Fu Xi 1155029172 WA 4

4.1

(1)

0	1	2	3	4	5	6	7	8	9
	4371		1323	4344					4199
			6173						9679
									1989

(2)

0	1	2	3	4	5	6	7	8	9
~~4371~~			~~1323~~	~~6173~~	~~9679~~		~~4344~~		~~4199~~
9679	4371	1989	1323	6173	4344				4199

(3)

0	1	2	3	4	5	6	7	8	9
9679	4371		1323	6173	4344			1989	4199

(4)

0	1	2	3	4	5	6	7	8	9
	4371,		1323,	6173,	9679		4344,		4199

(1989 failed)

(1): Fast Find operation, simple and efficient ~~but can't solve collision~~, Links require space, really deletion

(2) Can avoid collision, faster than other open addressing, but tend to clustering no pointers needed, efficient at higher load factor

~~(3)~~

(3) Reduce primary clustering, but can not probe all location in table efficient at lower load factor.

(4) Need a good hash(x) function, otherwise would not be good. efficient at lower load factor, key-dependent resolution sequences, insertion won't fails if one free field is avaiable.

(2) (3) (4) no link needed, real deletion not possible.

4.14 (i) $O(N)$ (ii) $O(N^2)$

6.1 (1) 142 123 65 111 102 , 543 123 65 453 434 111 242 102 , 123 65 111 102 , 453 434 111 242 102 , 879 572 434 111 242 811 102 ,

572 434 111 242 102 , 434 111 242 102 , 111 102 , 242 102 , 811 102 ,

34 pairs in total.

(2) $\frac{n(n-1)}{2}$

(3) It depends on which sort ~~no~~ algorithm your are using. For example, in Heap sort, it doesn't matter how many inversions do you have. However in bubble sort, the time can be reduced when inversions ~~is~~ ~~be~~ number is less.

6.2

(1) 3 1 4 1 5 9 2 6 5

+ ~~+ 3 4 + 5 9 2 65~~

= ~~1 3 1 4 5 9 2 65~~

① 1 3 1 4 2 5 9 5 6

② 1 1 3 2 4 5 5 9 6

③ 1 1 2 3 4 5 5 6 9

(2) 3 1 4 1 5 9 2 6 5

① 3 1 4 1 5 9 2 6 5

② 1 3 4 1 5 9 2 6 5

③ 1 3 4 1 5 9 2 6 5

④ 1 1 3 4 5 9 2 6 5

⑤ 1 1 3 4 5 9 2 6 5

⑥ 1 1 3 4 5 9 2 6 5

⑦ 1 1 2 3 4 5 9 6 5

⑧ 1 1 2 3 4 5 6 9 5

⑨ 1 1 2 3 4 5 5 6 9

(3) 3 1 4 1 5 9 2 6 5

① 1 3 4 1 5 9 2 6 5

② 1 1 3 4 5 9 2 6 5

③ 1 1 2 3 4 5 9 6 5

④ 1 1 2 3 4 5 9 6 5

⑤ 1 1 2 3 4 5 9 6 5

⑥ 1 1 2 3 4 5 9 6 5

⑦ 1 1 2 3 4 5 5 9 6

⑧ 1 1 2 3 4 5 5 6 9

⑨ 1 1 2 3 4 5 5 6 9.

6.6

3 1 4 1 5 9 3 6 5 3 5

5 1 4 1 5 9 3 6 5 3 3

(5 1 4 1 5 3 3 3) 5 6 9

3 5 1 4 1 5 3 3 5 6 9

(3 1 3 1) 3 (5 4 5) 5 6 9

(1 3 1 3) 3 4 5 5 5 6 9

(1 1 1 (3 3) 3 4 5 5 5 6 9.

1 1 3 3 3 4 5 5 5 6 9.

Pivot i j

5

5 1 10

3 1 7

3 1 7

1 1 4