

```

1. #lang racket
2. (require racket/trace)
3. ; esc control = is uncomment
4. ; esc control ; is comment
5.
6. ;> (allcombs 2)
7. ;'((0 0) (0 1) (1 0) (1 1))
8. ;> (allcombs 3)
9. ;'((0 0 0) (0 0 1) (0 1 0) (0 1 1) (1 0 0) (1 0 1) (1 1 0) (1 1 1))
10. (define (allcombs n)
11.   (cond
12.     [(<= n 0) '(())]
13.     [else
14.      (let ((lst (allcombs (- n 1))))
15.        (append (map (lambda (x) (cons 0 x)) lst) (map (lambda (x) (cons 1 x)) lst)))]))
16.
17. ;> (cubesort (allcombs 2) 4)
18. ;'((0 0) (0 1) (1 1) (1 0))
19. ;> (cubesort (allcombs 3) 8)
20. ;'((0 0 0) (0 0 1) (0 1 1) (0 1 0) (1 1 0) (1 1 1) (1 0 1) (1 0 0))
21. (define (cubesort lst len)
22.   (cond
23.     [(<= len 2) lst]
24.     [else
25.      (let* (
26.        (newlen (floor (/ len 2)))
27.        (zerolist (map (lambda (x) (cons 0 x))
28.                        (cubesort
29.                          (map (lambda (x) (cdr x)) (take lst newlen))
30.                          newlen)))
31.        (onelist (map (lambda (x) (cons 1 x))
32.                   (cubesort
33.                     (map (lambda (x) (cdr x)) (reverse (list-tail lst newlen)))
34.                     newlen))))
35.      (if
36.        (= (caar lst) 0)
37.        (append zerolist onelist)
38.        (append onelist zerolist)))]))
39.
40. ;> (hamiltonian_cycle_on_cube 2)
41. ;'((0 0) (0 1) (1 1) (1 0))
42. ;> (hamiltonian_cycle_on_cube 3)
43. ;'((0 0 0) (0 0 1) (0 1 1) (0 1 0) (1 1 0) (1 1 1) (1 0 1) (1 0 0))
44. (define (hamiltonian_cycle_on_cube n)
45.   (cubesort (allcombs n) (expt 2 n)))
46.
47. (trace cubesort)
48. (hamiltonian_cycle_on_cube 3)

```

```

49. ;>(cubesort
50. ;   '((0 0 0) (0 0 1) (0 1 0) (0 1 1) (1 0 0) (1 0 1) (1 1 0) (1 1 1))
51. ;   8)
52. ;> (cubesort '((0 0) (0 1) (1 0) (1 1)) 4)
53. ;> >(cubesort '((0) (1)) 2)
54. ;< <'((0) (1))
55. ;> >(cubesort '((1) (0)) 2)
56. ;< <'((1) (0))
57. ;< '((0 0) (0 1) (1 1) (1 0))
58. ;> (cubesort '((1 1) (1 0) (0 1) (0 0)) 4)
59. ;> >(cubesort '((1) (0)) 2)
60. ;< <'((1) (0))
61. ;> >(cubesort '((0) (1)) 2)
62. ;< <'((0) (1))
63. ;< '((1 0) (1 1) (0 1) (0 0))
64. ;<'((0 0 0) (0 0 1) (0 1 1) (0 1 0) (1 1 0) (1 1 1) (1 0 1) (1 0 0))
65. ;'((0 0 0) (0 0 1) (0 1 1) (0 1 0) (1 1 0) (1 1 1) (1 0 1) (1 0 0))
66.
67.
68.
69.
70. ;(hamiltonian_cycle_on_cube 10)
71. ;(let ((cycle (hamiltonian_cycle_on_cube 5))) (- (length cycle) (length (remove-duplicates cycle))))
72. ;(hamiltonian_cycle_on_cube 4)
73. ;(hamiltonian_cycle_on_cube 2)
74. ;(hamiltonian_cycle_on_cube 1)
75. ;(hamiltonian_cycle_on_cube 0)
76. ;(hamiltonian_cycle_on_cube -1)
77.
78.
79.
80. ; scrap
81. ;   (append
82. ;     (map (lambda (x) (cons 0 x)) (cubesort (map (lambda (x) (cdr x)) (take lst newlen)) newlen))
83. ;     (map (lambda (x) (cons 1 x)) (cubesort (map (lambda (x) (cdr x)) (reverse (list-tail lst newlen))) newlen))
84. ;   )
85. ;   (append
86. ;     (map (lambda (x) (cons 1 x)) (cubesort (map (lambda (x) (cdr x)) (reverse (list-tail lst newlen))) newlen))
87. ;     (map (lambda (x) (cons 0 x)) (cubesort (map (lambda (x) (cdr x)) (take lst newlen)) newlen))
88. ;   ))))
89. ;(cons 1 '())
90. ;(cons '((1)) '((0)))
91. ;(append '((1)) '((0)))
92. ;(list-tail '(1 2 3 4) 2)
93. ;(take '(1 2 3 4) 2)
94. ;(expt 2 3)
95. ;(floor (/ 1 2))
96. ;(map (cons 0) '(1 2 3 4))

```