SRN | | | | | | | | | | | |

PES UNIVERSITY

PES University, Bengaluru

(Established under Karnataka Act No. 16 of 2013)

UE15CS202

DECEMBER 2016: END SEMESTER ASSESSMENT

UE15CS202: Data Structures (Autonomy)

Time: 3 Hrs (180 minutes) | Answer All Questions | Max Marks: 50

Instructions: This examination is closed-book, but you are allowed one double-sided cheat sheet of handwritten notes. No electronic devices (calculators, mobile phones, etc.) are permitted in the testing area. This examination has **FOUR** questions and **TWO** pages.

Question 1 (10 points) Consider the following function in the List ADT that returns the *gap* between two given `ListEntrys` in a `List` (the gap between a `ListEntry` and itself is defined as 0, the gap between adjacent `ListEntrys` is defined as 1, etc.), or returns –1 if the arguments are invalid:

```
int gap(List list, ListEntry e1, ListEntry e2);
```

Let $T(n)$ be the worst-case running time of `gap` on a list of size n.

(a) (3 points) Explain why $T(n)$ is $O(n)$ for a linked list implementation.
(b) (7 points) Explain why $T(n)$ is $O(1)$ for an array implementation.

Question 2 (10 points) Consider a binary search tree where every node `r` has a `numNodes` field that indicates the number of nodes present in the sub-tree rooted at `r`. Describe (as precisely as possible) what the following recursive function computes:

```
tNode *solve(tNode *r, int k) {
  if(r == NULL || k <= 0 || k > r->numNodes) {
    return NULL;
  }
  int temp = 1;
  if(r->left != NULL) { temp += r->left->numNodes; }
  if(k == temp) { return r; }
  if(k < temp) { return solve(r->left, k); }
  return solve(r->right, k - temp);
}
```

Note: For partial credit, explain how the function executes and what it returns on these arguments:

```
r = makeNode(7,
      makeNode(4, makeNode(3,NULL,NULL), NULL),
      makeNode(12, makeNode(9,NULL,NULL), makeNode(16,NULL,NULL))
    );
k = 4;
```

SRN | | | | | | | | | | | |

Question 3 (15 points) An algorithm maintains a data structure with n items, where each item is a pair (*priority*, *value*). In each step, the algorithm performs **exactly one** of these three tasks:

i. Inserts a new item into the data structure.
ii. Extracts the item with lowest priority.
iii. Extracts the item with highest priority.

Describe an efficient data structure for this algorithm. For full credit, ensure that each step of the algorithm can be completed in $O(\log n)$ time.

Question 4 (15 points) Consider the following `hashCode` function for binary trees.

```
int hashCode(tNode *r) {
  int result = 1;
  if(r != NULL) {
    result = 31*result + hashCode(r->left);
    result = 31*result + hashCode(r->right);
  }
  return result;
}
```

(a) (7 points) Explain why this `hashCode` function can lead to excessive collisions.
(b) (8 points) Improve this `hashCode` function in a way that reduces the number of collisions.