

# Programmazione I-B 2021-22

Laboratorio T2  
(ultima cifra matricola PARI)

Attilio Fiandrotti  
[attilio.fiandrotti@unito.it](mailto:attilio.fiandrotti@unito.it)

24 Novembre 2022

# Outline

- Soluzione esercizi ricorsione su array
- Ricorsione dicotomica
- Esercizi ricorsione dicotomica

# Soluzione esercizi ricorsione su array

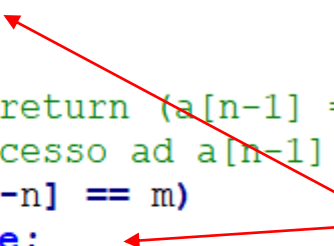
# ESERCIZIO ArrayRic – esisteRic()

- Scrivere un metodo *esisteRic()* che, dati un array di interi *a* ed un intero *m*, ritorni *true* se *a* contiene *m*, *false* altrimenti. Si implementi il metodo con un algoritmo ricorsivo gestendo opportunamente il caso in cui *a* sia *null* in un metodo wrapper e si verifichi il funzionamento del metodo per l'array *a* fornito nel test case e per i valori di *m* {0, 10, 40}.

# ESERCIZIO ArrayRic – esisteRic()

```
/* Ritorna true se m è presente in a, false altrimenti */
public static boolean esisteRic(int[] a, int m, int n) {
    //Caso base, volutamente lasciato vuoto
    if (n == 0) {
        return false;
    } else {
        // Implementa return (a[n-1] == m) || esiste2(a, n-1);
        // notare l'accesso ad a[n-1] anzichè a[n]
        if (a[a.length-n] == m)
            return true;
        else
            return esisteRic(a, m, n-1);
    }
}

/* Metodo wrapper */
public static boolean esisteRic(int[] a, int m) {
    boolean ret = false;
    if (a != null) {
        ret = esisteRic(a, m, a.length);
    }
    return ret;
}
```



*la ricorsione termina*

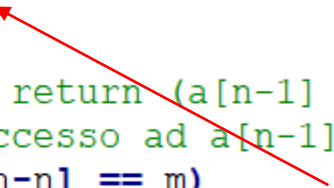
## ESERCIZIO ArrayRic – esisteRicPos()

- Scrivere un metodo *esisteRicPos()* che, dati un array di interi  $a$  ed un intero  $m$ , ritorni la prima posizione di  $m$  in  $a$ , la lunghezza di  $a$  altrimenti. Si implementi il metodo con un algoritmo ricorsivo gestendo opportunamente il caso in cui  $a$  sia *null* in un metodo wrapper e si verifichi il funzionamento del metodo per l'array  $a$  fornito nel test case e per i valori di  $m$   $\{0, 10, 40\}$ .

# ESERCIZIO ArrayRic – esisteRicPos()

```
/* Ritorna true se m è presente in a, false altrimenti */
public static boolean esisteRic(int[] a, int m, int n) {
    //Caso base, volutamente lasciato vuoto
    if (n == 0) {
        return false;
    } else {
        // Implementa return (a[n-1] == m) || esiste2(a, n-1);
        // notare l'accesso ad a[n-1] anzichè a[n]
        if (a[a.length-n] == m)
            return true;
        else
            return esisteRic(a, m, n-1);
    }
}

/* Metodo wrapper */
public static boolean esisteRic(int[] a, int m) {
    boolean ret = false;
    if (a != null) {
        ret = esisteRic(a, m, a.length);
    }
    return ret;
}
```



*la ricorsione termina*

## ESERCIZIO ArrayRic – tuttiPari()

- Scrivere un metodo *tuttiPari()* che, dato un array di interi *a*, ritorni *true* se tutti gli elementi di *a* sono pari, *false* altrimenti. Si implementi il metodo con un algoritmo ricorsivo gestendo opportunamente il caso in cui *a* sia *null* in un metodo wrapper e si verifichi il funzionamento del metodo per l'array *x* fornito nel test case.



# ESERCIZIO ArrayRic – tuttiPari()

- La condizione «tutti gli elementi di  $a$  sono pari» si può codificare con
$$(a[0] \%2 ==0) \&\& (a[1] \%2 ==0) \&\& \dots \&\& (a[n-1] \%2 ==0)$$
- Il metodo ricorsivo deve ritornare tale espressione logica

# ESERCIZIO ArrayRic – tuttiPari()

```
/* Ritorna true se tutti gli elementi dell'array sono pari, false altrimenti */
public static boolean tuttiPari(int[] a, int n) {
    if (n < 0) {
        // Caso base: ricorsione oltre il primo elemento di a
        return true;
    }
    else {
        return (a[n] % 2 == 0) && tuttiPari(a, n - 1);
    }
}

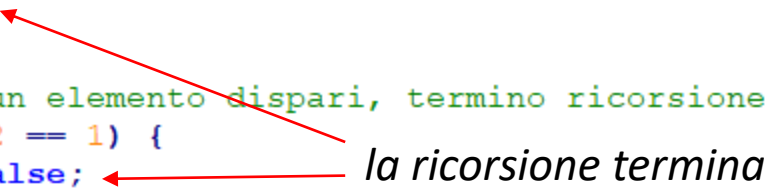
/* Metodo wrapper */
public static boolean tuttiPari(int[] a) {
    boolean ret = false;
    if (a != null) {
        ret = tuttiPari(a, a.length - 1);
    }
    return ret;
}
```

# ESERCIZIO ArrayRic – tuttiPari()

```
/* Ritorna true se tutti gli elementi dell'array sono pari, false altrimenti */
public static boolean tuttiPari(int[] a, int i) {
    if (i < 0) {
        // Caso base: ricorsione oltre il primo elemento di a
        return true;
    }
    else {
        // Trovato un elemento dispari, termino ricorsione
        if (a[i] % 2 == 1) {
            return false;
        }
        // Proseguo con la ricorsione
        else {
            // "true &&" esplicitato per ragioni didattiche
            return true && tuttiPari(a, i - 1);
        }
    }
}
```

# ESERCIZIO ArrayRic – tuttiPari()

```
/* Ritorna true se tutti gli elementi dell'array sono pari, false altrimenti */
public static boolean tuttiPari(int[] a, int i) {
    boolean ret = true;
    if (i < 0) {
        // Caso base: ricorsione oltre il primo elemento di a
        ret = true; /* ridondante */
    }
    else {
        // Trovato un elemento dispari, termino ricorsione
        if (a[i] % 2 == 1) {
            ret = false;
        }
        // Proseguo con la ricorsione
        else {
            // "true &&" esplicitato per ragioni didattiche
            ret = tuttiPari(a, i - 1);
        }
    }
    return ret;
}
```



*la ricorsione termina*

# ESERCIZIO ArrayRic – esisteMultiplo()

- Scrivere un metodo *esisteMultiplo(a, n)* che, dato un array di interi *a* ed un intero *m*, ritorni *true* se *a* contiene un elemento multiplo di *m*, false altrimenti. Si implementi il metodo con un algoritmo ricorsivo controvariante gestendo opportunamente il caso in cui *a* sia *null* in un metodo wrapper e si verifichi il funzionamento del metodo per l'array a fornito nel test case.

# ESERCIZIO ArrayRic – esisteMultiplo()

- La condizione « $a$  contiene un elemento multiplo di  $m$ » si può codificare con

$$(a[0] \% m == 0) \ || \ (a[1] \% m == 0) \ || \ ... \ || \ (a[n-1] \% m == 0)$$

- Il metodo ricorsivo deve ritornare tale espressione logica

# ESERCIZIO ArrayRic – esisteMultiplo()

```
/* Ritorna true se a contiene un elemento multiplo di m. */
public static boolean esisteMultiplo(int[] a, int m, int n) {
    // Caso base: ricorsione oltre l'ultimo elemento di a
    if (n >= a.length) {
        return false;
    }
    else {
        return (a[n] % m == 0) || esisteMultiplo(a, m, n + 1);
    }
}

/* Metodo wrapper */
public static boolean esisteMultiplo(int[] a, int m) {
    boolean ret = false;

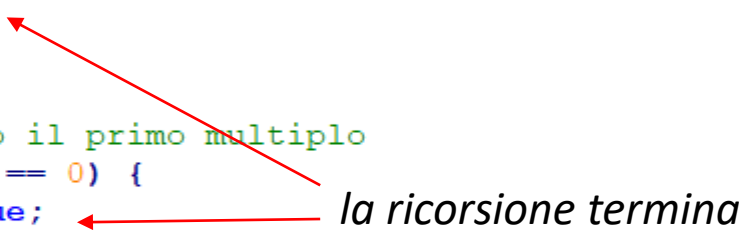
    if (a != null) {
        ret = esisteMultiplo(a, m, 0);
    }

    return ret;
}
```

# ESERCIZIO ArrayRic – esisteMultiplo()

```
/* Ritorna true se a contiene un elemento multiplo di m. */
public static boolean esisteMultiplo(int[] a, int m, int i) {
    boolean ret = true;
    // Caso base: ricorsione oltre l'ultimo elemento di a
    if (i >= a.length) {
        ret = false;
    }
    else {
        // Ho trovato il primo multiplo
        if (a[i] % m == 0) {
            ret = true;
        }
        // Ricorsione
        else {
            ret = esisteMultiplo(a, m, i + 1)
        }
    }

    return ret;
}
```



*la ricorsione termina*



# ESERCIZIO ArrayRic – filtraMaggioriDi()

- Scrivere un metodo *filtraMaggioriDi(a, m)* che, dato un array di interi *a* ed un intero *m*, ritorni un **nuovo** array contenente solo gli elementi di *a* che sono strettamente maggiori di *m* in ordine inverso per semplicità. Si implementi il metodo con un algoritmo ricorsivo in modo che, ad ogni passo della ricorsione, si mantenga un conteggio degli elementi che hanno superato il test. Nel caso base si avrà a disposizione il numero di elementi filtrati e si allocherà l'array da ritornare. Si verifichi il funzionamento del metodo per l'array a fornito nel test case.

# ESERCIZIO ArrayRic – filtraMaggioriDi()

- Per allocare il nuovo array  $b$  devo conoscerne la lunghezza, ovvero il numero di elementi di  $a$  maggiori di  $m$
- Tale informazione è nota solo al termine della ricorsione

# ESERCIZIO ArrayRic – filtraMaggioriDi()

```
/* Ritorna un nuovo array contenente solo gli elementi di a che sono
 * strettamente maggiori di m in ordine inverso.
 */
public static int[] filtraMaggioriDi(int[] a, int m, int n, int counter) {
    // caso base: alloca l'array per @counter elementi da memorizzare
    if (n < 0) {
        return new int[counter];
    }

    // caso generale
    if (a[n] > m) {
        // L'elemento a[n] viene copiato nell'array in uscita in posizione @counter.
        // Aumenta di 1 il conteggio, prosegui la ricorsione e poi
        // assegna a[n] nell'array da restituire
        int[] outArr = filtraMaggioriDi(a, m, n - 1, counter + 1);
        outArr[counter] = a[n];
        return outArr;
    }
    else // L'elemento a[n] non viene copiato nell'output
        return filtraMaggioriDi(a, m, n - 1, counter);
}

public static int[] filtraMaggioriDi(int[] a, int n) {
    int[] ret = null;

    if (a != null) {
        ret = filtraMaggioriDi(a, n, a.length - 1, 0);
    }

    return ret;
}
```

# La ricorsione dicotomica

# La ricorsione dicotomica

- Esempio fondamentale: algoritmo merge-sort visto a lezione
- Divido l'array ricorsivamente  $a$  fino ad arrivare ad array elementari
- 1 solo elemento per sotto-array è il caso base
- Faccio merge dei sotto-array in ordine inverso -> *merge()*
- I sotto-array to be merged sono sempre ordinati internamente, per questo l'algoritmo funziona

# La ricorsione dicotomica

- L'estremo sinistro e destro della ricorsione (del sotto array) sono  $l$  e  $r$
- $m$  rappresenta il *pivot* fra  $l$  ed  $r$   $m = \text{floor}((l+r)/2)$
- Ricorriamo, nell'ordine (per convenzione)
  1. Sul sotto-array sinistro:  $[l, m) \leftarrow a[m]$  escluso
  2. Sul sotto-array destro:  $[m, r) \leftarrow a[m]$  incluso
- Il margine sinistro ( $l$  o  $m$ ) é incluso, quello destro ( $m$  o  $r$ ) escluso

# ESERCIZIO ArrayDic – stampaDic()

- Scrivere un metodo *stampaDic()* che, dato un array di interi *a*, ne stampi il contenuto. Per esempio, dato l'array

```
int[] x = {10, 20, 30, 40};
```

il metodo dovrà stampare

«10 20 30 40 »

# ESERCIZIO ArrayDic – stampaDic()

- Signature semplificata *stampaDic*(int[] *a*, int *l*, int *r*)
- Invocazione iniziale su [0, a.length)
- Se *a* contiene un solo elemento ( $l == r-1$ ), stampo *a*[*l*]
- Altrimenti, calcolo  $m = (l + r) / 2$  e ricorro dicotomicamente:
  1. ricorsione sull'intervallo [*l*, *m*) di *a* -> *stampaDic*(*a*, *l*, *m*)
  2. ricorsione sull'intervallo [*m*, *r*) di *a* -> *stampaDic*(*a*, *m*, *r*)



# ESERCIZIO ArrayDic – stampaDic()

```
/* Stampa il contenuto di un array dicotomicamente
*/
public static void stampaDic(int[] a, int l, int r) {
    // Caso base, sotto-array di 1 solo elemento
    if (l+1 == r){
        System.out.print(a[l] + " ");
    }
    else {
        // Calcolo l'elemento in posizione centrale per l ed r
        int m = (r+l)/2;
        // Esploro il sotto-array sinistro [l, m)
        stampaDic(a, l, m);
        // e poi il sotto-array destro [m, r)
        stampaDic(a, m, r);
    }
    // Esplicitato per fini didattici
    return;
}

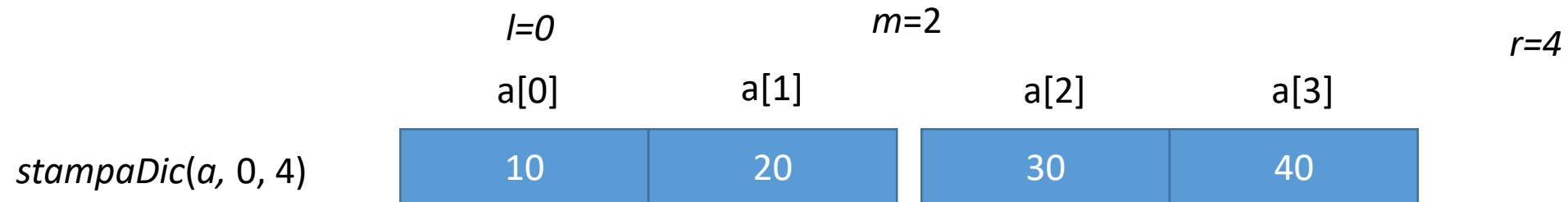
/* Metodo wrapper */
public static void stampaDic(int [] a) {
    if (a != null) {
        stampaDic(a, 0 /* l */ , a.length /* r */ );
    }
}
```

# ESERCIZIO ArrayDic – stampaDic()

	<i>l=0</i>				<i>r=4</i>
	a[0]	a[1]	a[2]	a[3]	
<i>stampaDic(a, 0, 4)</i>	10	20	30	40	

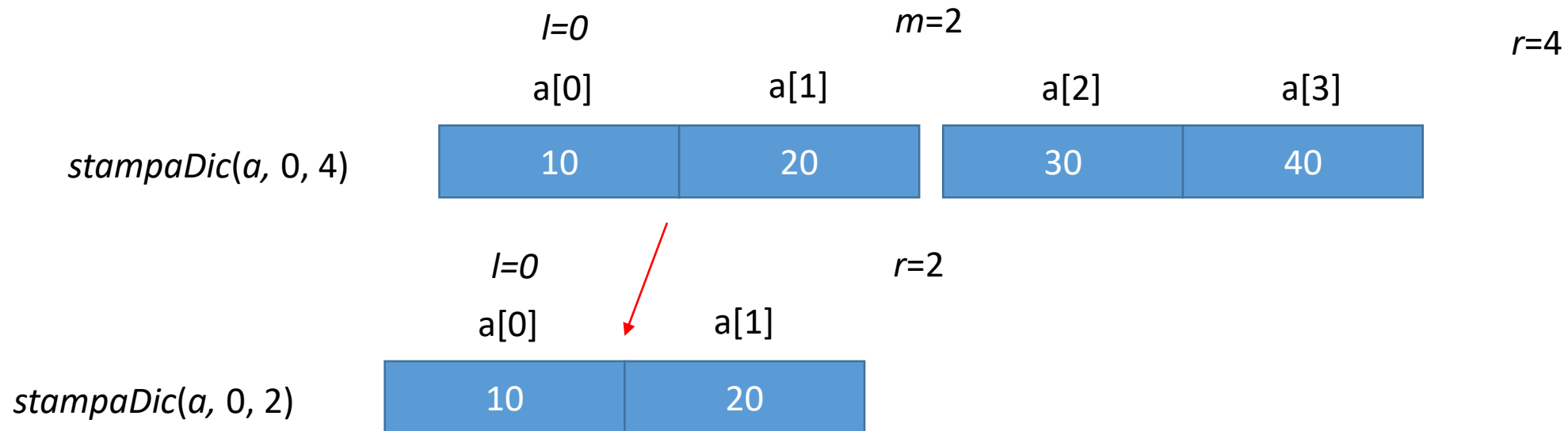
```
$java ArrayDicTest
```

# ESERCIZIO ArrayDic – stampaDic()



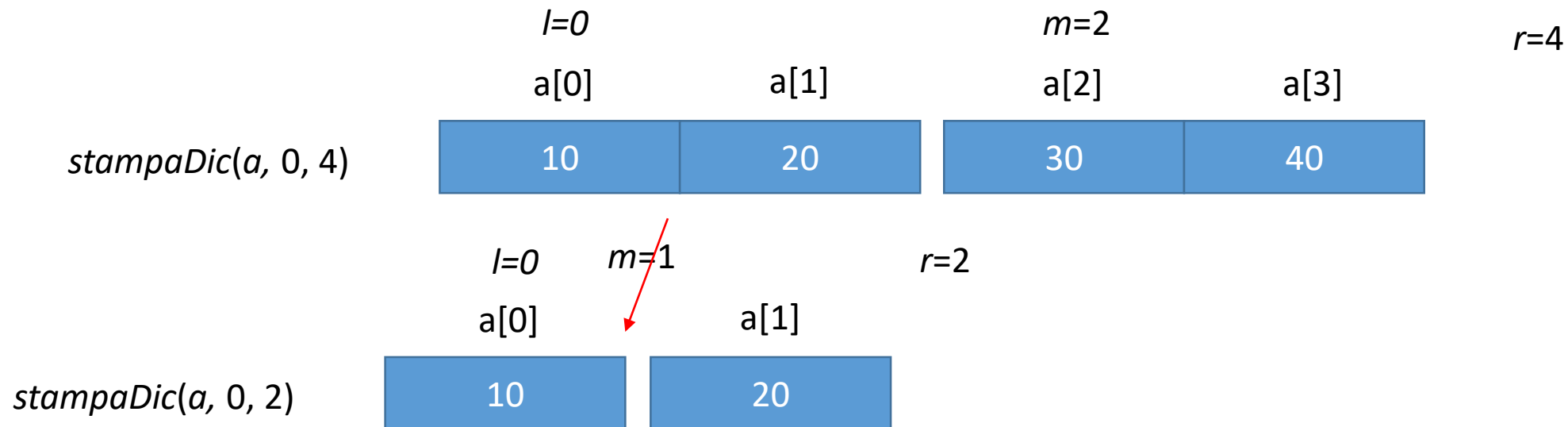
```
$java ArrayDicTest
```

# ESERCIZIO ArrayDic – stampaDic()



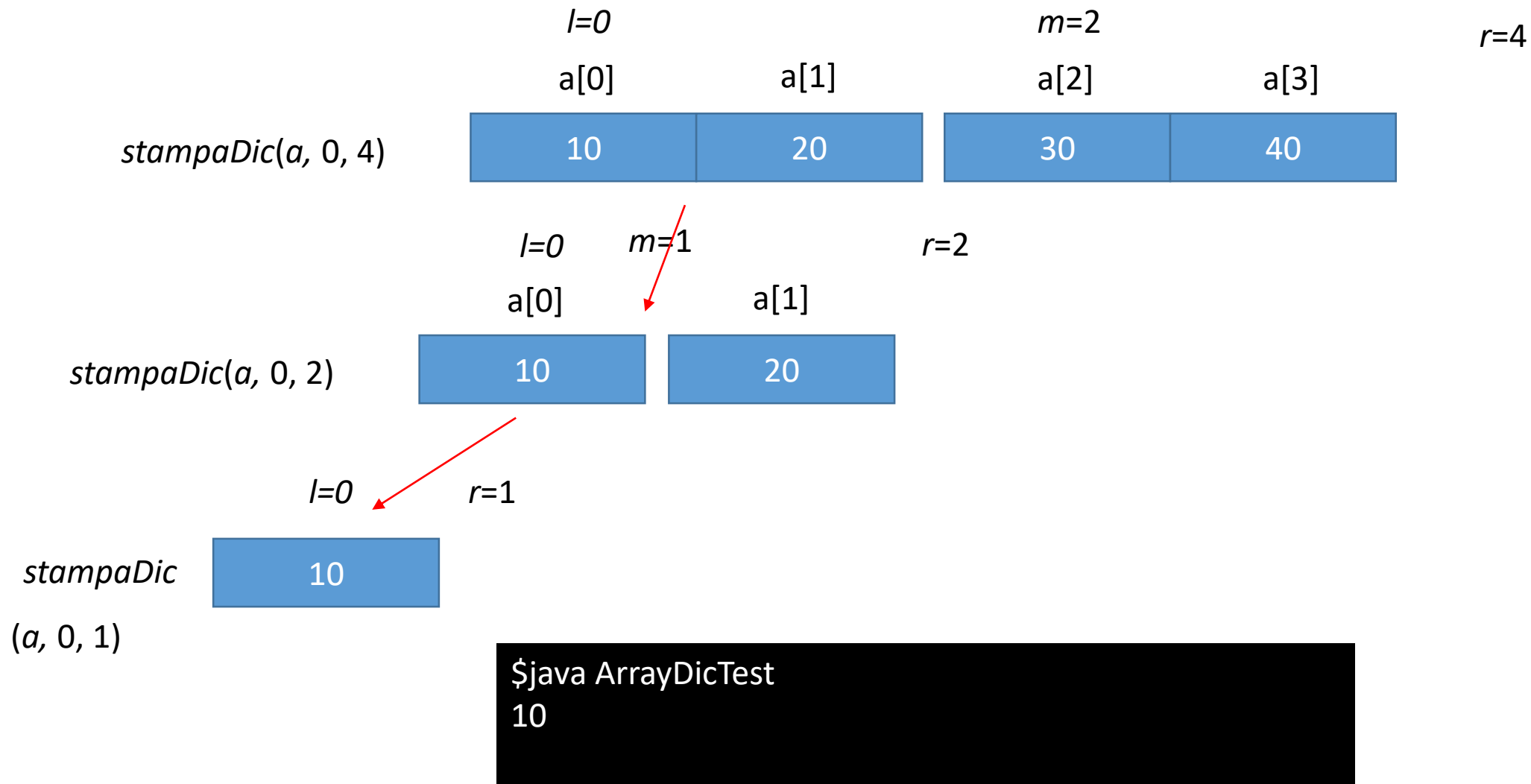
```
$java ArrayDicTest
```

# ESERCIZIO ArrayDic – stampaDic()

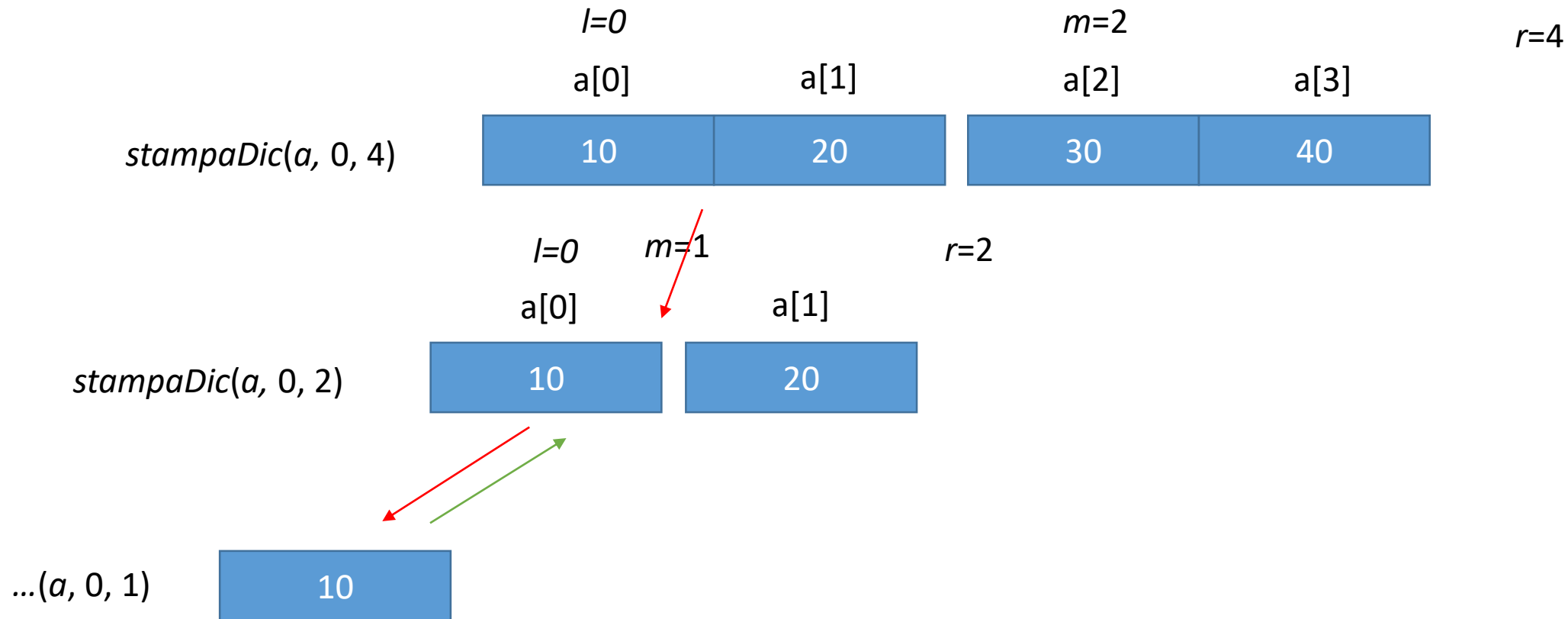


```
$java ArrayDicTest
```

# ESERCIZIO ArrayDic – stampaDic()

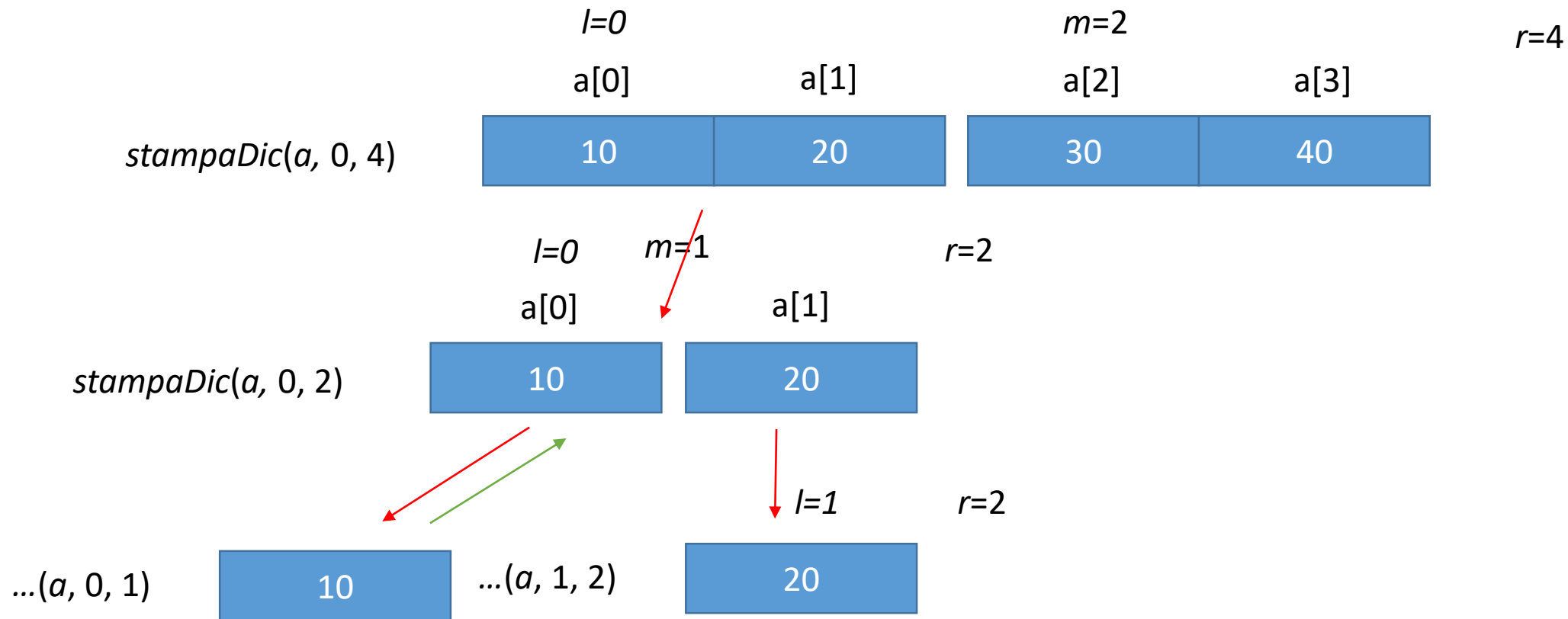


# ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest
10
```

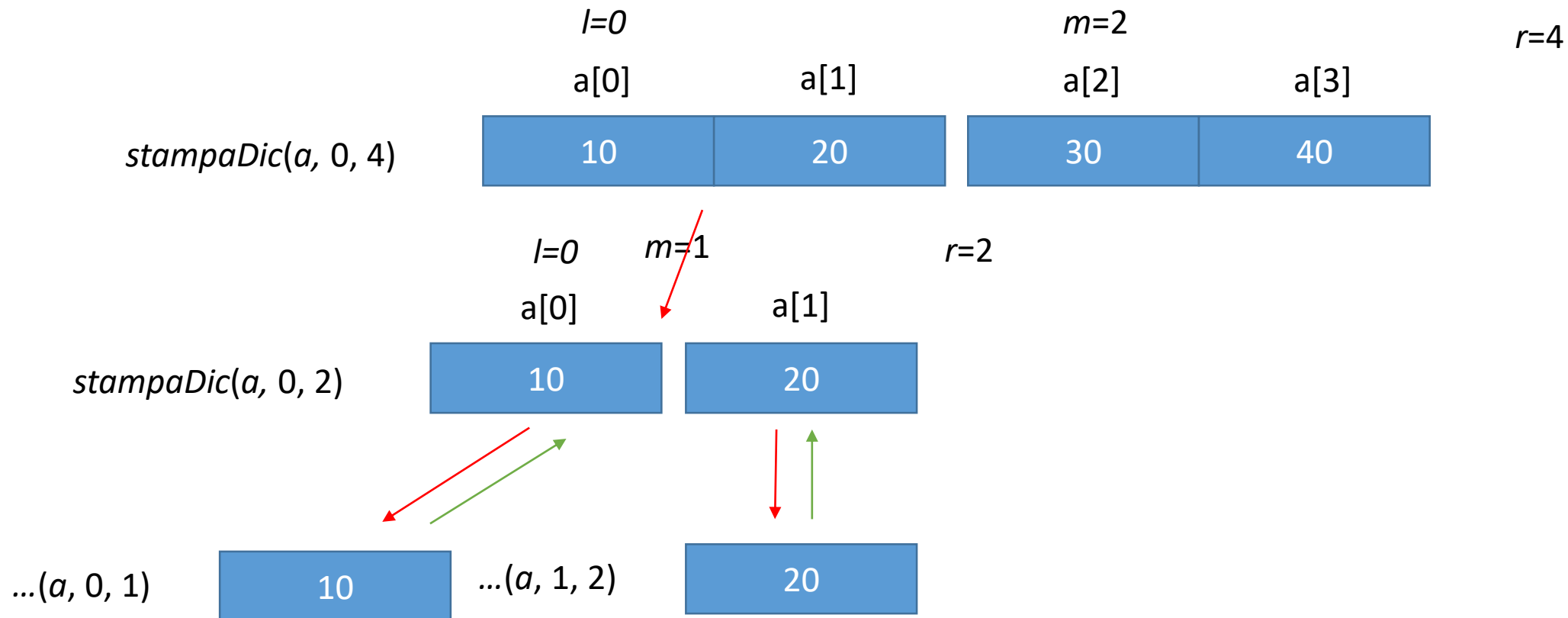
# ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest
10 20
```

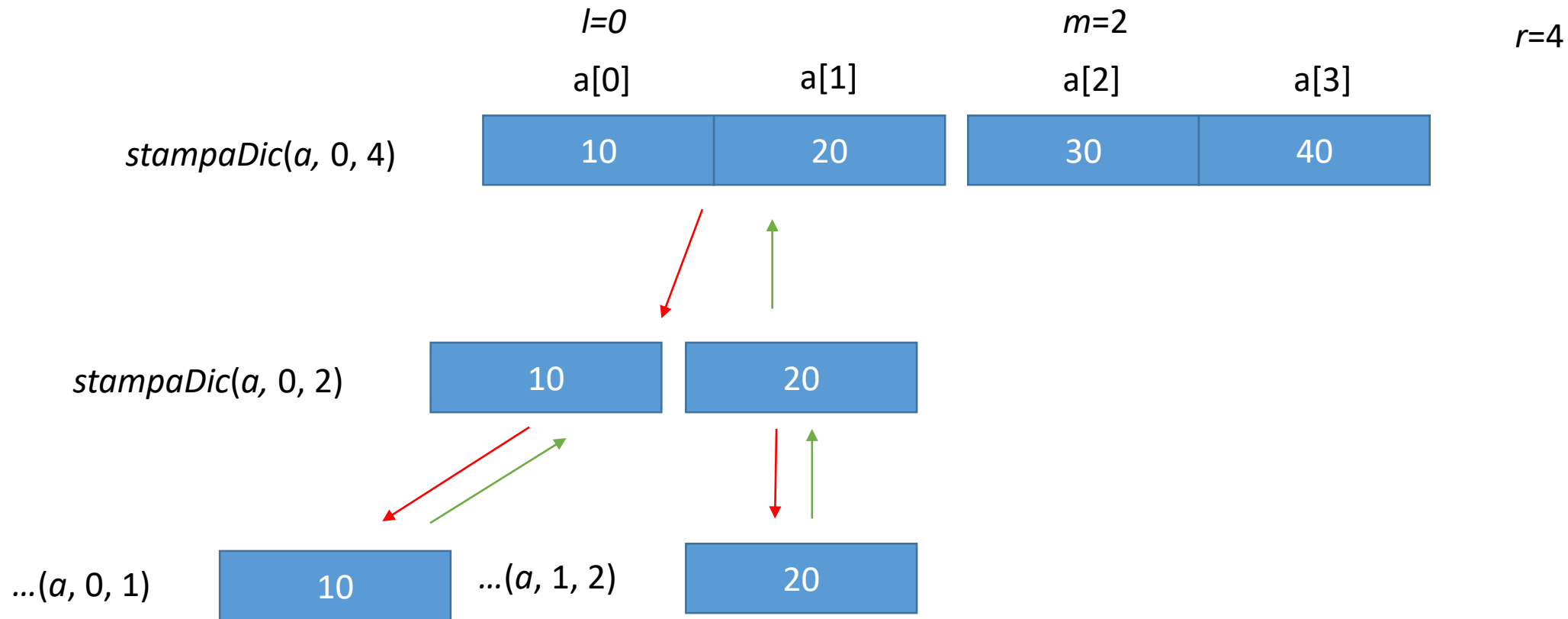


# ESERCIZIO ArrayDic – stampaDic()



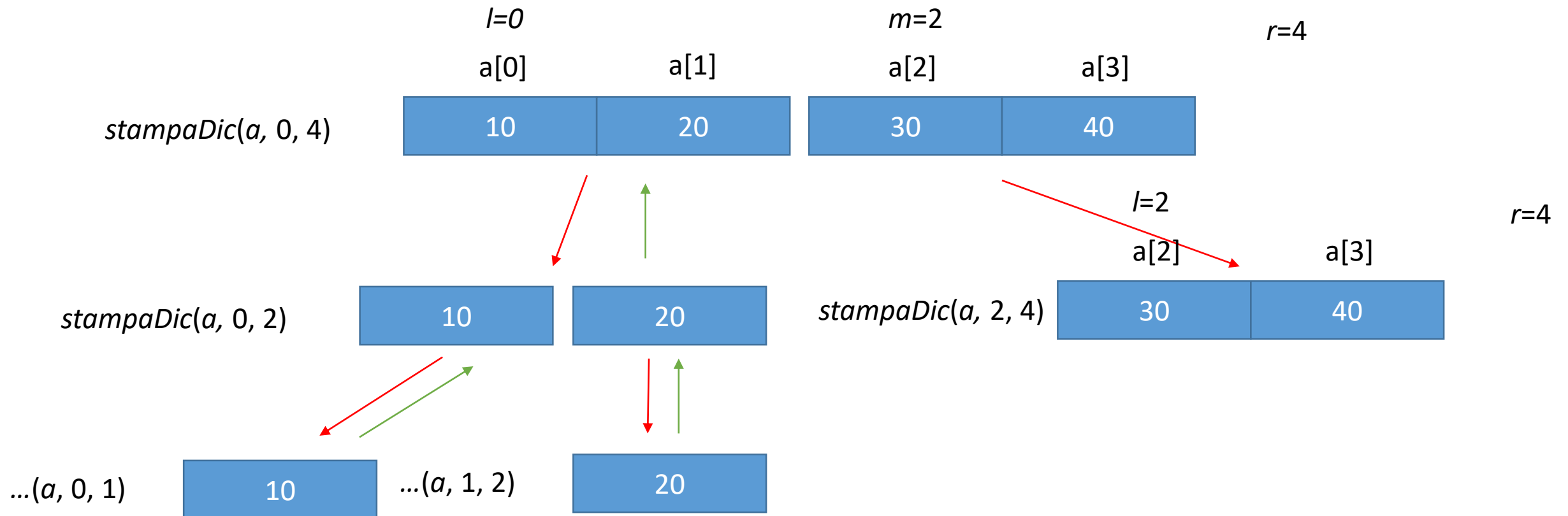
```
$java ArrayDicTest  
10 20
```

# ESERCIZIO ArrayDic – stampaDic()



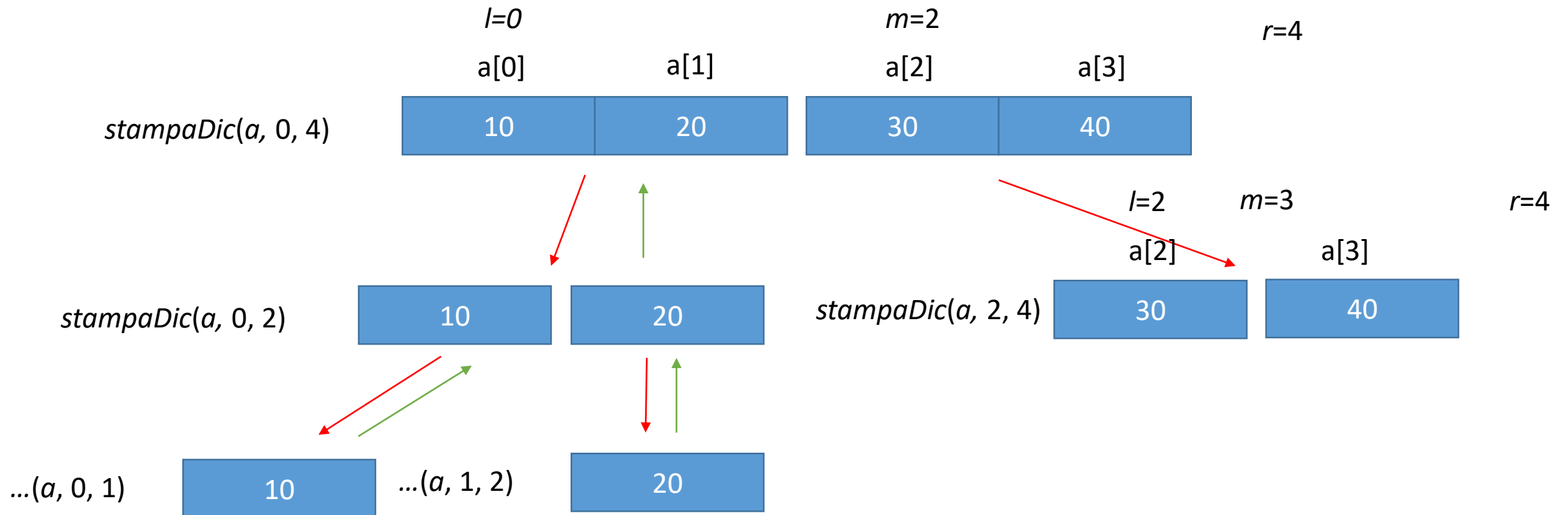
```
$java ArrayDicTest  
10 20
```

# ESERCIZIO ArrayDic – stampaDic()



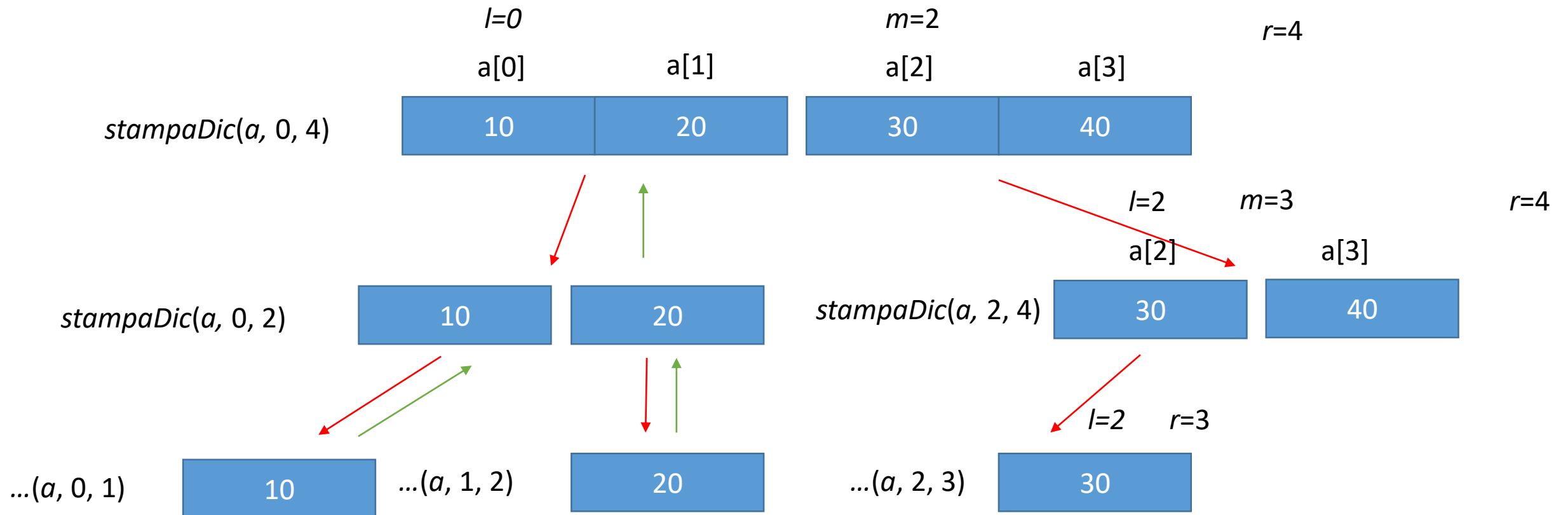
```
$java ArrayDicTest  
10 20
```

# ESERCIZIO ArrayDic – stampaDic()



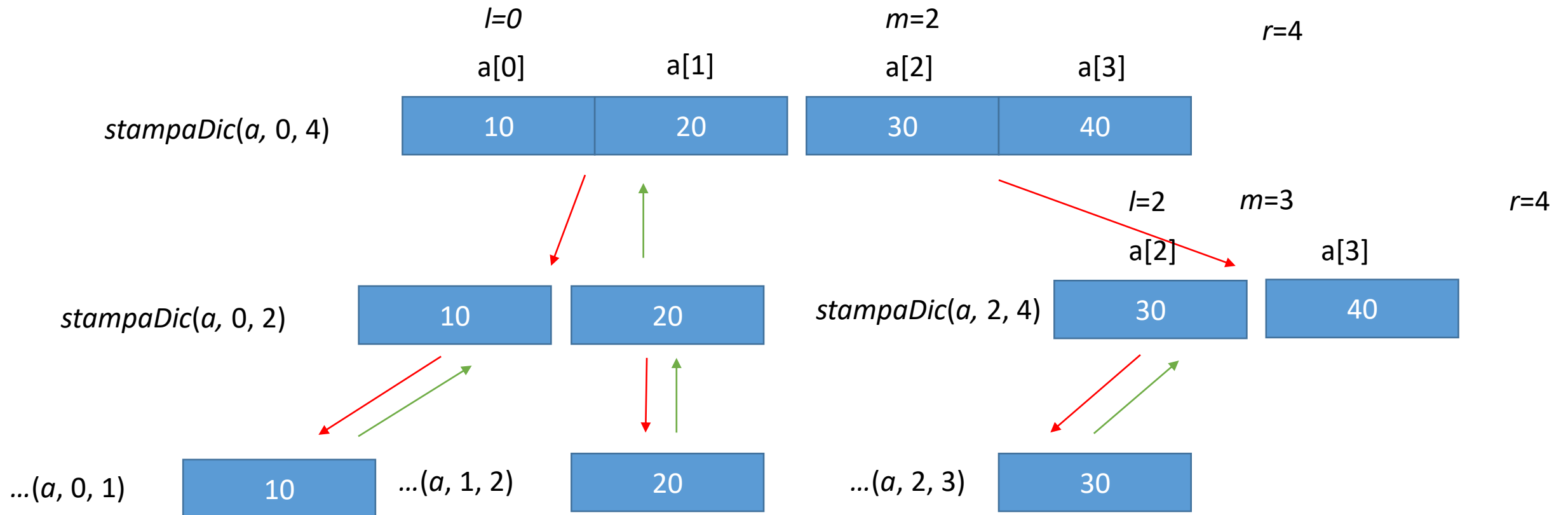
```
$java ArrayDicTest
10 20
```

# ESERCIZIO ArrayDic – stampaDic()



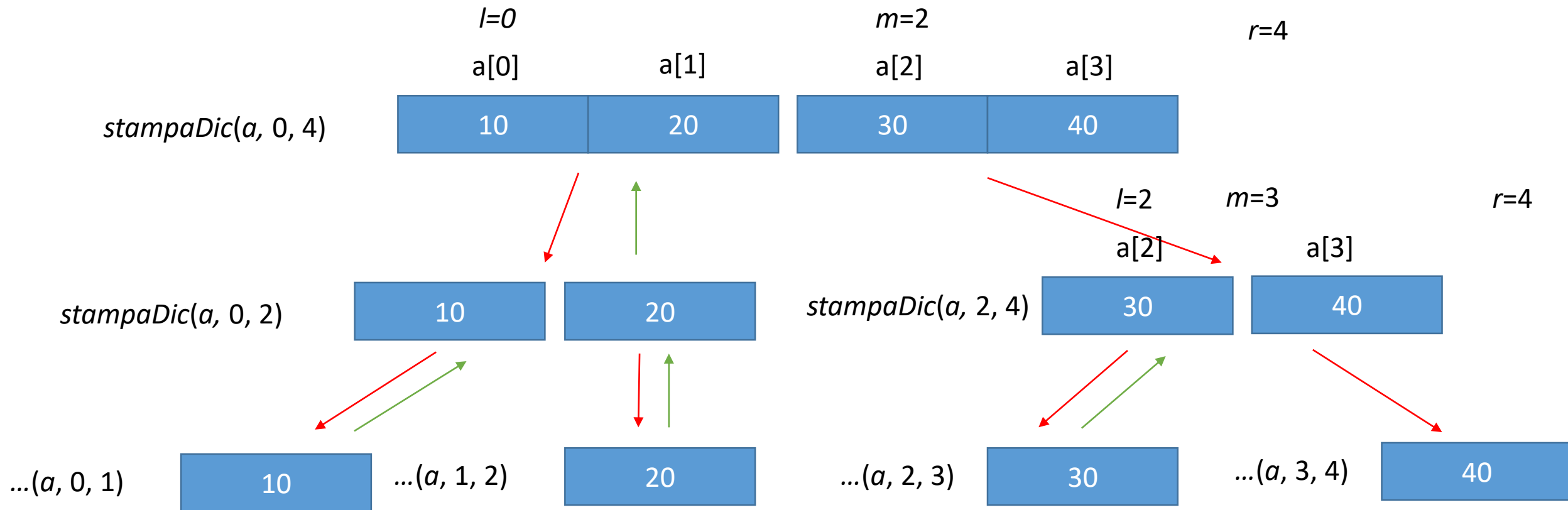
```
$java ArrayDicTest
10 20 30
```

# ESERCIZIO ArrayDic – stampaDic()



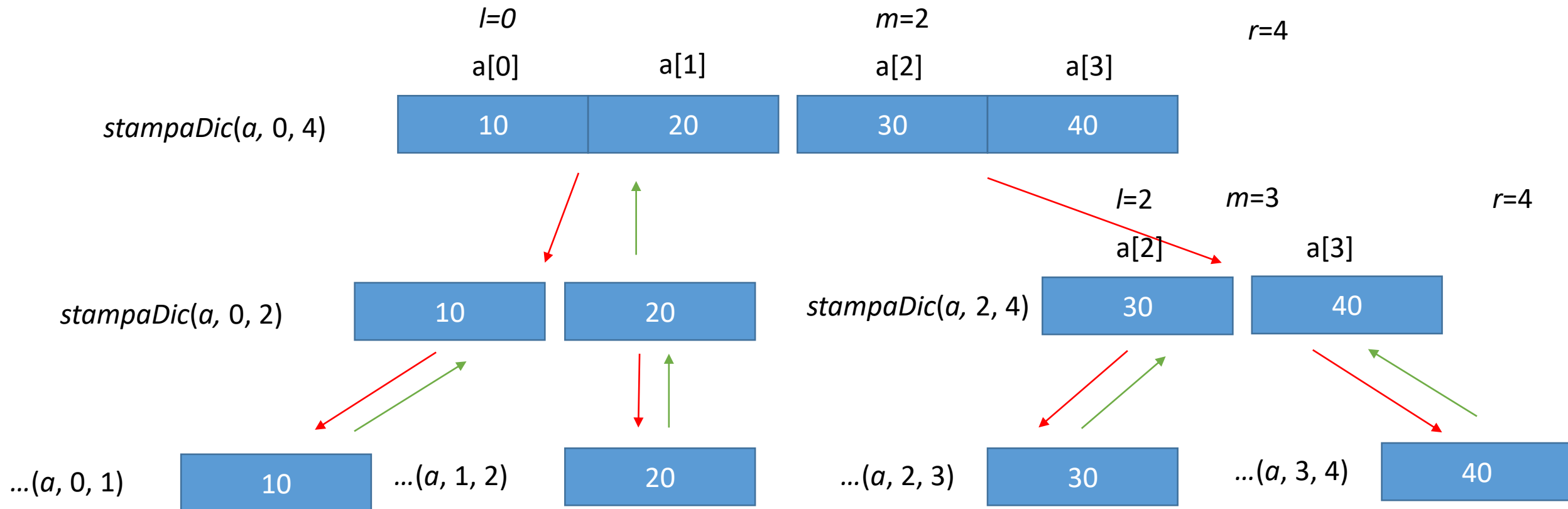
```
$java ArrayDicTest
10 20 30
```

# ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest  
10 20 30 40
```

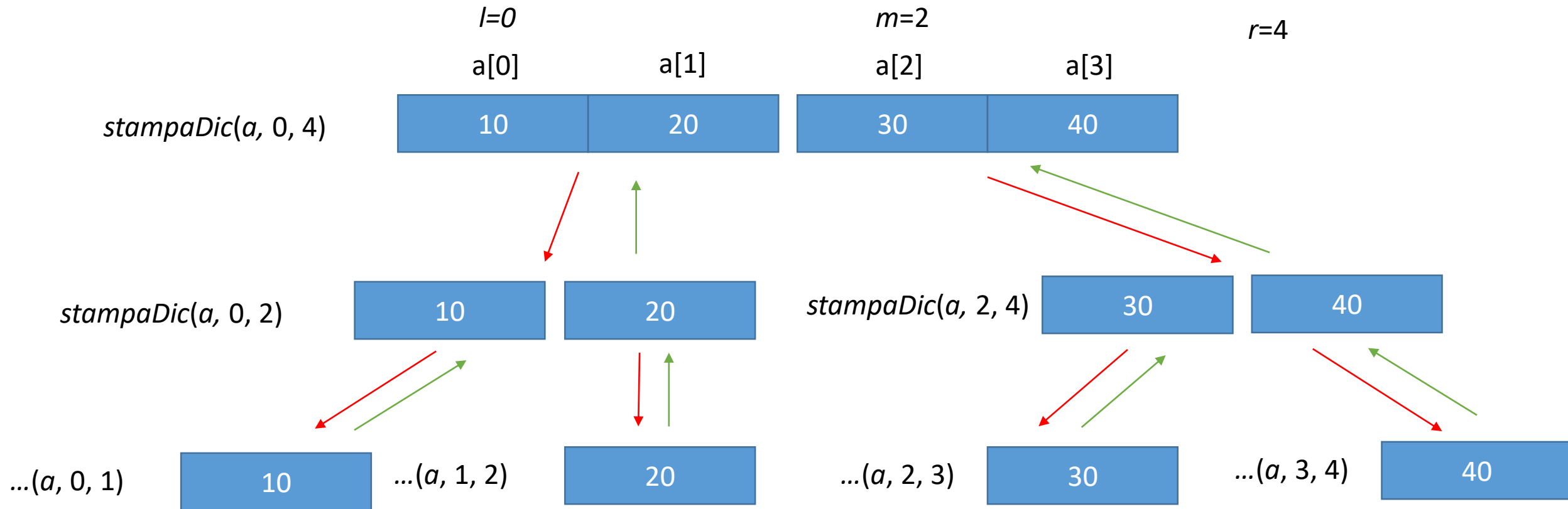
# ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest  
10 20 30 40
```

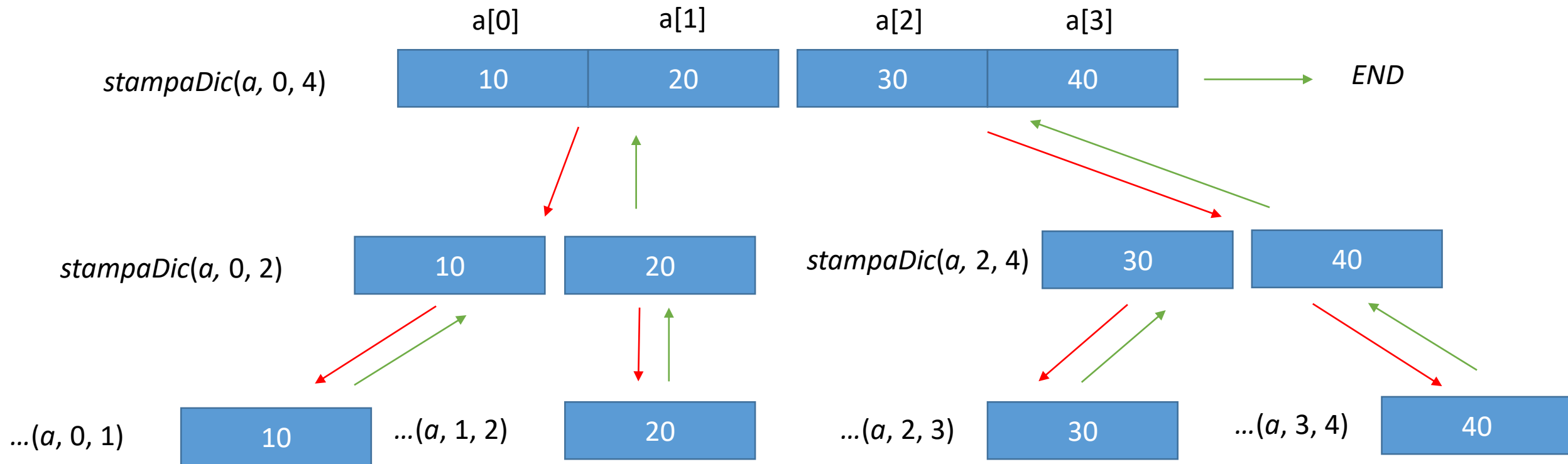


# ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest  
10 20 30 40
```

# ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest  
10 20 30 40
```

# ESERCIZIO ArrayDic – stringifyDic()

- Scrivere un metodo *stringfyDic()* che, dato un array di interi *a*, ne restituisca la sua rappresentazione in formato String, separandone gli elementi con spazi. Per esempio, dato l'array

```
int[] x = {10,20,30,40};
```

il metodo dovrà restituire (si noti lo spazio al termine della stringa)

«10 20 30 40 »

- Si verifichi che la stringa ritornata dal metodo corrisponde a quella attesa

# ESERCIZIO ArrayDic – stringifyDic()

- Signature semplificata *stringfyDic*(int[] *a*, int *l*, int *r*)
- Invocazione iniziale su [0, *a.length*)
- Se *a* contiene un solo elemento ( $l == r-1$ ), ritorno '*a[l]* + « »'
- Altrimenti, calcolo  $m = (l + r) / 2$  e ricorro dicotomicamente:
  1. ricorsione sull'intervallo [*l*, *m*) di *a* -> *stringfyDic*(*a*, *l*, *m*)
  2. ricorsione sull'intervallo [*m*, *r*) di *a* -> *stringfyDic*(*a*, *m*, *r*)

# ESERCIZIO ArrayDic – stringifyDic()

```
public static String stringifyDic(int[] a, String s, int l, int r) {
    // Caso base
    if (l+1 == r){
        s = a[l] + " ";
    }
    else {
        // Calcolo l'elemento in posizione centrale per l ed r *correnti*
        int m = (r+1)/2;
        // Esploro il sotto-array sinistro [l, m) e poi il sotto-array destro [m, r)
        s = s + stringifyDic(a, s, l, m) + stringifyDic(a, s, m, r);
    }

    return s;
}

public static String stringifyDic(int [] a) {
    String ret = "";

    if (a != null) {
        ret = stringifyDic(a, ret, 0, a.length);
    }

    return ret;
}
```

# Es1 – raddoppioPari()

Scrivere un metodo dicotomico ricorsivo *raddoppioPari()* tale che:

- ha un array di interi *a* ed un intero *v* come parametri;
- modifica *a* sfruttando l'aliasing in modo che ogni elemento di posizione pari **e** maggiore di *v* sia raddoppiato
- l'array *a* può essere null o contenere 0 elementi
- disponga dell'opportuno metodo wrapper

# Es1 – raddoppioPari()

```
public static void raddoppioPari(int[] a, int v, int l, int r) {
    // Caso elementare: ho un (sotto)array a di un solo elemento
    if (r == l+1) {
        if ((l%2 == 0) && (l >= v)) {
            a[l] = 2 * a[l];
        }
    }
    // Caso generale: il (sotto)array a ha almeno due elementi
    else {
        int m = (r+l)/2;
        raddoppioPari(a, v, l, m);
        raddoppioPari(a, v, m, r);
    }
    // Esplicitato per ragioni didattiche
    return;
}

public static int[] raddoppioPari(int[] a, int v) {
    int[] b = null;

    if (a != null) {
        b = Arrays.copyOf(a, a.length);
        if (a.length > 0) {
            raddoppioPari(b, v, 0 /* l */, a.length /* r */);
        }
    }

    return b;
}
```

## Es2 – raddoppioPariCnt()

Scrivere un metodo dicotomico ricorsivo *raddoppioPariCnt()* che:

- ha un array di interi  $a$  ed un intero  $v$  come parametri;
- modifica  $a$  sfruttando l'aliasing in modo tale ogni elemento di posizione pari e maggiore di  $v$  sia raddoppiato
- ritorna il numero di sostituzioni effettuate in  $a$
- disponga dell'opportuno metodo wrapper



## Es2 – raddoppioPariCnt()

```
public static int raddoppioPariCnt(int[] a, int v, int l, int r) {
    // Contatore di sostituzioni
    int cnt = 0;
    // Caso base: ho un (sotto)array a di un solo elemento
    if (r-l == 1) {
        if ((l%2 == 0) && (l >= v)) {
            a[l] = 2 * a[l];
            cnt = cnt + 1;
        }
        return cnt;
    }
    // Caso generale: il (sotto)array a ha almeno due elementi, procedo
    else {
        int m = (r+1)/2;
        int sostLeft = raddoppioPariCnt(a, v, l, m);
        int sostRight = raddoppioPariCnt(a, v, m, r);
        cnt = sostLeft + sostRight;
        return cnt;
    }
}
```

```
public static int raddoppioPariCnt(int[] a, int v) {
    int cnt = 0;
    if (a != null) {
        cnt = raddoppioPariCnt(a, v, 0, a.length);
    }
    return cnt;
}
```

## Es3 – copiaPrecedente()

Scrivere un metodo ricorsivo dicotomico `copiaPrecedente()` che:

- ha un array di interi  $a$  come parametro
- restituisce  $a$ , in cui ogni elemento con valore pari è ricoperto dal valore dell'elemento che lo precede, se esiste
- disponga dell'opportuno metodo wrapper

# Es3 – copiaPrecedente()

```
public static void copiaPrecedente(int[] a, int l, int r) {
    if (r-l == 1) {
        // La prima condizione di raddoppio "a[l] ha un predecessore" é in AND logico con "a[l]%2 == 0"
        if((l >= 1) && (l % 2 == 0)) {
            a[l] = a[l-1];
        }
        return;
    }
    else {
        int m = (r+1)/2;
        copiaPrecedente(a, l, m);
        copiaPrecedente(a, m, r);
        return;
    }
}

public static int[] copiaPrecedente(int[] a) {
    int[] b = null;
    if (a != null) {
        b = Arrays.copyOf(a, a.length);
        copiaPrecedente(b, 0, b.length);
    }
    return b;
}
```

## Es4 – `scambiaSuccessivoCov()`

Scrivere un metodo ricorsivo **covariante** `scambiaSuccessivoCov()` che:

- ha un array di interi  $a$  ed un intero  $v$  come parametri
- restituisce  $a$ , in cui ogni elemento con valore maggiore di  $v$  è scambiato con l'elemento immediatamente successivo, se esiste
- disponga dell'opportuno metodo wrapper

## Es4 – scambiaSuccessivoCov()

```
public static void scambiaSuccessivoCov(int[] a, int v, int i) {
    //Notare che l'ultimo elemento valido dell'array non può avere successori
    if (i == a.length - 1) {
        return;
    }
    else {
        scambiaSuccessivoCov(a, v, i+1);
        if((a[i] > v) && (i+1 < a.length)) {
            int tmp = a[i];
            a[i] = a[i+1];
            a[i+1] = tmp;
        }
        return;
    }
}

public static int[] scambiaSuccessivoCov(int[] a, int v) {
    int[] b = null;
    if (a != null) {
        b = Arrays.copyOf(a, a.length);
        scambiaSuccessivoCov(b, v, 0);
    }
    return b;
}
```

## Es4 – `scambiaSuccessivoDic()`

Scrivere un metodo ricorsivo **dicotomico** `scambiaSuccessivoDic()` che:

- ha un array di interi  $a$  ed un intero  $v$  come parametri
- restituisce  $a$ , in cui ogni elemento con valore maggiore di  $v$  è scambiato con l'elemento immediatamente successivo, se esiste
- disponga dell'opportuno metodo wrapper

## Es4 – scambiaSuccessivoDic()

```
public static void scambiaSuccessivoDic(int[] a, int v, int l, int r) {
    if (r-l == 1) {
        if((l+1 < a.length) && (a[l] > v)) {
            int tmp = a[l];
            a[l] = a[l+1];
            a[l+1] = tmp;
        }
        return;
    }
    else {
        int m = (r+l)/2;
        scambiaSuccessivoDic(a, v, l, m);
        scambiaSuccessivoDic(a, v, m, r);
        return;
    }
}

public static int[] scambiaSuccessivoDic(int[] a, int v) {
    int[] b = null;
    if (a != null) {
        int[] b = Arrays.copyOf(a, a.length);
        scambiaSuccessivoDic(b, v, 0, b.length);
    }
    return b;
}
```