1. Write an algorithm to find a node that contains the minimum value in a singly linked list of integers. You can assume that the list is not empty. [5]

```
int find_min(struct node *head)
{
  int min = head->val; // deduct one mark if they assumed that the
minimum value is greater than or equal to zero

  struct node *tmp = head->next;
  while (tmp != NULL) {
      if (min > tmp->val) {
        min = tmp->val;
      }
      tmp = tmp->next;
  }
  Give zero if the algorithm doesn't return the correct min, even if
we assume that the numbers are positive
  return min;
}
```

2. Write an algorithm to delete all nodes that contains an even number in a singly linked list of integers. You can assume that the list is not empty and the first node contains an odd number. The time complexity of your algorithm should not be more than O(n). The extra space taken by your algorithm should not depend on the number of elements in the list. [7]

```
void delete_even(struct node *head)
{
  struct node *tmp = head->next;
  struct node *prev = head;
  while (tmp != NULL) {
      if ((tmp->val % 2) == 0) {
        prev->next = tmp->next;
      }
      else {
        prev = tmp;
      }
      tmp = tmp->next;
  }
  // No partial marking. Give full marks only if the algorithm is
correct.
}
```

3. Write an algorithm to insert a new node at position 1000 in a doubly linked list of integers. You can assume that the list is not empty and a pointer to the new node is already given to you. The position of the first node in the linked list is zero, the second node is one, and so on. Your algorithm should print an error if the number of nodes in the linked list is less than 1000. What is the time complexity of your algorithm? The extra space taken by your algorithm should not depend on the number of elements in the list. [8]

We will evaluate it in two parts.
6 marks for the insert algorithm and 2 marks for the complexity
Marks for complexity will be awarded only if they get at least two marks in the insert algorithm.

```
void insert(struct node *head, struct node *n)
{
  struct node *tmp = head;
  int pos = 0;

  // give zero if this loop below is not correct
  while (pos != 999 && tmp) {
      tmp = tmp->next;
      pos++;
  }
  // give one mark if the solution up to this is correct

  if (tmp == NULL) {  // give one mark if this check is present
      printf("number of nodes are less than 1000\n");
      exit(0);
  }
  // give two marks if the solution up to this is correct
  // Now evaluate the rest of the algorithm as follows

  // give four marks if all four assignments are present, and the
check for null is also present
  // give three marks if the check for null is not present, but all
four assignments are correct
  // give one mark if both n->next and n->prev are updated correctly
  // Otherwise, give zero
  n->next = tmp->next;
  if (tmp->next != NULL) {
      tmp->next->prev = n;
  }
  n->prev = tmp;
  tmp->next = n;
}
```

Time complexity:O(1)
Give two marks if the time complexity is correct, and they get at least two marks for the insert algorithm; otherwise, give zero.

4. Suppose array A is an array of integers sorted in descending order. Our goal is to sort array A in ascending order. We have two choices to sort array A in ascending order. The first choice is Quicksort, and the second choice is Mergesort. Which of the two sorting algorithms is better for this job? Justify your answer. [5]

// zero marks if the complexities of quicksort and mergesort are not discussed in the answer
// full marks if the correct complexities are discussed

Because the array is sorted in descending order on every call to the partition algorithm, the final position of the pivot will be the last element of the subarray. Consequently, the array will always be partitioned into two arrays of size (n-1) and zero, respectively. The time complexity of the quicksort algorithm in this scenario is O(n^2). In mergesort, even, in this case, the array will be partitioned into two halves, and therefore the complexity is going to be O(nlogn). Therefore, mergesort is a better choice in this case.