1.
(a) Write an algorithm for the merge procedure in the mergesort algorithm that only requires O(1) extra space. Recall, the merge algorithm discussed in the class requires O(n) extra space. What is the time complexity of your algorithm? Answers with time complexity greater than O(n) are accepted. (3)
(b) Compute the overall time complexity of the mergesort algorithm that uses your merge algorithm in part A. (2)

a)
```
int find_min_idx(int arr[], int start, int end) {
  int min = arr[start];
  int i, idx = start;

  for (i = start+1; i <= end; i++) {
     if (arr[i] < min) {
       min = arr[i];
       idx = i;
     }
  }
  return idx;
}

void merge(int arr[], int start, int mid, int end) {
    for (i = start; i < end; i++) {
       int idx = find_min_idx(arr, i, end);
       if (idx != i) {
          int t = arr[i];
          arr[i] = arr[idx];
          arr[idx] = t;
       }
    }
}
```

In this algorithm, instead of using temporary arrays, we are replacing the first element with the smallest element, the second element with the second smallest, and so on. Other solutions are also accepted that don't use a temporary array of variable length.

Grading: Use of a temporary array of variable length - 0
The algorithm doesn't sort the elements in the range [start, end] - 0
Otherwise: 3 marks

b)

$T(n) = 2T(n/2) + cn^2$

$= 2(2T(n/2^2) + c(n/2)^2) + cn^2$

$= 2^2T(n/2^2) + cn^2((1/2) + 1)$

$= 2^2(2T(n/2^3) + c(n/4)^2) + cn^2((1/2) + 1)$

$= 2^3T(n/2^3) + cn^2((1/2)^2 + (1/2) + 1)$

= …

$= 2^kT(n/2^k) + cn^2((1/2)^{k-1} + \dots + (1/2)^2 + (1/2) + 1)$

$= 2^kT(n/2^k) + cn^2(1 - (1/2)^k) / (1 - (1/2))$

$= 2^kT(n/2^k) + 2cn^2 (1 - (1/2^k))$

Substituting $n = 2^k$

$T(n) = nT(1) + 2cn^2 - 2cn \qquad = O(n^2)$

Grading: Give zero if the algorithm in part a is incorrect.

Give one mark if the first line ($T(n) = 2T(n/2) + cn^2$) is correct. This may be different if they have used a faster correct sorting algorithm in part A.
Give full marks if the derivation also is correct.

2. Let's say we have two sorted arrays of integers A and B. Write an algorithm that takes an integer x as input and finds two integers x1 and x2 such that x1 belongs to A, x2 belongs to B, and the sum of x1 and x2 is x. If no such x1 and x2 exist, the algorithm prints an error. The time complexity of your algorithm should be less than O(n^2). (4)

int binary_search(int arr[], int n, int val);
binary_search algorithm searches the element in an array (arr) of length n and returns one if val is present in the arr; otherwise, it returns zero.

```
void find_sum (int arr1[], int arr2[], int x, int n) {
  int i;
   for (i = 0; i < n; i++) {
       if (bsearch(arr2, n, x - arr1[i]) == 1) {
         printf("x1: %d x2:%d\n", arr1[i], x - arr1[i]);
         return;
       }
   }
   printf("error: no such x1 and x2\n");
}
```

Grading:
Give-0 if the binary search or any other algorithm faster than linear search is not used.
Deduct two marks if the clear description of binary_search (i.e., the arguments and return value are unclear) or implementation is not given.
Give full marks if the algorithm is correct.

3. Write a recursive algorithm that takes n integers and an integer r as input and prints all nCr combinations. For example, if the input numbers are 1, 2, 3, 4 and the value of r is 2, then all the following six combinations will be printed.
1 2
1 3
1 4
2 3
2 4
3 4

Note that nCr is all possible ways of selecting r items out of n numbers in which the order of selection doesn't matter. For example, 1 2 is the same as 2 1 because the order doesn't matter. (5)

```
// res is a buffer to store the current selection
// nr is the number of elements in the current selection

void nCr(int arr[], int n, int lo, int r, int res[], int nr) {
  int i;
  if (nr == r) {
    for (i = 0; i < r; i++)
      printf("%d ", res[i]);
    printf("\n");
    return;
  }
  for (i = lo; i < n; i++) {
    res[nr] = arr[i];
    nCr(arr, n, i+1, r, res, nr+1);
  }
}
```

Grading:
It is unlikely that nCr can be implemented without using a buffer to store the current selection.
Give zero if an additional buffer is not used.

Give max two marks if an additional buffer is used, but the solution is not correct.
Give full marks if the solution is correct.

4.
(a) There are n balls in a row. Each ball is colored red, white, or blue. You are required to sort these balls so that all the red balls precede all the white balls, which in turn precede all the blue balls. The only operations you are allowed to perform are “check”, which returns the color of a ball, and “swap” which swaps the positions of two balls. Give an O(n) time algorithm for this problem. Justify your answer. (3)

(b) Consider k different colours c1, . . . , ck, where k is a constant. Now, assume that there are n balls in a row where each ball is coloured c1, c2, . . . , or ck. Then sort these balls such that balls coloured ci precede balls coloured cj for all i < j. Generalize your algorithm for part(a) to give an algorithm for this problem. What is the complexity of your generalized algorithm? (2)

a)
```
void sort(int arr[], int n) {
  int red_idx = -1;
  int blue_idx = n;
  int i;
  for (i = 0; i < blue_idx; i++) {
   int res = check(arr, i);
    if (res == RED) {
       red_idx += 1;
       swap(arr, i, red_idx);
    }
    else if (res == BLUE) {
      blue_idx -= 1;
      swap(arr, i, blue_idx);
    }
  }
}
```

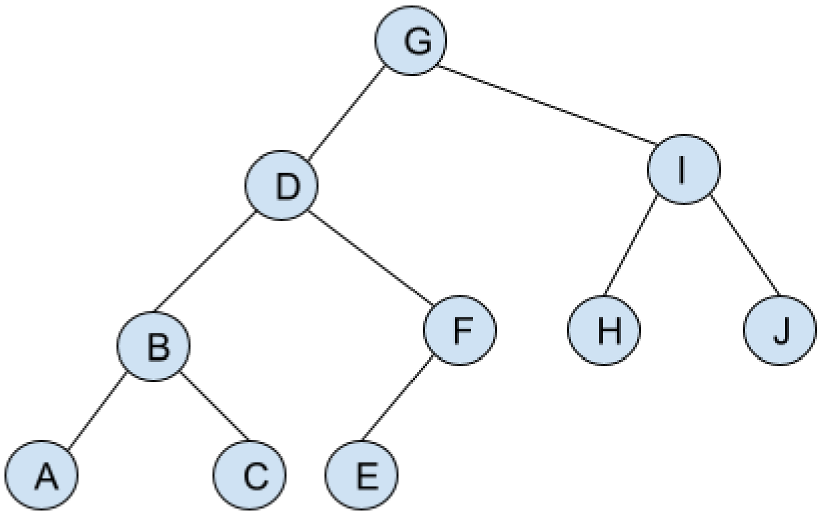
Grading:
Give zero if they have accessed/modified the array elements without using swap and check.
Full marks if the algorithm is correct.

b)
```
void sort_k(int arr[], int n) {
   int count[k+1] = {0};
   for (i = 0; i < n; i++) {
      int color = check(arr, i);  // assuming 1 <= color <= k
      count[color] += 1;
   }
```

```
 for (i = 2; i <= k; i++) {
   count[i] += count[i-1];
 }
 for (i = 0; i < n; i++) {
     int color = check(arr, i);  // assuming 1 <= color <= k
     int idx = count[color] - 1;
     swap(arr, i, idx);
     count[color] = idx;
  }
}
```

Grading:
Give zero if they have accessed/modified the array elements without using swap and check.
Full marks if the algorithm is correct.

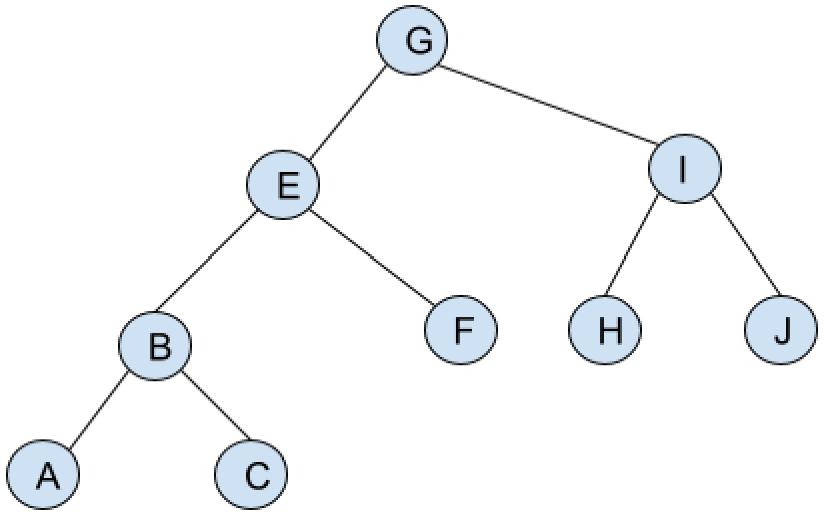
5. Create a binary search tree with the nodes G, D, I, B, F, J, H, A, C, E. Assume the ordering A < B < C < … < Z. Suppose the node D is deleted. Construct the new binary search tree. Generate the output of the Post-order traversal of the tree before and after the deletion. (You should clearly state any assumptions that you may make). (3)



Grading:
POSTORDER: A C B E F D H J I G
deduct one mark if the postorder is not correct
give zero if the BST is not correct

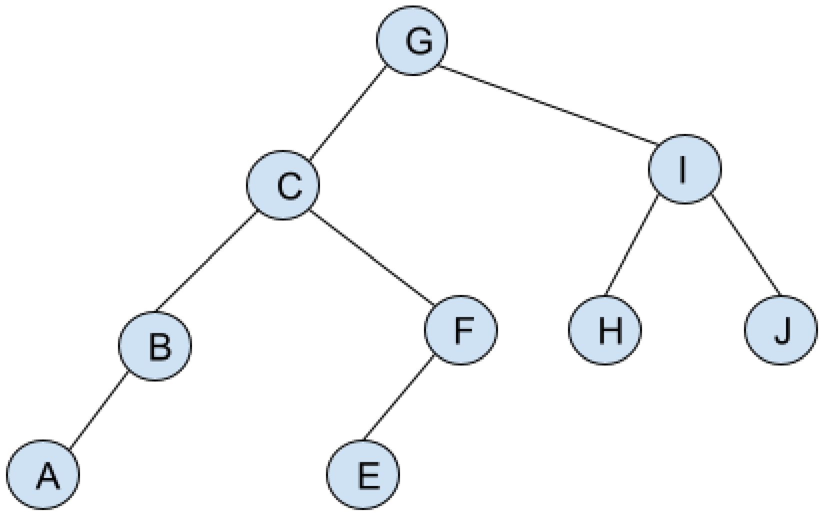
We can delete using the left or right subtree. Both solutions are fine.

DELETE (using right subtree)



POSTORDER: A C B F E H J I G

DELETE (using left subtree)



POSTORDER: A B E F C H J I G

Grading:
Deduct one mark if the postorder after delete is not correct.
Give max one mark if the BST after delete is not correct.

6. Write an algorithm to reverse a doubly-linked list. You can assume that the list is not empty and the head of the linked list is the input to the algorithm. You are not allowed to use an array in your algorithm. The time complexity of your algorithm should not be more than O(n). (3)

An iterative algorithm to reverse a linked list.

```
struct node *reverse_list(struct node *head) {
  struct node *prev, *next;
  struct node *cur = head;
  struct node *ret = cur;

  while (cur != NULL) {
        next = cur->next;
        prev = cur->prev;
        cur->next = prev;
        cur->prev = next;
        ret = cur;
        cur = next;
  }
  return ret;
}
```

A recursive algorithm for reversing a linked list.

```
struct node *reverse_list_rec(struct node *head) {
  assert(head != NULL);
  struct node *next = head->next;
  if (next == NULL) {
        head->prev = NULL;
        return head;
  }
  struct node *head_sublist = reverse_list_rec(next);
  next->next = head;
  head->prev = next;
  head->next = NULL;
  return head_sublist;
}
```

Grading:
Deduct two marks if prev is updated correctly but not next.
Deduct two marks if next is updated correctly but not prev.
Give zero if both prev and next are not being updated correctly.