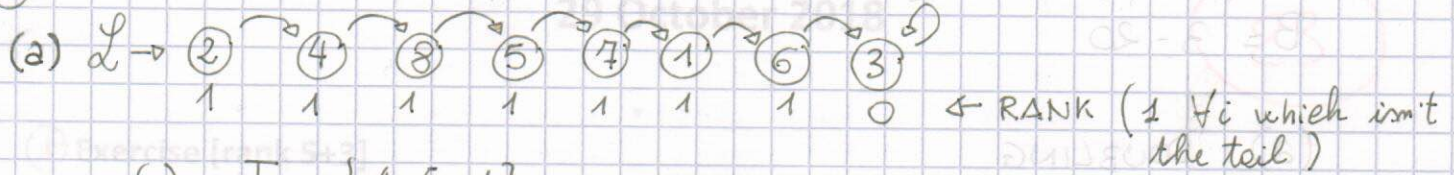
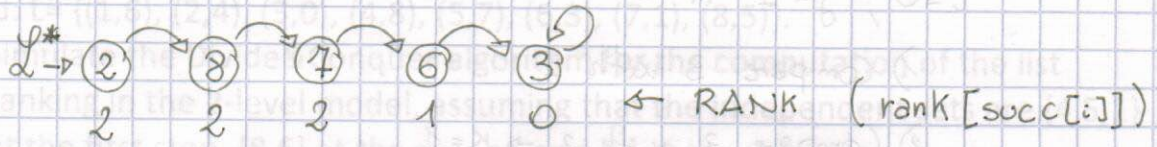


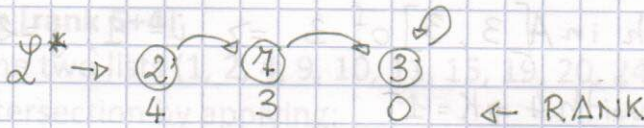
①



(1) $I = \{4, 5, 1\}$



(2) $I = \{8, 6\}$



(3) $I = \{7\}$



At this step we may start recombining

(1)

1	2	3	4	5	6	7	8
1	7	0	1	1	1	3	2

(2)

1	2	3	4	5	6	7	8
1	7	0	1	1	1	3	2

RANK += RANK(SUCC[i])
update rank of 7

(3)

1	2	3	4	5	6	7	8
1	7	0	1	1	1	3	5

RANK += RANK(SUCC[i])
update rank of 8 and 6

(4)

1	2	3	4	5	6	7	8
2	7	0	6	4	1	3	5

RANK += RANK(SUCC[i])
update rank of 4, 5 and 1

(b) $O\left(\frac{m}{B} \cdot \log_{\frac{m}{B}} \frac{m}{H} \cdot \log^* m\right) = \tilde{O}(m/B)$

OK

② $A = 1 - 2 - 4 - 9 - 10 - 11 - 15 - 19 - 20 - 24$

$B = 3 - 20$

(a) DOUBLING

$i = 0, j = 1, K = 0$

- 1) Compare 3 with 1
- 2) Compare 3 with 2 $\Rightarrow K = 1$
- 3) Compare 3 with 9 $\Rightarrow K = 2$
- 4) Binary search in $A[3, 3]$ of 3 $\Rightarrow i = 1, j = 2, K = 0$
- 5) Compare 20 with 4 $\Rightarrow K = 1$
- 6) Compare 20 with 9 $\Rightarrow K = 2$
- 7) Compare 20 with 11 $\Rightarrow K = 3$
- 8) Compare 20 with 20 \Rightarrow print 20. Stop.

(b) SHUFFLING

- 1) Choose permutation $\pi(x) = 2x + 3 \pmod{29}$

$\pi(A) = 5 - 7 - 11 - 21 - 23 - 25 - 4 - 12 - 14 - 22$

$\pi(B) = 9 - 14$

- 2) Sort $\pi(A) \Rightarrow A_s = 4 - 5 - 7 - 11 - 12 - 14 - 21 - 22 - 23 - 25$

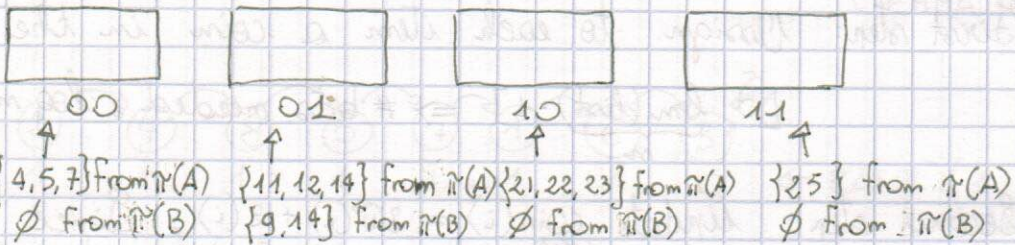
Sort $\pi(B) \Rightarrow B_s = 9 - 14$

- 3) Write A_s and B_s in base 2:

$A_s =$	00100 (4)	$B_s =$	01001 (9)
	00101 (5)		01110 (14)
	00111 (7)		
	01011 (11)		
	01100 (12)		
	01110 (14)		
	10101 (21)		
	10110 (22)		
	10111 (23)		
	11001 (25)		

- 4) # of buckets = 4, since $L = 3$

Take the 2 most significant digits of the elements to assign the bucket to each of them



5) Compute merge based intersection between $\pi(A)_2, \pi(B)_2 = \{14\}$

6) Use $\pi^{-1}(x)$ to find the original value: $\pi^{-1}(x) = 15x + 13$

$$\pi^{-1}(14) = \textcircled{20}$$

3) Idea: mimic the behaviour of BOUNDED QUICKSORT, to eliminate tail recursion

③ Per limitare lo spazio a $O(\log m)$ si può utilizzare la medesima idea del bounded quicksort:

se la sequenza in cui si dovrebbe ripetere la ricorsione dovesse essere più lunga della metà dell'array (quindi $\frac{i+j}{2}$) bisogna evitare la ricorsione in quanto risulterebbe sbilanciata e optare per un processo iterativo.

```
randSelect(A, i, j, k) {  
  while (j - i > 0) {
```

```
    r = random pickUna good pivot
```

```
    S< = items of A < of A[r]
```

```
    S> = items of A > of A[r]
```

```
    M< = |S<|
```

```
    M= = |A| - |S<| - |S>|
```

```
    if (k ≤ M<) {
```

```
      if (M< >  $\frac{i+j}{2}$ ) // la parte sinistra  
                          è troppo lunga per la  
                          ricorsione  
        j = i + M<
```

```
      else
```

```
        return randSelect(A, i, M<, k)
```

```
    }
```

```
    else if (k ≤ (M< + M=))
```

```
      return A[r]
```

OK bene!

```
    else {
```

```
      if (M> >  $\frac{i+j}{2}$ ) // parte destra troppo  
                          lunga per ricorsione  
        i = i + M< + M=
```

```
      else
```

```
        return randSelect(A, i + M< + M=,  
                          k - M< - M=)
```

```
    }
```

```
  }
```

} In questo modo si ottengono ricorsioni bilanciate al più $O(\log m)$