 0 )
            high = i;
        else
            low = i;
    }
    if ( key.equals(r[high]) )
        return( high );
    else
        return( -1 );
}
```

Compito a casa

- Studiare le inefficienze dell’implementazione d’esempio
- Trovare soluzioni per eliminare tali inefficienze

Ricerca Binaria in Java

```
public static int binarySearch(List list,
                               Object key,
                               Comparator c)
```

Searches the specified list for the specified object using the binary search algorithm. The list must be sorted into ascending order according to the specified comparator (as by the `Sort(List, Comparator)` method, above), prior to making this call. If it is not sorted, the results are undefined. If the list contains multiple elements equal to the specified object, there is no guarantee which one will be found.

This method runs in log(n) time for a "random access" list (which provides near-constant-time positional access). If the specified list does not implement the `RandomAccess` and is large, this this method will do an iterator-based binary search that performs O(n) link traversals and O(log n) element comparisons.

Parameters:
`list` - the list to be searched.
`key` - the key to be searched for.
`c` - the comparator by which the list is ordered. A `null` value indicates that the elements' *natural ordering* should be used.
Returns:
index of the search key, if it is contained in the list; otherwise, `(-(`*`insertion point`*`) - 1)`. The *insertion point* is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or `list.size()`, if all elements in the list are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.
Throws:
`ClassCastException` - if the list contains elements that are not *mutually comparable* using the specified comparator, or the search key in not mutually comparable with the elements of the list using this comparator.
See Also:
`Comparable`, `sort(List, Comparator)`

Ordino in uscita

- Se ordino in uscita:
  - Inserimento a costo fisso
  - Estrazione richiede un ordinamento
- SEMPRE?

Non Necessariamente

- Devo riordinare solo se dalla precedente estrazione e’ avvenuto un inserimento
- Se non e’ avvenuto alcun inserimento ho ancora la lista in un ordine valido
  - Uso dei flag di stato
  - L’esecuzione del sort setta il flag
  - L’inserimento di un elemento resetta il flag

Pseudo implementazione

```
public class Pqueue {
	private boolean ordered = false;
	private LinkedList pq = new LinkedList();
	public void add(Object o) {
		<inserisco o>;
		ordered = false;
	}
	public Object get() {
		if(!ordered) {
			Collections.sort(pq);
			ordered = true;
		}
		return <funzione per estrarre elemento da lista ora ordinata>;
	}
}
```