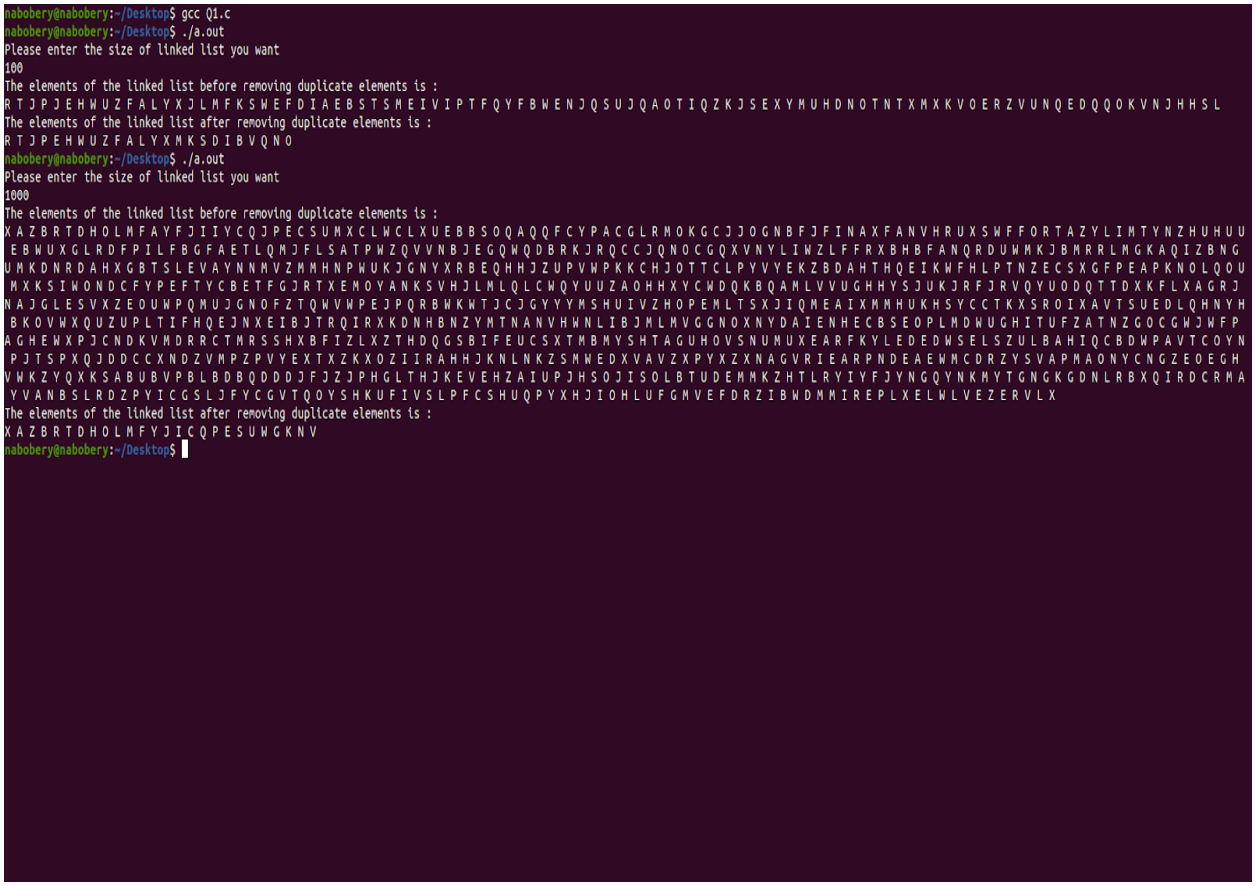
Question 1 Code :

```
/* Question 1 :  Create a Linked List of size N, where each node will have a
random character and pointer to the next node. N should be given as user input.
WAP to remove the duplicate characters from the list.
This question is done by CS20B1044 Avinash R Changrani */
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

// Declaration of structure (node) of the Linked List
struct linkedlist{
      char ch;
      struct linkedlist *nextnode_ptr;
};

void main()
{
      struct linkedlist *head = NULL;
      head = (struct linkedlist*)malloc(sizeof(struct linkedlist));
      int n;
      // Asking size of linked list from user
      printf("Please enter the size of linked list you want\n");
      scanf("%d", &n);
      struct linkedlist *ptr = head;
      srand(time(NULL));
      for(int i = 0; i < n; i++){
      // Generate random characters for the nodes of the linked list using
rand() function
        ptr->ch = 'A' + (rand() % 26);
      // For nodes other than the end, take and  assign the data then allocate
memory for the next node pointer and change the pointer from current node to
the next one
        if (i < n-1){
          ptr->nextnode_ptr = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
          ptr = ptr->nextnode_ptr;
        }
        else {
          ptr->nextnode_ptr = NULL;
        }
      }
      // Printing the elements of the linked list
```

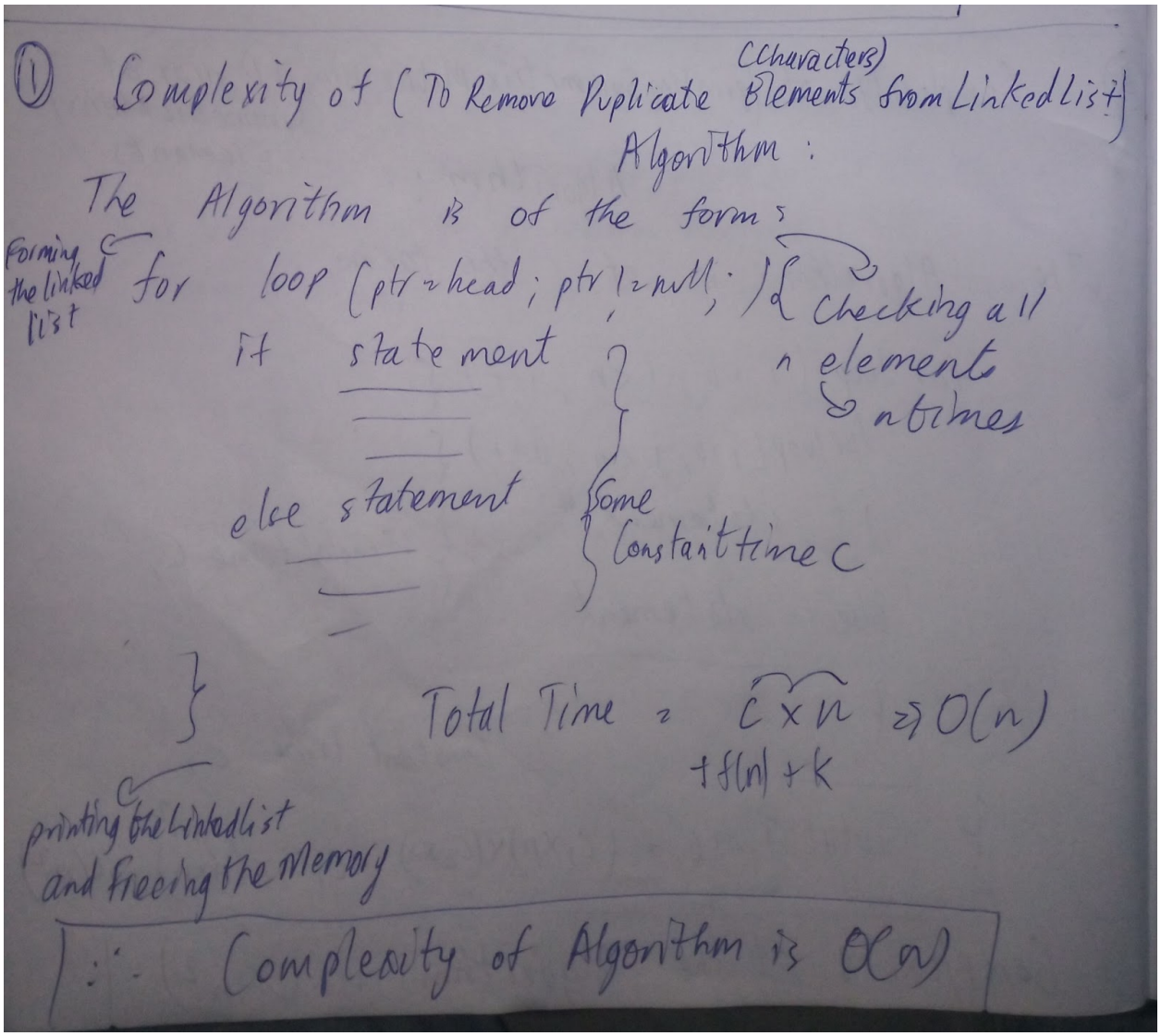
```
ptr = head;
      printf("The elements of the linked list before removing duplicate
elements is : \n");
      while (ptr != NULL){
          printf("%c ", ptr->ch);
          ptr = ptr->nextnode_ptr;
       }
       printf("\n");
       // Defining an array that will keep count of number of characters
spotted
       int *arr;
       // Changing all array elements to zero first
       arr = (int *)malloc(26 * sizeof(int));
       for(int i = 0 ;i < n;i++){
         *(arr + i) = 0;
      }
      // deleting the duplicate elements and preparing the linked list with
non- duplicate elements
      struct linkedlist *prev, *temp;
      for(ptr = head; ptr != NULL;){
         if ( *(arr + (ptr->ch - 65)) == 0){
               *(arr + (ptr->ch - 65)) = 1;
               prev = ptr;
               ptr = ptr->nextnode_ptr;
         }
         else {
               temp = ptr->nextnode_ptr;
               free(ptr);
               prev->nextnode_ptr = temp;
               ptr = temp;
         }
      }
      // Printing the elements of the linked list after removing the duplicate
elements
      ptr = head;
      printf("The elements of the linked list after removing duplicate elements
is : \n");
      while (ptr != NULL){
          printf("%c ", ptr->ch);
          ptr = ptr->nextnode_ptr;
      }
      printf("\n");
```

```
//Freeing the dynamically allocated memory
       free(arr);
       ptr = head;
       struct linkedlist *free_ptr = NULL;
       while (ptr != NULL){
          free_ptr = ptr->nextnode_ptr;
          free(ptr);
          ptr = free_ptr;
       }

}
```

Sample Output :



Complexity Calculation :



Question 2 Code :

```
/* Question 2 : Create a Linked List of N Fibonacci numbers. N should be given
as user input. WAP to find the summation of odd and even Fibonacci numbers in
the series. The program should be run in one pass (only one for/while loop in
the entire program).
This question is done by CS20B1044 Avinash R Changrani */
#include<stdio.h>
#include<stdlib.h>
// Declaration of structure (node) of the Linked List
struct linkedlist{
      long long fib_number;
```

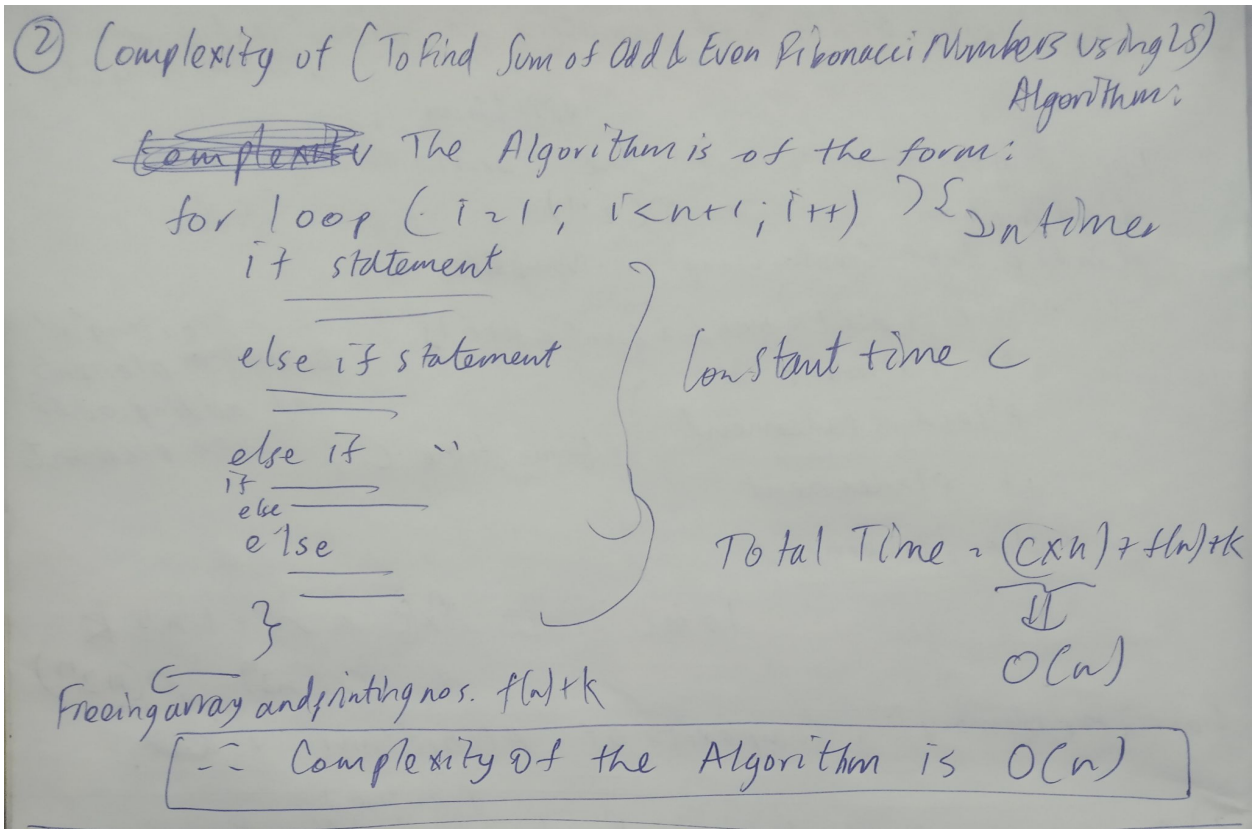
```
      struct linkedlist *nextnode_ptr;
};

void main()
{
      int n;
      // Asking size of fibonacci linked list (Number of fibonacci numbers in
the linked list) from user
      printf("Please enter the size of fibonacci linked list you want\n");
      scanf("%d", &n);
      struct linkedlist *head = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
      struct linkedlist *ptr = head;
      long long fib1 = 1, fib2 = 1;
      long long even_sum = 0, odd_sum = 0;
      // Entering and printing the fibonacci numbers of the linked list
      for (int i = 1; i < n + 1; i++){
         if (i == 1){
            ptr->fib_number = fib1;
            printf("%lld ", ptr->fib_number);
            odd_sum += 1;
            ptr->nextnode_ptr = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
            ptr = ptr->nextnode_ptr;
         }
         else if (i == 2){
            ptr->fib_number = fib2;
            printf("%lld ", ptr->fib_number);
            odd_sum += 1;
            ptr->nextnode_ptr = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
            ptr = ptr->nextnode_ptr;
         }
         else if (i < n){
            ptr->fib_number = fib1 + fib2;
            printf("%lld ", ptr->fib_number);
            fib1 = fib2;
            fib2 = ptr->fib_number;
            if ( ptr->fib_number % 2 == 0){
               even_sum += ptr->fib_number;
            }
            else {
               odd_sum += ptr->fib_number;
            }
```

```
            ptr->nextnode_ptr = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
            ptr = ptr->nextnode_ptr;
         }
         else {
            ptr->fib_number = fib1 + fib2;
            printf("%lld ", ptr->fib_number);
            fib1 = fib2;
            fib2 = ptr->fib_number;
            if ( ptr->fib_number % 2 == 0){
               even_sum += ptr->fib_number;
            }
            else {
               odd_sum += ptr->fib_number;
            }
            ptr = ptr->nextnode_ptr = NULL;
         }
      }

      // Printing the sum of odd and even fibonacci numbers
      printf("\n");
      printf("The sum of even numbers in the fibonacci is %lld\n", even_sum );
      printf("The sum of  odd numbers in the fibonacci is %lld\n", odd_sum );

      //Freeing the dynamically allocated memory
      ptr = head;
      struct linkedlist *free_ptr = NULL;
      while (ptr != NULL){
         free_ptr = ptr->nextnode_ptr;
         free(ptr);
         ptr = free_ptr;
      }

}
```

Sample Output :

```
nabobery@nabobery:~$ cd Desktop
nabobery@nabobery:~/Desktop$ gcc Q2.c
nabobery@nabobery:~/Desktop$ ./a.out
Please enter the size of fibonacci linked list you want
10
1 1 2 3 5 8 13 21 34 55
The sum of even numbers in the fibonacci is 44
The sum of  odd numbers in the fibonacci is 99
nabobery@nabobery:~/Desktop$ ./a.out
Please enter the size of fibonacci linked list you want
50
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 6324
5986 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049 12586269025
The sum of even numbers in the fibonacci is 6293134512
The sum of  odd numbers in the fibonacci is 26658145586
nabobery@nabobery:~/Desktop$
```

Complexity Calculation :



Question 3 Code :

```
/* Question 3 :  Create a Linked List of N students. Each student node will
have roll_no, percentage of marks, and the corresponding grade. The roll_no
will vary from 1 to N. Percentage of marks will be taken as a random input.
This question is done by CS20B1044 Avinash R Changrani */
#include<stdio.h>
#include<stdlib.h>
#include <time.h>
// Declaration of structure (node) of the Linked List containing list of
students
struct studentlist{
      int roll_no;
      float percentage;
      char grade;
      struct studentlist *next;
};

// A function to free dynamically allocated memory of the linked lists
void freelist(void *head){
      struct studentlist *ptr = head;
      struct studentlist *free_ptr = NULL;
      while (ptr != NULL){
         free_ptr = ptr->next;
         free(ptr);
         ptr = free_ptr;
      }
}

void main()
{
      struct studentlist *head = NULL;
      head = (struct studentlist*)malloc(sizeof(struct studentlist));
      int n;
      // Asking size of linked list(number of students) from user
      printf("Please enter the size of linked list you want\n");
      scanf("%d", &n);
      struct studentlist *ptr = head;
      srand(time(NULL));
      for(int i = 0; i < n; i++){
          ptr->roll_no = i+1;
```

```
// Generate random percentages for the nodes(student) of the linked list using
rand() function
          ptr->percentage = ((float)rand()/(float)(RAND_MAX)) * 100 ;
          if (ptr->percentage >= 90)
             ptr->grade = 'A';
          else if (ptr->percentage >= 80 && ptr->percentage < 90)
             ptr->grade = 'B';
          else if (ptr->percentage >= 70 && ptr->percentage < 80)
             ptr->grade = 'C';
          else if (ptr->percentage >= 60 && ptr->percentage < 70)
             ptr->grade = 'D';
          else if (ptr->percentage >= 50 && ptr->percentage < 60)
             ptr->grade = 'E';
          else if (ptr->percentage >= 35 && ptr->percentage < 50)
             ptr->grade = 'P';
          else if (ptr->percentage < 35)
             ptr->grade = 'F';
          if (i < n-1){
            ptr->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            ptr = ptr->next;
          }
          else {
          ptr->next = NULL;
        }
      }

      // Printing the elements(student details) of the linked list before
splitting into assigned grade lists
      ptr = head;
      printf("The Students list before being listed grade wise : \n");
      while (ptr != NULL){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr->roll_no, ptr->percentage, ptr->grade);
        ptr = ptr->next;
        }
        // Creating heads of lists grade wise and assigning the pointer to it
and a temporary pointer
      struct studentlist *head_a =(struct studentlist*) malloc(sizeof(struct
studentlist));
      struct studentlist *head_b =(struct studentlist*) malloc(sizeof(struct
studentlist));
      struct studentlist *head_c =(struct studentlist*) malloc(sizeof(struct
studentlist));
```

```
struct studentlist *head_d =(struct studentlist*) malloc(sizeof(struct
studentlist));
      struct studentlist *head_e =(struct studentlist*) malloc(sizeof(struct
studentlist));
      struct studentlist *head_p =(struct studentlist*) malloc(sizeof(struct
studentlist));
      struct studentlist *head_f =(struct studentlist*) malloc(sizeof(struct
studentlist));
      struct studentlist *ptr_a = head_a, *temp_a = NULL;
      struct studentlist *ptr_b = head_b, *temp_b = NULL;
      struct studentlist *ptr_c = head_c, *temp_c = NULL;
      struct studentlist *ptr_d = head_d, *temp_d = NULL;
      struct studentlist *ptr_e = head_e, *temp_e = NULL;
      struct studentlist *ptr_p = head_p, *temp_p = NULL;
      struct studentlist *ptr_f = head_f, *temp_f = NULL;
      // dividing the list into lists based on grades
      int a = 0,b = 0, c = 0, d = 0, e = 0, p = 0, f = 0; // to keep count of
the number of elements of the grade based lists
      for(ptr = head; ptr != NULL; ptr = ptr->next){
         if (ptr->grade == 'A'){
            a =+ 1;
            ptr_a->grade = ptr->grade;
            ptr_a->percentage = ptr->percentage;
            ptr_a->roll_no = ptr->roll_no;
            ptr_a->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            temp_a = ptr_a;
            ptr_a = ptr_a->next;
         }
         else if (ptr->grade == 'B'){
            b =+ 1;
            ptr_b->grade = ptr->grade;
            ptr_b->percentage = ptr->percentage;
            ptr_b->roll_no = ptr->roll_no;
            ptr_b->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            temp_b = ptr_b;
            ptr_b = ptr_b->next;
         }
         else if (ptr->grade == 'C'){
            c =+ 1;
            ptr_c->grade = ptr->grade;
            ptr_c->percentage = ptr->percentage;
            ptr_c->roll_no = ptr->roll_no;
```

```
 ptr_c->next = (struct studentlist*)malloc(sizeof(struct studentlist));
            temp_c = ptr_c;
            ptr_c = ptr_c->next;
         }
         else if (ptr->grade == 'D'){
            d =+ 1;
            ptr_d->grade = ptr->grade;
            ptr_d->percentage = ptr->percentage;
            ptr_d->roll_no = ptr->roll_no;
            ptr_d->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            temp_d = ptr_d;
            ptr_d = ptr_d->next;
         }
         else if (ptr->grade == 'E'){
            e =+ 1;
            ptr_e->grade = ptr->grade;
            ptr_e->percentage = ptr->percentage;
            ptr_e->roll_no = ptr->roll_no;
            ptr_e->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            temp_e = ptr_e;
            ptr_e = ptr_e->next;
         }
         else if (ptr->grade == 'P'){
            p =+ 1;
            ptr_p->grade = ptr->grade;
            ptr_p->percentage = ptr->percentage;
            ptr_p->roll_no = ptr->roll_no;
            ptr_p->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            temp_p = ptr_p;
            ptr_p = ptr_p->next;
         }
         else if (ptr->grade == 'F'){
            f =+ 1;
            ptr_f->grade = ptr->grade;
            ptr_f->percentage = ptr->percentage;
            ptr_f->roll_no = ptr->roll_no;
            ptr_f->next = (struct studentlist*)malloc(sizeof(struct
studentlist));
            temp_f = ptr_f;
            ptr_f = ptr_f->next;
         }
      }
```

```
// Keeping the end pointer which is temp as NULL if we get a grade and if not
just keep the head is NULL (list is empty)
      if (a > 0)
         temp_a->next = NULL;
      else
         head_a = NULL;
      if (b > 0)
         temp_b->next = NULL;
      else
         head_b = NULL;
      if (c > 0)
         temp_c->next = NULL;
      else
         head_c = NULL;
      if (d > 0)
         temp_d->next = NULL;
      else
         head_d = NULL;
      if (e > 0)
         temp_e->next = NULL;
      else
         head_e = NULL;
      if (p > 0)
         temp_p->next = NULL;
      else
         head_p = NULL;
      if (f > 0)
         temp_f->next = NULL;
      else
         head_f = NULL;
      // Printing the elements(student details) of the linked list after
splitting into assigned grade lists
      printf("The Students list after being listed grade wise : \n");
      printf("Grade A : \n");
      for(ptr_a = head_a; ptr_a != NULL; ptr_a = ptr_a->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_a->roll_no, ptr_a->percentage, ptr_a->grade);
        }
        printf("Grade B : \n");
        for(ptr_b = head_b; ptr_b != NULL; ptr_b = ptr_b->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_b->roll_no, ptr_b->percentage, ptr_b->grade);
        }
      printf("Grade C : \n");
```

```
for(ptr_c = head_c; ptr_c != NULL; ptr_c = ptr_c->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_c->roll_no, ptr_c->percentage, ptr_c->grade);
        }
        printf("Grade D : \n");
      for(ptr_d = head_d; ptr_d != NULL; ptr_d = ptr_d->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_d->roll_no, ptr_d->percentage, ptr_d->grade);
        }
        printf("Grade E : \n");
      for(ptr_e = head_e; ptr_e != NULL; ptr_e = ptr_e->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_e->roll_no, ptr_e->percentage, ptr_e->grade);
        }
        printf("Grade P : \n");
      for(ptr_p = head_p; ptr_p != NULL; ptr_p = ptr_p->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_p->roll_no, ptr_p->percentage, ptr_p->grade);
        }
        printf("Grade F : \n");
      for(ptr_f = head_f; ptr_f != NULL; ptr_f = ptr_f->next){
        printf("Roll number: %d, Percentage : %f and Grade : %c of the student
\n", ptr_f->roll_no, ptr_f->percentage, ptr_f->grade);
      }
      // Freeing the dynamically allocated memory
      freelist(head);
      freelist(head_a);
      freelist(head_b);
      freelist(head_c);
      freelist(head_d);
      freelist(head_e);
      freelist(head_p);
      freelist(head_f);

}
```

Sample Output 1 : CS20B1044 Avinash R Changrani

```
nabobery@nabobery:~/Desktop$ gcc Q3.c
nabobery@nabobery:~/Desktop$ ./a.out
Please enter the size of linked list you want
10
The Students list before being listed grade wise :
Roll number: 1, Percentage : 2.391151 and Grade : F of the student
Roll number: 2, Percentage : 20.644844 and Grade : F of the student
Roll number: 3, Percentage : 83.547714 and Grade : B of the student
Roll number: 4, Percentage : 2.938834 and Grade : F of the student
Roll number: 5, Percentage : 37.937443 and Grade : P of the student
Roll number: 6, Percentage : 24.329275 and Grade : F of the student
Roll number: 7, Percentage : 65.596092 and Grade : D of the student
Roll number: 8, Percentage : 54.883457 and Grade : E of the student
Roll number: 9, Percentage : 67.509811 and Grade : D of the student
Roll number: 10, Percentage : 64.303108 and Grade : D of the student
The Students list after being listed grade wise :
Grade A :
Grade B :
Roll number: 3, Percentage : 83.547714 and Grade : B of the student
Grade C :
Grade D :
Roll number: 7, Percentage : 65.596092 and Grade : D of the student
Roll number: 9, Percentage : 67.509811 and Grade : D of the student
Roll number: 10, Percentage : 64.303108 and Grade : D of the student
Grade E :
Roll number: 8, Percentage : 54.883457 and Grade : E of the student
Grade P :
Roll number: 5, Percentage : 37.937443 and Grade : P of the student
Grade F :
Roll number: 1, Percentage : 2.391151 and Grade : F of the student
Roll number: 2, Percentage : 20.644844 and Grade : F of the student
Roll number: 4, Percentage : 2.938834 and Grade : F of the student
Roll number: 6, Percentage : 24.329275 and Grade : F of the student
nabobery@nabobery:~/Desktop$
```

Sample Output 2 :

```
nabobery@nabobery:~/Desktop$ ./a.out
Please enter the size of linked list you want
15
The Students list before being listed grade wise :
Roll number: 1, Percentage : 38.127457 and Grade : P of the student
Roll number: 2, Percentage : 79.609421 and Grade : C of the student
Roll number: 3, Percentage : 83.994682 and Grade : B of the student
Roll number: 4, Percentage : 2.447652 and Grade : F of the student
Roll number: 5, Percentage : 85.753349 and Grade : B of the student
Roll number: 6, Percentage : 92.004913 and Grade : A of the student
Roll number: 7, Percentage : 81.593163 and Grade : B of the student
Roll number: 8, Percentage : 18.039257 and Grade : F of the student
Roll number: 9, Percentage : 0.450925 and Grade : F of the student
Roll number: 10, Percentage : 30.048954 and Grade : F of the student
Roll number: 11, Percentage : 64.913391 and Grade : D of the student
Roll number: 12, Percentage : 63.209267 and Grade : D of the student
Roll number: 13, Percentage : 40.021992 and Grade : P of the student
Roll number: 14, Percentage : 36.828190 and Grade : P of the student
Roll number: 15, Percentage : 80.058380 and Grade : B of the student
The Students list after being listed grade wise :
Grade A :
Roll number: 6, Percentage : 92.004913 and Grade : A of the student
Grade B :
Roll number: 3, Percentage : 83.994682 and Grade : B of the student
Roll number: 5, Percentage : 85.753349 and Grade : B of the student
Roll number: 7, Percentage : 81.593163 and Grade : B of the student
Roll number: 15, Percentage : 80.058380 and Grade : B of the student
Grade C :
Roll number: 2, Percentage : 79.609421 and Grade : C of the student
Grade D :
Roll number: 11, Percentage : 64.913391 and Grade : D of the student
Roll number: 12, Percentage : 63.209267 and Grade : D of the student
Grade E :
Grade P :
Roll number: 1, Percentage : 38.127457 and Grade : P of the student
Roll number: 13, Percentage : 40.021992 and Grade : P of the student
Roll number: 14, Percentage : 36.828190 and Grade : P of the student
Grade F :
Roll number: 4, Percentage : 2.447652 and Grade : F of the student
Roll number: 8, Percentage : 18.039257 and Grade : F of the student
Roll number: 9, Percentage : 0.450925 and Grade : F of the student
Roll number: 10, Percentage : 30.048954 and Grade : F of the student
nabobery@nabobery:~/Desktop$
```

Complexity Calculation :