Sample Questions: SICP Section 2.3

Symbolic Data

1. What is the print value of the expression `(list '1 2 3)`?
   (A) `('1 2 3)`
   (B) `(list 1 2 3)`
   (C) `(1 2 3)`
   (D) `(list (quote 1) 2 3)`
2. Assuming a and b have value 2 and 3, respectively, what is the print value of the expression `(list 'a b)`?
   (A) `(list (quote a) 3)`
   (B) `(list a 3)`
   (C) `(a 3)`
   (D) `((quote 2) 3)`
3. Assuming a and b have value 2 and 3, respectively, what is the print value of the expression `'(list 'a b)`?
   (A) `(list a b)`
   (B) `((quote a) b)`
   (C) `(list a 3)`
   (D) `(list (quote a) b)`
4. Assuming a and b have value 2 and 3, respectively, what is the print value of the expression `('list a b)`?
   (A) no value, an error occurs
   (B) `(list 2 3)`
   (C) `((quote list) 2 3)`
   (D) `(list a b)`
5. Assuming a and b have value 2 and 3, respectively, what is the print value of the expression `'('list a b)`?
   (A) no value, an error occurs
   (B) `(list a b)`
   (C) `((quote list) a b)`
   (D) `((quote list) 2 3)`
6. Assuming a, b, and c have value 2, 3, and 4, respectively, what is the print value of the expression `(car (list 'a 'b 'c))`?
   (A) `2`
   (B) `(quote a)`
   (C) `a`
   (D) `list`
7. Assuming a, b, and c have value 2, 3, and 4, respectively, what is the print value of the expression `(car '(list 'a 'b 'c))`?
   (A) `(quote a)`
   (B) `a`
   (C) `list`
   (D) `(quote list)`
8. Assuming a, b, and c have value 2, 3, and 4, respectively, what is the print value of the expression `(cdr (list 'a 'b 'c))`?
   (A) `(b c)`
   (B) `(3 4)`
   (C) `(a b c)`
   (D) `(2 3 4)`

9. What does the function bound to `=` do?

   (A) performs assignment

   (B) performs structural equality

   (C) performs pointer equality

   (D) performs numeric equaility

10. What does the function bound to *eq?* do?

    (A) performs structural equality

    (B) performs numeric equaility

    (C) performs pointer equality

    (D) performs assignment

11. What does the function bound to *equal?* do?

    (A) performs pointer equality

    (B) performs structural equality

    (C) performs numeric equaility

    (D) performs assignment

12. **T** or **F**: `(eq? '(a b) '(a b))`

13. **T** or **F**: `(equal? (list 'a 'b) '(a b))`

14. **T** or **F**: `(equal? (list 2 'b) '(2 'b))`

15. Consider adding exponentiation to the differentiation program with `**` as the exponentiation operator. What would be a vaild implementation for *exponentiation?*.

    (A) `(define (exponentiation? p) (cadr p))`

    (B) `(define (exponentiation? p) (cons '** (deriv p)))`

    (C) `(define (exponentiation? p) (eq? (car p) '**))`

    (D) `(define (exponentiation? p) (eq? (cadr p) '**))`

16. Consider adding exponentiation to the differentiation program with `**` as the exponentiation operator. What would be a vaild implementation for *base*.

    (A) `(define (base p) (if (= (exponent p) 0) 1))`

    (B) `(define (base p) (if (= (exponent p) 1) (base p)))`

    (C) `(define (base p) (cadr p))`

    (D) `(define (base p) (car p))`

17. Consider adding exponentiation to the differentiation program with `**` as the exponentiation operator. What would be a vaild implementation for *exponent*.

    (A) `(define (exponent p) (car p))`

    (B) `(define (exponent p) (caddr p))`

    (C) `(define (exponent p) (if (= (base p) 1) (exponent p)))`

    (D) `(define (exponent p) (if (= (exponent p) 0) 1))`

18. Consider adding exponentiation to the differentiation program with `**` as the exponentiation operator. Which of the following cases is correct for the *make-exponentiation* working on an expression p that has a base of zero? Assume the exponent is non-zero.

    (A) 0

    (B) `(make-product (make-product (exponent p) (make-exponentiation (base p) (- (exponent p) 1))) (derive (base p) var)`

    (C) `(make-product (exponent p) (make-exponentiation (base p) (- (exponent p) 1)))`

    (D) 1

19. Consider modifying the *make-sum* function of the text's differentiation program to handle arity of 2 or more operands. Which of the following cases is correct for the *addend* selector if a sum s has three or more operands? Note: the *addend* is the right-hand-side operand.

    (A) `(make-sum (caddr s) (make-sum (cadddr s)))`

    (B) `(make-sum (car s) (cadr s))`

    (C) `(apply make-sum (cddr s))`

    (D) `(caddr s)`

20. Consider modifying the *make-sum* function of the text's differentiation program to handle arity of one or more operands. Which of the following cases is correct for the addend selector if a sum s has exactly two operands?

    (A) `(make-sum (car s) (cadr s))`

    (B) `(make-sum (caddr s) (make-sum (cadddr s)))`

    (C) `(apply make-sum (cddr s))`

    (D) `(caddr s)`

21. Consider modifying the *make-sum* function of the text's differentiation program to handle arity of 1 or more operands. Which of the following cases is correct for the addend selector if a sum s has exactly one operands?

    (A) `(make-sum (cadr s) (make-sum (caddr s)))`

    (B) `(apply make-sum (cddr s))`

    (C) `(caddr s)`

    (D) 0

22. What is the time complexity of *adjoin*, *member?*, *union*, and *intersection* operations on sets if the sets are represented as unordered lists with no duplicates?

    (A) linear, linear, linear, linear

    (B) constant, linear, linear, linear

    (C) constant, linear, linear, quadratic

    (D) linear, linear, quadratic, quadratic

    (E) linear, linear, linear, quadratic

    (F) constant, linear, quadratic, quadratic

23. What is the time complexity of *adjoin*, *member?*, *union*, and *intersection* operations on sets if the sets are represented as unordered lists with duplicates allowed? Assume there will never be more than n duplicates per element where n is the number of (unique) elements in the set.

    (A) constant, quadratic, cubic, cubic

    (B) constant, quadratic, quartic, quartic

    (C) linear, linear, quadratic, quadratic

    (D) linear, linear, quadratic, cubic

    (E) constant, quadratic, cubic, quartic

    (F) constant, quadratic, quadratic, quadratic

24. Define the *adjoin* operation for unordered lists, both with and without duplicates.

25. What is the time complexity of *adjoin*, *member?*, *union*, and *intersection* operations on sets if the sets are represented as ordered lists? Assume the sets are roughly equal in size.

    (A) linear, linear, linear, quadratic

    (B) log, log, quadratic, quadratic

    (C) linear, linear, linear, linear

    (D) log, linear, linear, linear

    (E) log, log, linear, linear

26. What is the time complexity of *adjoin*, *member?*, *union*, and *intersection* operations on sets if the sets are represented as ordered arrays? Assume the sets are roughly equal in size.

    (A) linear, linear, linear, quadratic

    (B) log, log, linear, linear

    (C) log, log, quadratic, quadratic

    (D) linear, log, linear, linear

    (E) log, linear, linear, linear

27. What is the time complexity of *adjoin*, *member?*, *union*, and *intersection* operations on sets if the sets are represented as ordinary binary search trees? Assume the sets are roughly equal in size.

    (A) log, log, linear, linear

    (B) linear, linear, linear, linear

    (C) log, linear, linear, linear

    (D) linear, linear, quadratic, quadratic

    (E) linear, linear, linear, quadratic

    (F) log, log, quadratic, quadratic

Expect questions in the same vein as the previous set.