BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
HYDERABAD CAMPUS
First Semester 2023-2024 Mid-Semester Examinations
BIFS F232: Foundations of Data Structures and Algorithms
Solutions Key 15th Oct 2023

Q.1 a)

```
while(!S1.isEmpty()){
        val = pop(S1);
        insert val into rear end of D;
}
while(!S2.isEmpty()){
        val = pop(S2);
        insert val into rear end of D;
}
while(!D.isEmpty()){
        val = element removed from rear end of D;
        push(S1, val); //push val into S1
}
```

b)

```
void moveZeroes(vector<int>& nums) {
  int last = 0, n = nums.size();

  for(int i = 0; i < n; i++){
    if(nums[i] != 0){
      nums[last] = nums[i];
      last++;
    }
  }

  for(int i = last; i < n; i++){
    nums[i] = 0;
  }
}
```

c)

```
int m=0, p=0; //O(1)
    for(int k=0; k< n; k++){ //O(n)
      p = p + 1;
    }
    for(int i=0; i< n; i++) { //O(n)
      for (int j=0; j< n; j++) { O(n)
        m = m + 1; // O(1)
      }
    }
```

Run-time complexity = O(1) + O(n) + O(n*n) = O(n*n)

Q.2 a)

```
int LinkedListLength (struct Node* head) {
  while (head && head->next) {
          head = head->next->next;
  }
  if (!head)
        return 0;
  return 1;
}
```

b)

```
Node* delete(Node* head) {
  // base case
  if (head == NULL || head->next == NULL ) return head;
  Node* second = head->next;
  Node* rem = delete(second->next);
  head->next = rem;
  delete second;
  return head;
}
```

c) `return root->data;`

```
return (left > right? left : right) + root->data;
```

Q.3 a)

Step1:

200	25	X

100

100	36	X

200

Ptr1 Ptr2

Step2:

200	25	X

100

X	36	100

200

Ptr1 (H) Ptr2=X

b) The worst case running time of find2D is $O(n^2)$. This is seen by examining the worst case where the element x is the very last item in the n X n array to be examined. In this case, find2D calls the algorithm arrayFind n times. arrayFind will then have to search all n elements for each call until the final call when x is found. Therefore, n comparisons are done for each arrayFind call. Since arrayFind is called n times, we have n . n operations, or an $O(n^2)$ running time. But the size, N, of A is n^2, so this is also O(N)-time algorithm. Thus, this is actually a linear-time algorithm, since its running time is equal to a linear function of the input size.

c) // replace current node data with the next node’s data and keep moving until we reach the second last node

```
while (cursor->next->next != NULL) {
   cursor->data = cursor->next->data;
   cursor = cursor->next;
}
// get hold of the last node
Node *last = cursor->next;

// replace [cursor]'s data with the last node's data.
cursor->data = last->data;

// update [cursor]'s next pointer
cursor->next = NULL;

// free up space
delete (last);
```

Q.4 a)

$T(n) = T(n-1) +1/n$

$T(n-1) = T(n-2) + 1/ n-1$

➔ $T(n) = T(n-2) + 1/n + 1/n-1$

Likewise, $T(n) = T(n-3) +1/n + 1/n-1 + 1/n-2$

➔ $T(n) = T(n-k) + 1/ n – (k-1) + 1 / n – (k-2) + \dots + 1/n$

Substituting $n-k = 1$:

$T(n) = T(1) + 1/2 + 1/3 + \dots 1/n$

➔ $T(n) = \theta (\log n)$

b) Sequence = 10, 7, 12, 4

Pass	Swaps	Sequence
1	10-7, 12-4	(7, 10, 4, 12)
2	10-4	(7, 4, 10, 12)
3	7-4	(4, 7, 10, 12)

i-th pass will be limited to first n-i+1 elements, which will lead to the below worst case complexity:

O ($\sum_{i=1}^{n} n + i - 1$) ➔ 1 + 2 + 3 +… + n ➔ n (n+1)/2 ➔ O(n^2)

A node-based implementation would give O(n) worst case complexity of atIndex function, which will lead to $O(n^2)$ complexity for the inner loop and $O(n^3)$ for the outer loop.

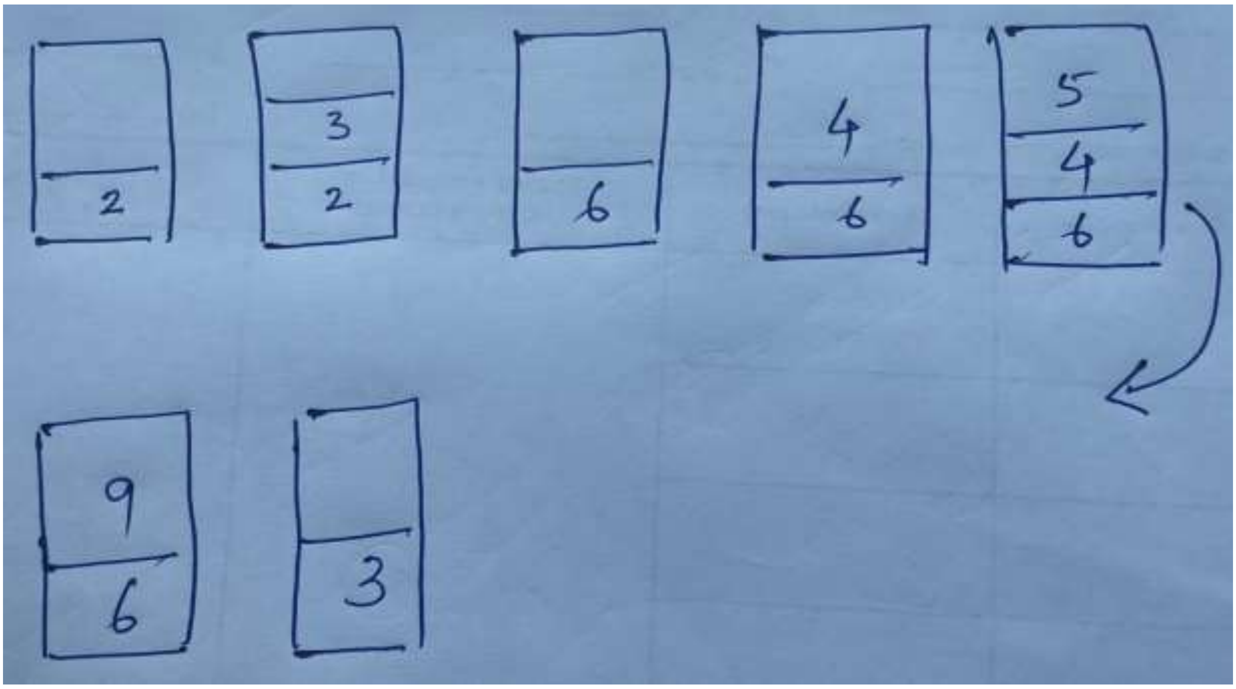
Q.5 a)

	CPU1	CPU2	CPU3
t1:	T1	T3	T4
t2:	T2	T9	T5
t3:	T8	T7	T6

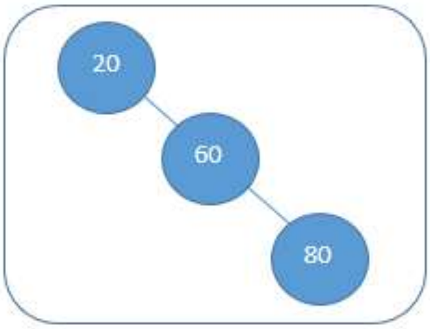
Or:

t3:	T7	T8	T6

b) The equivalent postfix expression is: 2 3 * 4 5 + -
In the below sequence, scenarios (a) and (b) are seen, not (c). Hence, scenario (c) is not possible.



c) The binary tree will be created as below:



------------------- ~ ---------------------