Studytonight – DS test 1 – Aditya Jain

(Stack)

1. **Which one of the following is an application of Stack Data Structure?**

a) Managing function calls
b) The stock span problem
c) Arithmetic expression evaluation
d) All of the above

2. **Which of the following is true about linked list implementation of stack?**

a) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end.
b) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning.
c) Both of the above
d) None of the above

Explanation: To keep the **Last In First Out (LIFO) order**, a stack can be implemented using linked list in two ways: a) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from beginning. b) In push operation, if new nodes are inserted at the end of linked list, then in pop operation, nodes must be removed from end.

3. **Consider the following pseudocode that uses a stack**

```
declare a stack of characters
while ( there are more characters in the word to read )
{
  read a character
  push the character on the stack
}
while ( the stack is not empty )
{
  pop a character off the stack
  write the character to the screen
```

}

What is output for input "Studytonight"?

a) StudytonightStudytonight
b) thginotydutS
c) Studytonight
d) thginotydutSthginotydutS

Explanation: Since the stack data structure follows LIFO order. When we pop() items from stack, they are popped in reverse order of their insertion (or push())

4. **Following is an incorrect pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:**

```
declare a character stack
while ( more input is available)
{
  read a character
  if ( the character is a '(' )
    push it on the stack
  else if ( the character is a ')' and the stack is not empty )
    pop a character off the stack
  else
    print "unbalanced" and exit
 }
 print "balanced"
```

Which of these unbalanced sequences does the above code think is balanced?

a) ((())
b) ())(()
c) (()()))
d) (()))()

Explanation: At the end of while loop, we must check whether the stack is empty or not. For input ((()), the stack doesn't remain empty after the loop

5. The following postfix expression with single digit operands is evaluated using a stack:

```
8 2 3 ^ / 2 3 * + 5 1 * -
```

Note that ^ is the exponentiation operator. The top two elements of the stack after the first * is evaluated are:

a) 6, 1
b) 5, 7
c) 3, 2
d) 1, 5

Explanation: The algorithm for evaluating any postfix expression is fairly straightforward:

```
1. While there are input tokens left
   o Read the next token from input.
   o If the token is a value
     + Push it onto the stack.
   o Otherwise, the token is an operator
    (operator here includes both operators, and functions).
     * It is known a priori that the operator takes n arguments.
     * If there are fewer than n values on the stack
      (Error) The user has not input sufficient values in the expression.
     * Else, Pop the top n values from the stack.
     * Evaluate the operator, with the values as arguments.
     * Push the returned results, if any, back onto the stack.
2. If there is only one value in the stack
   o That value is the result of the calculation.
3. If there are more values in the stack
   o (Error)  The user input has too many values.
```

Source for algorithm: http://en.wikipedia.org/wiki/Reverse_Polish_notation#The_postfix_algorithm

6. Assume that the operators +, -, × are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, x , +, -. The postfix expression corresponding to the infix expression a + b × c - d ^ e ^ f is

   a) **abcx+def^^-**
   b) abcx+de^f^-
   c) ab+cxd-e^f^
   d) -+axbc^^def

7. To evaluate an expression without any embedded function calls:

a) **One stack is enough**
b) Two stacks are needed
c) As many stacks as the height of the expression tree are needed
d) A Turing machine is needed in the general case

Explanation: Any expression can be converted into Postfix or Prefix form.
Prefix and postfix evaluation can be done using a single stack.

8. The result evaluating the postfix expression 10 5 + 60 6 / * 8 – is

   a) 284
   b) 213
   c) **142**
   d) 71

9. Suppose a stack is to be implemented with a linked list instead of an array. What would be the effect on the time complexity of the push and pop operations of the stack implemented using linked list (Assuming stack is implemented efficiently)?

a) O(1) for insertion and O(n) for deletion
b) O(1) for insertion and O(1) for deletion
c) O(n) for insertion and O(1) for deletion
d) O(n) for insertion and O(n) for deletion

Explanation: Stack can be implemented using link list having O(1) bounds for both insertion as well as deletion by inserting and deleting the element from the beginning of the list.

10. Consider n elements that are equally distributed in k stacks. In each stack, elements of it are arranged in ascending order (min is at the top in each of the stack and then increasing downwards). Given a queue of size n in which we have to put all n elements in increasing order. What will be the time complexity of the best known algorithm?

a) O(nlogk)
b) O(nk)
c) O(n^2)
d) O(k^2)

Explanation:
In nlogk it can be done by creating a min heap of size k and adding all the top - elements of all the stacks. After extracting the min , add the next element from the stack from which we have got our 1st minimum. Time Complexity = O(k) (For Creating Heap of size k) + (n-k)log k (Insertions into the heap).

11. Consider the following statements:

i. First-in-first out types of computations are efficiently supported by STACKS.
ii. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.
iii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.
iv. Last-in-first-out type of computations are efficiently supported by QUEUES.

Which of the following is correct?

a) (ii) and (iii) are true

b) (i) and (ii) are true
c) (iii) and (iv) are true
d) (ii) and (iv) are true

12. A priority queue Q is used to implement a stack S that stores characters. PUSH(C) is implemented as INSERT(Q, C, K) where K is an appropriate integer key chosen by the implementation. POP is implemented as DELETEMIN(Q). For a sequence of operations, the keys chosen are in

a) Non-increasing order
b) Non-decreasing order
c) strictly increasing order
d) strictly decreasing order

Explanation: We are implementing a STACK using Priority Queue. Note that Stack implementation is always last in first out (LIFO) order. As given "POP is implemented as DELETEMIN(Q)" that means Stack returns minimum element always. So, we need to implement PUSH(C) using INSERT(Q, C, K) where K is key chosen from strictly-decreasing order(only this order will ensure stack will return minimum element when we POP an element). That is answer, option (D) is true.

13. A function f defined on stacks of integers satisfies the following properties. f(Ø) = 0 and

f (push (S, i)) = max (f(S), 0) + i for all stacks S and integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

a) 6

b) 4

c) 3

d) 2

Explanation:
$f(S) = 0$, $\max(f(S), 0) = 0$, $i = 2$ $f(S)_{new} = \max(f(S), 0) + i = 0 + 2 = 2$
$f(S) = 2$, $\max(f(S), 0) = 2$, $i = -3$ $f(S)_{new} = \max(f(S), 0) + i = 2 - 3 = -1$
$f(S) = -1$, $\max(f(S), 0) = 0$, $i = 2$ $f(S)_{new} = \max(f(S), 0) + i = 0 + 2 = 2$
$f(S) = 2$, $\max(f(S), 0) = 2$, $i = -1$ $f(S)_{new} = \max(f(S), 0) + i = 2 - 1 = 1$
$f(S) = 1$, $\max(f(S), 0) = 1$, $i = 2$ $f(S)_{new} = \max(f(S), 0) + i = 1 + 2 = 3$

Thus, option (C) is correct.

14. **Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing.**

```
void fun(int n)
{
   Stack S;  // Say it creates an empty stack S
   while (n > 0)
   {
     // This line pushes the value of n%2 to stack S
     push(&S, n%2);

     n = n/2;
   }

   // Run while Stack S is not empty
   while (!isEmpty(&S))
     printf("%d ", pop(&S)); // pop an element from S and print it
}
```

What does the above function do in general?

a) Prints binary representation of n in reverse order

b) Prints binary representation of n

c) Prints the value of Logn

d) Prints the value of Logn in reverse order

15. A single array A[1..MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (topl< top 2) point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for “stack full” is

a) (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2+1)
b) top1 + top2 = MAXSIZE
c) (top1= MAXSIZE/2) or (top2 = MAXSIZE)
d) top1= top2 -1

Explanation: If we are to use space efficiently then size of the any stack can be more than MAXSIZE/2. Both stacks will grow from both ends and if any of the stack top reaches near to the other top then stacks are full. So the condition will be top1 = top2 -1 (given that top1 < top2)