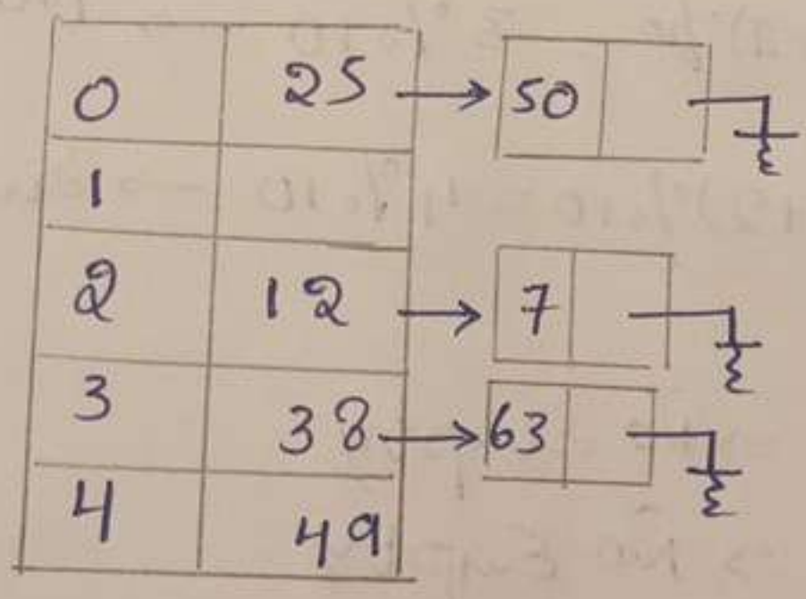
1. Given a hash table size 5 & a hash function x%5, insert the following keys into the hash table using seperate chaining to handle collisions:
12, 25, 38, 49, 50, 63, 7.

Sol.

* Given Table size 5
* Given Hash function x%5

→ 12%5 = 2
→ 25%5 = 0
→ 38%5 = 3
→ 49%5 = 4
→ 50%5 = 0
→ 63%5 = 3
→ 7%5 = 2

Hash Table:



j = 25, j > 18
j = 10, j < 18
Here i, j are co

| 10 | 18 | 25 | 27 | 36 | 45 |

2. Given hash table of size 10 & the following keys: 12, 18, 13, 2, 3, 23, 5, 15. Use the division method of hashing & linear probing to handle collisions. Show the final state of the hash table.

Sol. Given table size = 10

Elements = 12, 18, 13, 2, 3, 23, 5, 15

1) Insert '12', index = 12 % 10 = 2, i = 0

Index = (2+0) % 10 = 2 → Empty

2) Insert '18', 18 % 10 = 8, i = 0

Index = (8+0) % 10 = 8 → Empty

3) Insert '13', 13 % 10 = 3, i = 0

Index = (3+0) % 10 = 3 → Empty

4) Insert '2', 2 % 10 = 2, i = 0

Index = 2 → No empty

i = 1

(2+1) % 10 = 3 % 10 → No Empty

i = 2

(2+2) % 10 = 4 % 10 → Empty

5) Insert '3'

i = 0 => No Empty

i = 1 => No Empty

i = 2 => (3+2) % 10 = 5 → Empty

6) Insert '23'

23 % 10 = 3

i = 0 => No Empty

i = 1 => No Empty

i = 2 => No Empty

i = 3 => 6 % 10 => Empty

j = 10, j

Here i, j are

10	18	25	27	36	45

↓ sorted

7) Insert '5' , 5%10 = 5 => No Empty

i=0 => No Empty

i=1 => No Empty

i=2 => (5+2) %10 = 7 → Empty

8) Insert '25' , 25%10 = 5

i=0 => No Empty

i=1 => No Empty

i=2 => No Empty

i=3 => (5+3)%10 = 8 → NOEmpty

i=4 => 9%10 = 9 -> Empty

After '10'	18	13	2	3	23	5	15
0							
1							
2	12	12	12	12	12	12	12
3		13	13	13	13	13	13
4			2	2	2	2	2
5				3	3	3	3
6					23	23	23
7						5	5
8		18	18	18	18	18	18
9							15
10							

10	18	25	27	36	45

↓ Sorted

j=10, j<18, stop

Here i, j are corssed

3) a) Implement a Bubble Sort algorithm with an optimization to reduce the number of comparisons in cases where array is partially sorted.

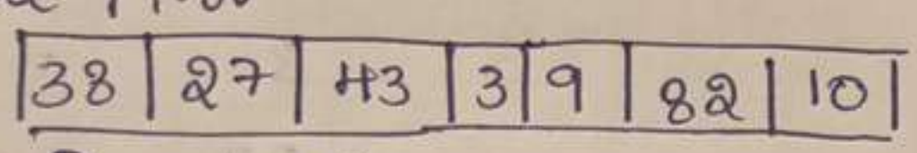
Sol.

Bubble sort is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items, & swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. An optimized version of Bubble Sort can reduce the number of comparisons when the array is partially sorted by adding a flag to monitor if any swaps were made during a pass. If no swaps were made, the array is already sorted, & we can exit easily.

3) b) Sort the following [38, 27, 43, 3, 9, 82, 10] using Quick sort.

Sol.

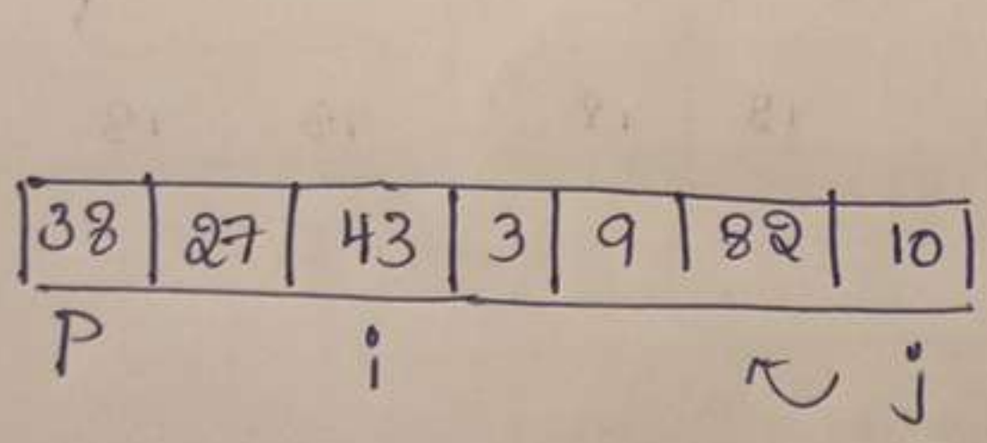
Choose Pivot



Compare P & i

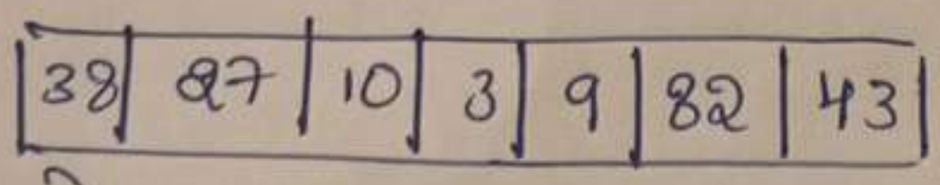
i=27, 27<P, increment

i=43, 43>P, stop



compare P & j

j=10, j<P, stop

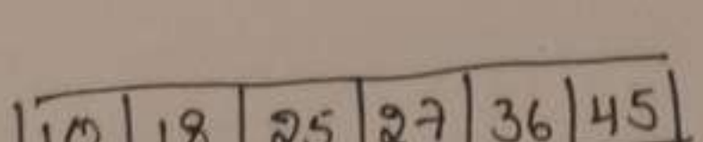


swap i & j

j=25, j>18, "

j=10, j<18, stop

Here i, j are crossed



compare P&j

j=45, j>P,

j=36, 36>P, decrement

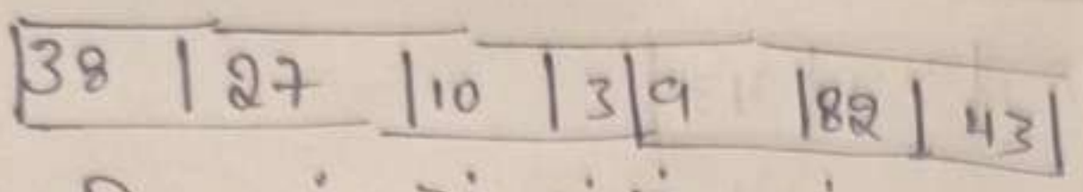
j=19, 19<27, stop

(i crossed j)

here i & j

swap P&j

partition result



Compare i&P

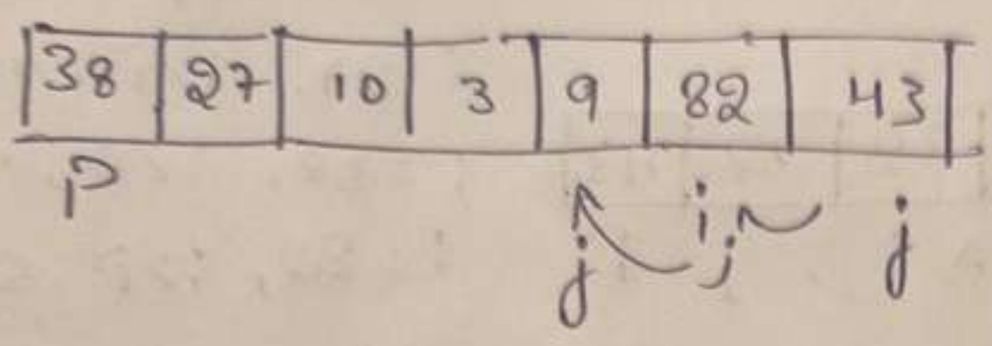
i=27, 27<P, increment

i=10, 10<P, increment

i=3, 3<P, increment

i=9, 9<P, increment

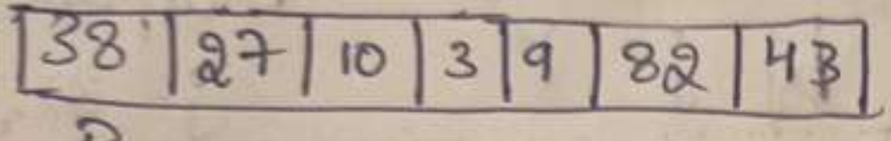
i=82, 82>P, stop



j=43, j>P, decrement

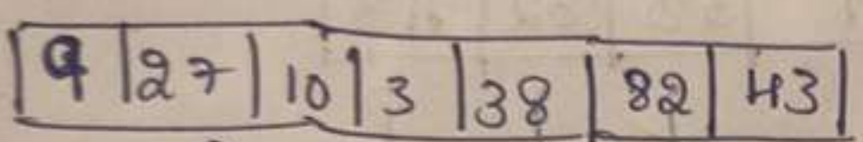
j=82, j>P, decrement

j=9, j<P, stop

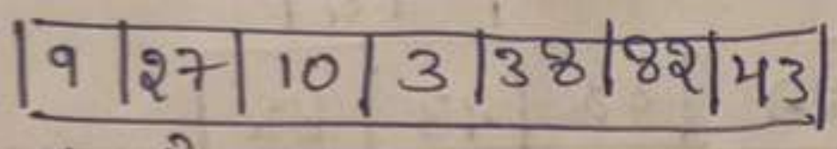


i x j

Swap j&P



Now choose New P



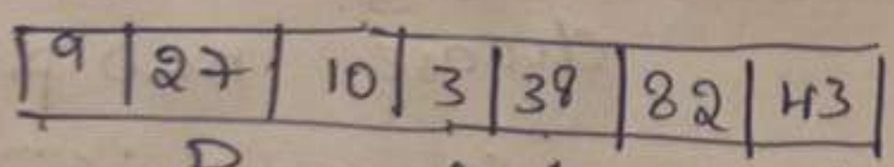
Compare P&i

i=9, i<P, increment

i=10, i<P, "

i=3, i<P, "

i=38, i>P, stop

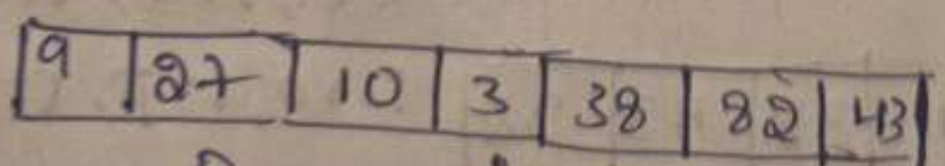


compare j&P

j=43, j>P, decrem

j=82, j>P, "

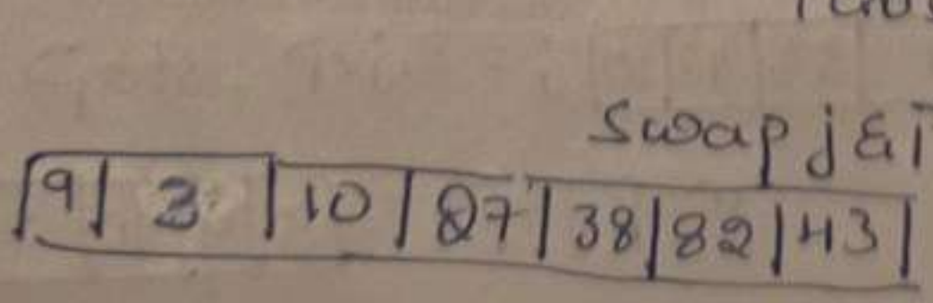
j=3, j<P. stop



i x j

i crossed j

Swap j&P



partition result

element as pivot

9	3	10	27	38	82	43

choose new p

9	3	10	27	38	82	43

i P

compare i & P

i = 9, i < P, increment

i = 3, i < P, "

i = 10, i < P, "

i = 27, i < P, "

i = 38, i < P, "

i = 82, i > P, stop

9	3	10	27	38	82	43

i j P

9	3	10	27	38	82	43

i=j j P

Compare j & P

j = 82, j > P, decr

j = 38, j < P, stop

9	3	10	27	38	82	43

j i P

swap i & P

9	3	10	27	38	43	82

9	3	10	27	38	43	82

P i → i

choose new pivot

compare i & P

i = 3, i < P, increment

i = 10, i > P, stop

9	3	10	27	38	43	82

P ↖ i j

compare j & P

j = 27, j > P, decrement

9	3	10	27	38	43	82

P j i

j = 3, j < P, stop

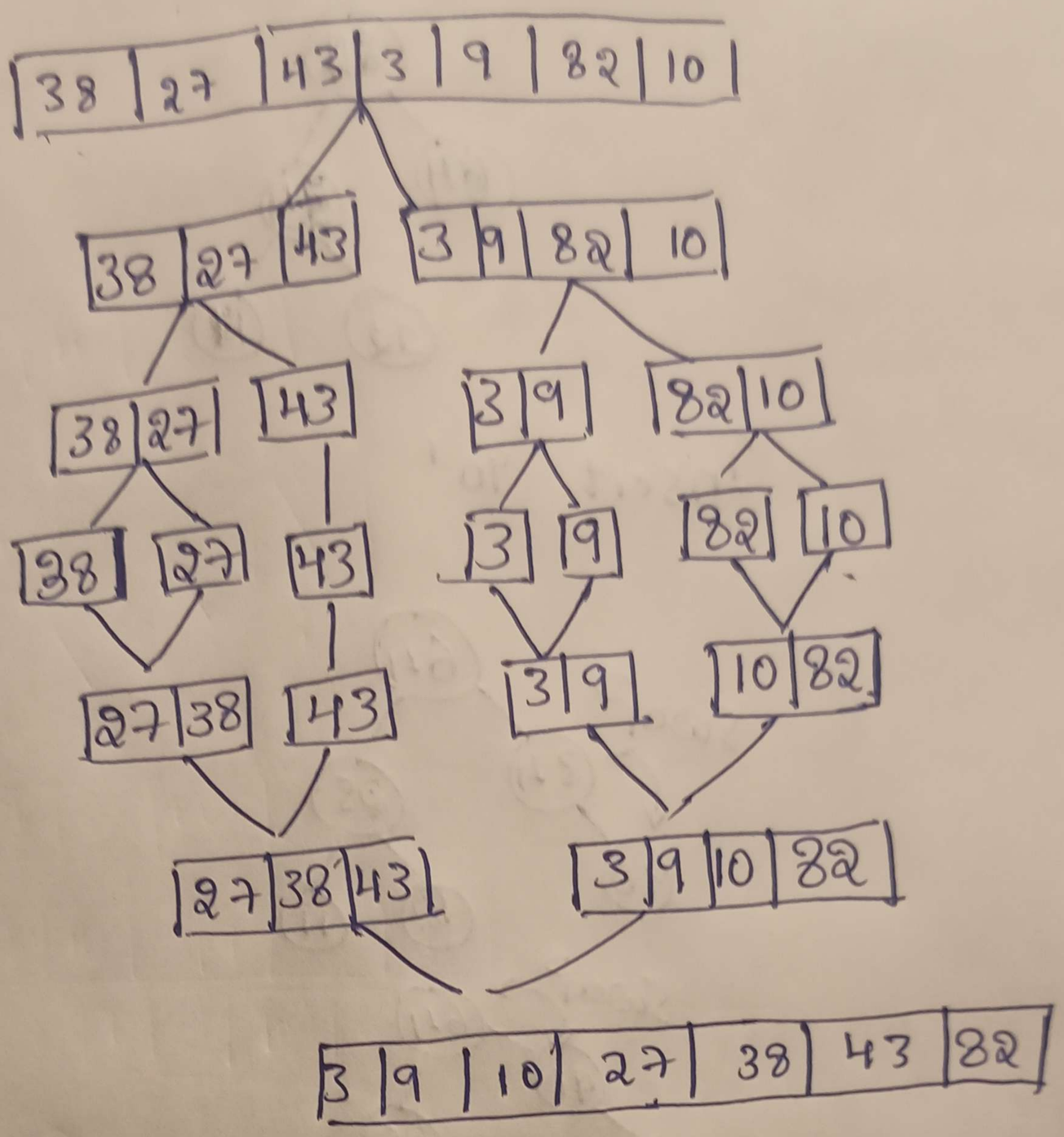
swap j & P

Sorted

3	9	10	27	38	43	82

4. a) Sort the array [38, 27, 43, 3, 82, 10] using Merge Sort.

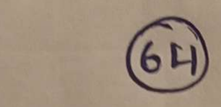
Sol.



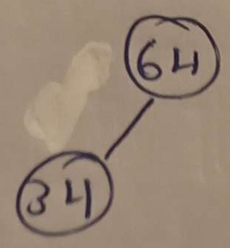
b) Sort the array using Heap sort [64, 34, 25, 12, 11, 90]

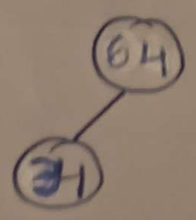
Sol.

Insert '64'

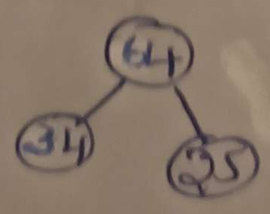


Insert '34'

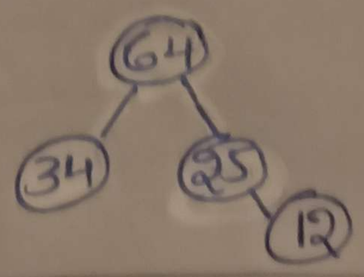




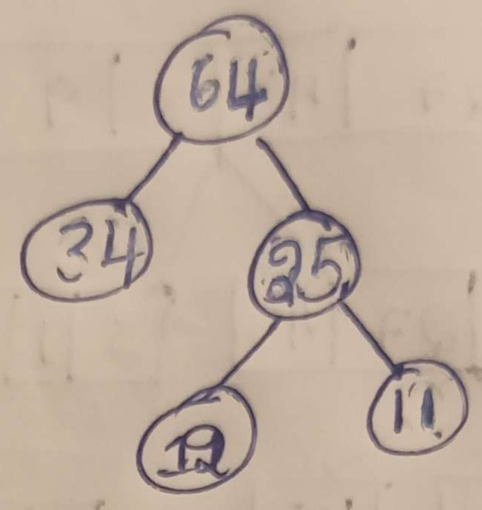
Insert '25'



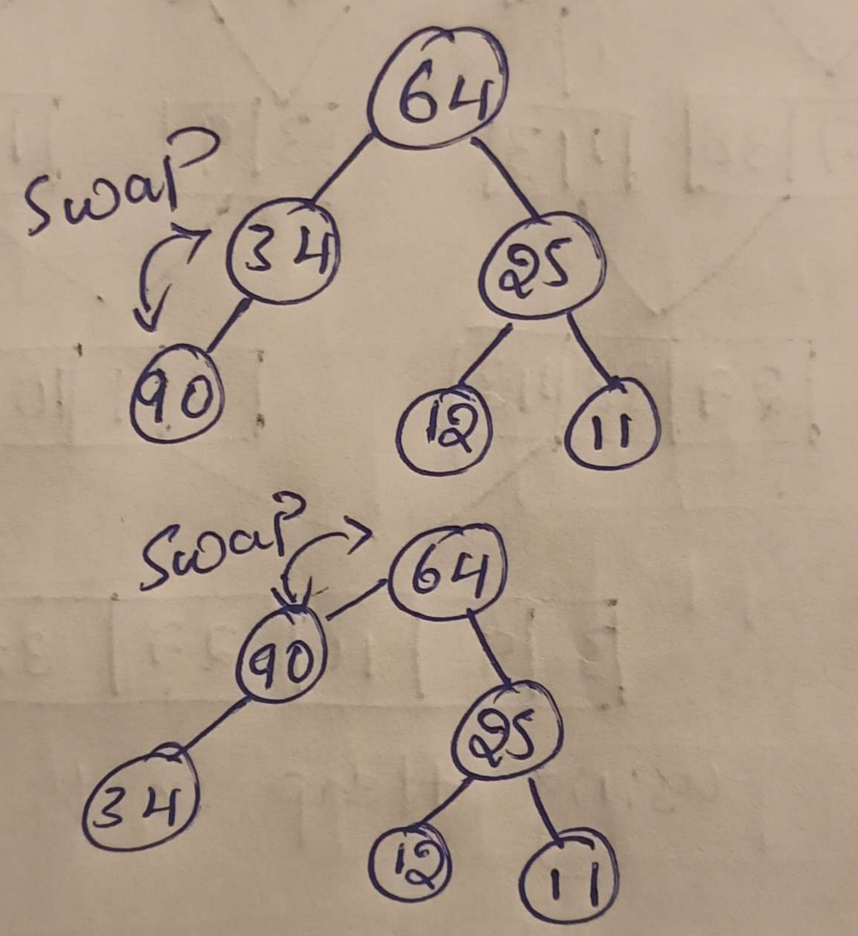
Insert '12'



Insert '12'



Insert '90'



Final Sort