Questions for Question Bank

21CSC201J: Data Structures and Algorithms

Unit – II: Implementation of Lists (Array, Cursor based, Linked Lists)

Prepared by: Dr. Indhumathi Raman

Q1 – Q5: Objective Type Questions

1. Which of the following is false about array implementation of lists?
   a. Arrays supports random access of elements.
   b. Array elements are stored in a contiguous manner.
   c. New elements can be added dynamically.
   d. Arrays have fixed size.

Answer: c

2. Which of the following a correct declaration of a linked list?
   a.
   ```
   struct node *
   {
      int data;
      node * link;
   }
   ```
   b.
   ```
   struct node
   {
      int data;
      struct node * link;
   }
   ```
   c.
   ```
   struct node
   {
      int data;
      node link;
   }
   ```
   d.
   ```
   struct node *
   {
      int data;
      struct node * link;
   }
   ```

Answer: b

3. Which of the following false about a Cursor Linked List?
   a. A CursorList is an array version of a Linked List

b. In a CursorList, instead of each node containing a pointer to the next item in the linked list, each node element in the array contains the index for the next node element.
c. Insert a new node at position pointed by header
d. Deletes the element at position pointed by header

Answer: d

4. Suppose cursor refers to a node in a linked list (using the IntNode class with instance variables called data and link). What statement changes cursor so that it refers to the next node?
   a. cursor++;
   b. cursor = link;
   c. cursor += link;
   d. cursor = cursor.link;

Answer: d

5. Which of the following is not feasible to implement in a linked list?
   a. Linear Search
   b. Merge Sort
   c. Insertion Sort
   d. Binary Search

Answer: d

<u>Understanding question [Blooms level two]</u>

6. Assume there are two linked lists M and N with head pointers headM and headN respectively. Write a program to join the list M after the end of list N.

   Answer:

```
typedef struct node {

    int data;

    node * next;

} temp;

temp = headN;

while(temp->next !=NULL)

            temp = temp->next;

temp->next=headM;
```

7. How does an array and a linked list representation of items answer the question "What is the item at position n?". Which is better?

   Answer:
   Array: *print(a[n])* provides the item at position n.
   Linked List: The nth item is printed as

   ```
   temp=head;
   for(i=1; i<=n; i++)
      temp = temp->next;
   print(temp.data);
   ```

   Arrays provide random access to elements by providing the index value within square brackets. In the linked list, we need to traverse through each element until we reach the nth position. Time taken to access an element represented in arrays is less than the singly, doubly and circular linked lists. Thus, array implementation is used to access the item at the position n.

8. For which of the following operations, is array implementation of list is best. Justify your claim in each case.
   a. Random access of elements
   b. Insertion at the end (assuming there is space in array)
   c. Insertion at the front.
   d. Deletion at the front.
   e. Deletion at the end.

   Answer: a, b and e.

   a. Arrays provide random access to elements by providing the index value within square brackets. In the linked list, we need to traverse through each element until we reach the nth position.
   b. Defining a[n]=item will insert an item at the end of the array, where in a linked list, we need to traverse till the end and then insert the item.
      Same reason holds for option e also.

Scenario Based

9. Suppose a list of 100 songs are stored in a list in the ascending order of its song id. (Assume that a song id is an integer assigned to a song uniquely). If we want to search for a particular song id for hearing the particular song, then which implementation of the list would facilitate answering such a query in a quick manner? Write an algorithm to perform your suggested implementation method. What is its time complexity?

   Answer:

Arrays provide random access to elements by providing the index value within square brackets. In the linked list, we need to traverse through each element until we reach the particular song. Since the songs are sorted in alphabetical order, binary search can be performed.

Algorithm for Binary Search

```
#include <stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
   int mid;
   if(end >= beg)
   {       mid = (beg + end)/2;
/* if the item to be searched is present at middle */
       if(a[mid] == val)
       {
          return mid+1;
       }
          /* if the item to be searched is smaller than middle, then it can only be in left subarray */
       else if(a[mid] < val)
       {
          return binarySearch(a, mid+1, end, val);
       }
          /* if the item to be searched is greater than middle, then it can only be in right subarray */
       else
       {
          return binarySearch(a, beg, mid-1, val);
       }
   }
   return -1;
}
int main() {
  int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array of song ids.
  int val = 40; // value to be searched – song to be heard
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = binarySearch(a, 0, n-1, val); // Store result
  printf("The elements of the array are - ");
  for (int i = 0; i < n; i++)
  printf("%d ", a[i]);
  printf("\nElement to be searched is - %d", val);
  if (res == -1)
  printf("\nElement is not present in the array");
  else
  printf("\nElement is present at %d position of array", res);
  return 0;
}
```

Time complexity:

T(n) = T(n/2) + O(1) which, on solving, yields **T(n) = O(logn)**.

10. The CGPA scores of a team of 6 students are given as below:

Name	CGPA
AAA	6.8
BBB	8
CCC	7.2
DDD	9.2
EEE	6.2
FFF	7.8

If we want to delete the names of those students who secured less than 6 CGPA, then which implementation of the list would facilitate answering such a query in a quick manner? Write an algorithm to perform your suggested implementation method. What is its time complexity?

Linked list provides O(1) time deletion of nodes and hence is prefereable.

```
void deleteNode(struct node ** head, int key) // key = 6 here
{
 struct node * temp;
  if (( * head) -> data == key) {    // key = 6 here
    temp = * head;
    * head = ( * head) -> next;
    free(temp);
  } else {
    struct node * current = * head;
    while (current -> next != NULL) {
      if (current -> next -> data == key) {
        temp = current -> next;
        current -> next = current -> next -> next;
        free(temp);
        break;
      } else
        current = current -> next;
    }
  }
}
```

Time complexity: n*O(1) = O(n). // Worst case: All n nodes have CGPA less than 6 and hence has to be deleted. Each deletion takes O(1) time.