Practical Assignment on Disk Scheduling

Write a simulation program for disk scheduling using FCFS algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
55, 58, 39, 18, 90, 160, 150, 38, 184
Start Head Position: 50
Program -

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n, i;
   int total_head_movement = 0;
   int start_head_position;

   // Get the total number of disk blocks
   printf("Enter the total number of disk blocks: ");
   scanf("%d", &n);

   // Create an array for disk requests
   int requests[n];

   // Get the disk request string
   printf("Enter the disk request string:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &requests[i]);
   }

   // Get the start head position
   printf("Enter the start head position: ");
   scanf("%d", &start_head_position);

   // Display the request sequence
   printf("Disk requests to be served in the order:\n");
   printf("%d ", start_head_position); // Start position is also served first

   // Calculate the total head movements and print the order in which requests are served
   int current_position = start_head_position;
   for (i = 0; i < n; i++) {
      total_head_movement += abs(requests[i] - current_position); // Calculate head movement
      printf("%d ", requests[i]); // Display the request
      current_position = requests[i]; // Update current head position
   }

   // Display the total head movements
   printf("\nTotal head movements: %d\n", total_head_movement);
```

```
    return 0;
}
```

Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
86, 147, 91, 170, 95, 130, 102, 70
Starting Head position= 125
Direction: Left
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    int total_head_movement = 0;
    int start_head_position, direction;

    // Get the total number of disk blocks (not used for calculation but input requirement)
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);

    // Create an array for disk requests
    int requests[n];

    // Get the disk request string
    printf("Enter the disk request string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    // Get the start head position
    printf("Enter the start head position: ");
    scanf("%d", &start_head_position);
```

```
// Get the direction (1 for Left, 0 for Right)
printf("Enter the direction (1 for Left, 0 for Right): ");
scanf("%d", &direction);

// Sort the request array
sortRequests(requests, n);

printf("\nDisk requests to be served in the order:\n");

// Handle SCAN algorithm based on direction
if (direction == 1) {  // Left direction
   // First, process all requests from the current head to the leftmost position (0)
   printf("Head moves left: ");
   total_head_movement += abs(start_head_position - requests[0]);
   printf("%d ", start_head_position);  // Display starting position

   for (i = 0; i < n; i++) {
      if (requests[i] < start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }

   // Then, process all requests from the leftmost position to the rightmost
   printf("\nHead moves right: ");
   start_head_position = 0;  // Reset to the leftmost position
   for (i = n - 1; i >= 0; i--) {
      if (requests[i] > start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }
} else {  // Right direction
   // First, process all requests from the current head to the rightmost position
   printf("Head moves right: ");
   total_head_movement += abs(start_head_position - requests[n - 1]);
   printf("%d ", start_head_position);  // Display starting position

   for (i = n - 1; i >= 0; i--) {
      if (requests[i] > start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }
```

```
    // Then, process all requests from the rightmost position to the leftmost
    printf("\nHead moves left: ");
    start_head_position = requests[n - 1];  // Reset to the rightmost position
    for (i = 0; i < n; i++) {
        if (requests[i] < start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
}

// Display the total head movements
printf("\nTotal head movements: %d\n", total_head_movement);

return 0;
}
```

Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
80, 150, 60,135, 40, 35, 170
Starting Head Position: 70
Direction: Right
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    int total_head_movement = 0;
    int start_head_position, direction;

    // Get the total number of disk blocks (not used for calculation but input requirement)
```

```
printf("Enter the total number of disk blocks: ");
scanf("%d", &n);

// Create an array for disk requests
int requests[n];

// Get the disk request string
printf("Enter the disk request string:\n");
for (i = 0; i < n; i++) {
   scanf("%d", &requests[i]);
}

// Get the start head position
printf("Enter the start head position: ");
scanf("%d", &start_head_position);

// Get the direction (1 for Left, 0 for Right)
printf("Enter the direction (1 for Left, 0 for Right): ");
scanf("%d", &direction);

// Sort the request array
sortRequests(requests, n);

printf("\nDisk requests to be served in the order:\n");

if (direction == 0) {  // Right direction
   // First, process all requests from the current head to the rightmost position
   printf("Head moves right: ");
   total_head_movement += abs(start_head_position - requests[n - 1]);
   printf("%d ", start_head_position);  // Display starting position

   // Serve all requests to the right
   for (i = 0; i < n; i++) {
      if (requests[i] >= start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }

   // Now, move the head to the rightmost end (maximum value, assuming total disk blocks size)
   total_head_movement += abs(start_head_position - 199);  // Assumed maximum disk block is 199
   start_head_position = 199;  // Set head position to the rightmost point

   // Then process requests from the leftmost end to the current head position
   for (i = 0; i < n; i++) {
      if (requests[i] < start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
```

```
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
} else {  // Left direction
    // First, process all requests from the current head to the leftmost position
    printf("Head moves left: ");
    total_head_movement += abs(start_head_position - requests[0]);
    printf("%d ", start_head_position);  // Display starting position

    // Serve all requests to the left
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] <= start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

    // Now, move the head to the leftmost end (0)
    total_head_movement += abs(start_head_position - 0);  // Move to position 0
    start_head_position = 0;  // Set head position to the leftmost point

    // Then process requests from the rightmost end to the current head position
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] > start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
}

// Display the total head movements
printf("\nTotal head movements: %d\n", total_head_movement);

return 0;
}
```

Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
82, 170, 43, 140, 24, 16, 190
Starting Head Position: 50
Direction: Right
Program-

```
#include <stdio.h>
#include <stdlib.h>
```

```
void sortRequests(int requests[], int n) {
   int temp;
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (requests[i] > requests[j]) {
            temp = requests[i];
            requests[i] = requests[j];
            requests[j] = temp;
         }
      }
   }
}

int main() {
   int n, i;
   int total_head_movement = 0;
   int start_head_position, direction;

   // Get the total number of disk blocks (not used for calculation but input requirement)
   printf("Enter the total number of disk blocks: ");
   scanf("%d", &n);

   // Create an array for disk requests
   int requests[n];

   // Get the disk request string
   printf("Enter the disk request string:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &requests[i]);
   }

   // Get the start head position
   printf("Enter the start head position: ");
   scanf("%d", &start_head_position);

   // Get the direction (1 for Left, 0 for Right)
   printf("Enter the direction (1 for Left, 0 for Right): ");
   scanf("%d", &direction);

   // Sort the request array
   sortRequests(requests, n);

   printf("\nDisk requests to be served in the order:\n");

   if (direction == 0) {  // Right direction
      // First, process all requests from the current head to the rightmost position
      printf("Head moves right: ");
```

```
    total_head_movement += abs(start_head_position - requests[n - 1]);
    printf("%d ", start_head_position);  // Display starting position

    // Serve all requests to the right
    for (i = 0; i < n; i++) {
        if (requests[i] >= start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

    // Now, after serving to the rightmost, we reverse direction and serve to the left
    // Move the head to the rightmost end (max block number is assumed to be 199)
    total_head_movement += abs(start_head_position - 199);  // Assumed maximum disk block is 199
    start_head_position = 199;  // Set head position to the rightmost point

    // Then process requests from the rightmost end to the leftmost (in reverse order)
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] < start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
} else {  // Left direction
    // First, process all requests from the current head to the leftmost position
    printf("Head moves left: ");
    total_head_movement += abs(start_head_position - requests[0]);
    printf("%d ", start_head_position);  // Display starting position

    // Serve all requests to the left
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] <= start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

    // Now, move the head to the leftmost end (0)
    total_head_movement += abs(start_head_position - 0);  // Move to position 0
    start_head_position = 0;  // Set head position to the leftmost point

    // Then process requests from the rightmost end to the current head position
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] > start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
```

```
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
      }
   }

   // Display the total head movements
   printf("\nTotal head movements: %d\n", total_head_movement);

   return 0;
}
```

Write a simulation program for disk scheduling using SSTF algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
186, 89, 44, 70, 102, 22, 51, 124
Start Head Position: 70
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sortRequests(int requests[], int n) {
   int temp;
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (requests[i] > requests[j]) {
            temp = requests[i];
            requests[i] = requests[j];
            requests[j] = temp;
         }
      }
   }
}

int main() {
   int n, i;
   int total_head_movement = 0;
   int start_head_position;

   // Get the total number of disk blocks (not used for calculation but input requirement)
   printf("Enter the total number of disk blocks: ");
   scanf("%d", &n);

   // Create an array for disk requests
   int requests[n];

   // Get the disk request string
```

```
printf("Enter the disk request string:\n");
for (i = 0; i < n; i++) {
   scanf("%d", &requests[i]);
}

// Get the start head position
printf("Enter the start head position: ");
scanf("%d", &start_head_position);

// SSTF Algorithm Logic
int visited[n];  // Array to track visited requests
for (i = 0; i < n; i++) {
   visited[i] = 0;
}

int total_requests = n;
int current_position = start_head_position;
int request_order[n];
int served_count = 0;

printf("\nDisk requests to be served in the order:\n");

while (served_count < total_requests) {
   int min_distance = 1000000;  // A large number to find the closest request
   int closest_request_index = -1;

   // Find the closest request that hasn't been served
   for (i = 0; i < n; i++) {
      if (visited[i] == 0) {
         int distance = abs(current_position - requests[i]);
         if (distance < min_distance) {
            min_distance = distance;
            closest_request_index = i;
         }
      }
   }

   // Serve the closest request
   visited[closest_request_index] = 1;
   request_order[served_count] = requests[closest_request_index];
   total_head_movement += min_distance;
   current_position = requests[closest_request_index];

   printf("%d ", requests[closest_request_index]);

   served_count++;
}
```

```
    // Display the total head movements
    printf("\nTotal head movements: %d\n", total_head_movement);

    return 0;
}
```

Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
176, 79, 34, 60, 92, 11, 41, 114
Starting Head Position: 65
Direction: Left
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    int total_head_movement = 0;
    int start_head_position, direction;

    // Get the total number of disk blocks (not used for calculation but input requirement)
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);

    // Create an array for disk requests
    int requests[n];

    // Get the disk request string
    printf("Enter the disk request string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    // Get the start head position
```

```
printf("Enter the start head position: ");
scanf("%d", &start_head_position);

// Get the direction (1 for Left, 0 for Right)
printf("Enter the direction (1 for Left, 0 for Right): ");
scanf("%d", &direction);

// Sort the request array in ascending order
sortRequests(requests, n);

printf("\nDisk requests to be served in the order:\n");

if (direction == 1) {  // Left direction
   // First, process all requests to the left of the current head position
   printf("Head moves left: ");
   total_head_movement += abs(start_head_position - requests[0]);
   printf("%d ", start_head_position);  // Display starting position

   // Serve all requests to the left
   for (i = n - 1; i >= 0; i--) {
      if (requests[i] <= start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }

   // After serving to the leftmost request, reverse direction to serve requests to the right
   for (i = 0; i < n; i++) {
      if (requests[i] > start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }
} else {  // Right direction
   // First, process all requests to the right of the current head position
   printf("Head moves right: ");
   total_head_movement += abs(start_head_position - requests[n - 1]);
   printf("%d ", start_head_position);  // Display starting position

   // Serve all requests to the right
   for (i = 0; i < n; i++) {
      if (requests[i] >= start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
```

```
        }

        // After serving to the rightmost request, reverse direction to serve requests to the left
        for (i = n - 1; i >= 0; i--) {
            if (requests[i] < start_head_position) {
                total_head_movement += abs(start_head_position - requests[i]);
                printf("%d ", requests[i]);
                start_head_position = requests[i];
            }
        }
    }

    // Display the total head movements
    printf("\nTotal head movements: %d\n", total_head_movement);

    return 0;
}
```

Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
33, 99, 142, 52, 197, 79, 46, 65
Start Head Position: 72
Direction: Left
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    int total_head_movement = 0;
    int start_head_position, direction;

    // Get the total number of disk blocks (not used for calculation but input requirement)
    printf("Enter the total number of disk blocks: ");
```

```
scanf("%d", &n);

// Create an array for disk requests
int requests[n];

// Get the disk request string
printf("Enter the disk request string:\n");
for (i = 0; i < n; i++) {
   scanf("%d", &requests[i]);
}

// Get the start head position
printf("Enter the start head position: ");
scanf("%d", &start_head_position);

// Get the direction (1 for Left, 0 for Right)
printf("Enter the direction (1 for Left, 0 for Right): ");
scanf("%d", &direction);

// Sort the request array
sortRequests(requests, n);

printf("\nDisk requests to be served in the order:\n");

if (direction == 1) {  // Left direction
   // First, process all requests to the left of the current head position
   printf("Head moves left: ");
   total_head_movement += abs(start_head_position - requests[0]);
   printf("%d ", start_head_position);  // Display starting position

   // Serve all requests to the left
   for (i = n - 1; i >= 0; i--) {
      if (requests[i] <= start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }

   // After serving to the leftmost request, jump to the rightmost end and serve to the right
   total_head_movement += abs(start_head_position - 199);  // Assumed maximum disk block is 199
   start_head_position = 199;  // Set head position to the rightmost point

   // Now serve the remaining requests from the rightmost point to the left
   for (i = 0; i < n; i++) {
      if (requests[i] > start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
```

```
            start_head_position = requests[i];
        }
    }
} else {  // Right direction
    // First, process all requests to the right of the current head position
    printf("Head moves right: ");
    total_head_movement += abs(start_head_position - requests[n - 1]);
    printf("%d ", start_head_position);  // Display starting position

    // Serve all requests to the right
    for (i = 0; i < n; i++) {
        if (requests[i] >= start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

    // After serving to the rightmost request, jump to the leftmost end (0) and serve to the right
    total_head_movement += abs(start_head_position - 0);  // Assumed leftmost block is 0
    start_head_position = 0;  // Set head position to the leftmost point

    // Then serve the remaining requests from the leftmost end to the right
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] < start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
}

// Display the total head movements
printf("\nTotal head movements: %d\n", total_head_movement);

return 0;
}
```

Write a simulation program for disk scheduling using C-LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
23, 89, 132, 42, 187, 69, 36, 55
Start Head Position: 40
Direction: Right
Program-

```
#include <stdio.h>
#include <stdlib.h>
```

```
void sortRequests(int requests[], int n) {
   int temp;
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (requests[i] > requests[j]) {
            temp = requests[i];
            requests[i] = requests[j];
            requests[j] = temp;
         }
      }
   }
}

int main() {
   int n, i;
   int total_head_movement = 0;
   int start_head_position, direction;

   // Get the total number of disk blocks (not used for calculation but input requirement)
   printf("Enter the total number of disk blocks: ");
   scanf("%d", &n);

   // Create an array for disk requests
   int requests[n];

   // Get the disk request string
   printf("Enter the disk request string:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &requests[i]);
   }

   // Get the start head position
   printf("Enter the start head position: ");
   scanf("%d", &start_head_position);

   // Get the direction (1 for Left, 0 for Right)
   printf("Enter the direction (1 for Left, 0 for Right): ");
   scanf("%d", &direction);

   // Sort the request array
   sortRequests(requests, n);

   printf("\nDisk requests to be served in the order:\n");

   if (direction == 0) {  // Right direction
      // First, process all requests greater than or equal to the current head position
      printf("Head moves right: ");
      total_head_movement += abs(start_head_position - requests[0]);
```

```
    printf("%d ", start_head_position);  // Display starting position

    // Serve all requests to the right
    for (i = 0; i < n; i++) {
        if (requests[i] >= start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

    // After serving requests to the right, reverse direction and serve remaining requests to the left
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] < start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

} else {  // Left direction
    // First, process all requests smaller than or equal to the current head position
    printf("Head moves left: ");
    total_head_movement += abs(start_head_position - requests[n - 1]);
    printf("%d ", start_head_position);  // Display starting position

    // Serve all requests to the left
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] <= start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }

    // After serving requests to the left, reverse direction and serve remaining requests to the right
    for (i = 0; i < n; i++) {
        if (requests[i] > start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
}

// Display the total head movements
printf("\nTotal head movements: %d\n", total_head_movement);
```

```
    return 0;
}
```

Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
176, 79, 34, 60, 92, 11, 41, 114
Starting Head Position: 65
Direction: Left
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sortRequests(int requests[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (requests[i] > requests[j]) {
                temp = requests[i];
                requests[i] = requests[j];
                requests[j] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    int total_head_movement = 0;
    int start_head_position, direction;

    // Get the total number of disk blocks (not used for calculation but input requirement)
    printf("Enter the total number of disk blocks: ");
    scanf("%d", &n);

    // Create an array for disk requests
    int requests[n];

    // Get the disk request string
    printf("Enter the disk request string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    // Get the start head position
    printf("Enter the start head position: ");
    scanf("%d", &start_head_position);
```

```
// Get the direction (1 for Left, 0 for Right)
printf("Enter the direction (1 for Left, 0 for Right): ");
scanf("%d", &direction);

// Sort the request array
sortRequests(requests, n);

printf("\nDisk requests to be served in the order:\n");

if (direction == 1) {  // Left direction
   // First, process all requests to the left of the current head position
   printf("Head moves left: ");
   total_head_movement += abs(start_head_position - requests[0]);
   printf("%d ", start_head_position);  // Display starting position

   // Serve all requests to the left
   for (i = n - 1; i >= 0; i--) {
      if (requests[i] <= start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }

   // After serving to the leftmost request, reverse direction to serve requests to the right
   for (i = 0; i < n; i++) {
      if (requests[i] > start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }
} else {  // Right direction
   // First, process all requests to the right of the current head position
   printf("Head moves right: ");
   total_head_movement += abs(start_head_position - requests[n - 1]);
   printf("%d ", start_head_position);  // Display starting position

   // Serve all requests to the right
   for (i = 0; i < n; i++) {
      if (requests[i] >= start_head_position) {
         total_head_movement += abs(start_head_position - requests[i]);
         printf("%d ", requests[i]);
         start_head_position = requests[i];
      }
   }

   // After serving to the rightmost request, reverse direction to serve requests to the left
```

```
    for (i = n - 1; i >= 0; i--) {
        if (requests[i] < start_head_position) {
            total_head_movement += abs(start_head_position - requests[i]);
            printf("%d ", requests[i]);
            start_head_position = requests[i];
        }
    }
}

// Display the total head movements
printf("\nTotal head movements: %d\n", total_head_movement);

return 0;
}
```

Write a simulation program for disk scheduling using SSTF algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.

55, 58, 39, 18, 90, 160, 150, 38, 184

Start Head Position: 50

Program-

```
#include <stdio.h>
#include <stdlib.h>

void find_and_serve_nearest_request(int request[], int n, int* current_position, int*
total_head_movement) {
    int nearest_request_index = -1;
    int min_distance = 999999;  // A large number to start with

    // Find the request closest to the current head position
    for (int i = 0; i < n; i++) {
        if (request[i] != -1) {
            int distance = abs(request[i] - *current_position);
            if (distance < min_distance) {
                min_distance = distance;
                nearest_request_index = i;
            }
        }
    }

    // Serve the nearest request
    *total_head_movement += min_distance;
    *current_position = request[nearest_request_index];
    request[nearest_request_index] = -1;  // Mark this request as served
}

int main() {
```

```
    int n, i, total_head_movement = 0, current_position;

    // Accept total number of disk blocks, request string, and current head position
    printf("Enter the total number of requests: ");
    scanf("%d", &n);

    int request[n];

    printf("Enter the disk request string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &request[i]);
    }

    printf("Enter the start head position: ");
    scanf("%d", &current_position);

    // Display the original request string
    printf("\nOriginal request string: ");
    for (i = 0; i < n; i++) {
        printf("%d ", request[i]);
    }

    // Process the requests using SSTF algorithm
    printf("\nOrder of requests served: ");
    while (1) {
        int served_requests = 0;

        // Find and serve the nearest request
        for (i = 0; i < n; i++) {
            if (request[i] != -1) {
                find_and_serve_nearest_request(request, n, &current_position, &total_head_movement);
                printf("%d ", current_position);
                served_requests = 1;
                break;
            }
        }

        if (served_requests == 0) {
            break;  // All requests are served
        }
    }

    // Display total head movement
    printf("\nTotal head movements: %d\n", total_head_movement);

    return 0;
}
```

Write a simulation program for disk scheduling using C-SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
80, 150, 60,135, 40, 35, 170
Starting Head Position: 70
Direction: Right
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sort_requests(int request[], int n) {
   int temp;
   // Sorting the requests in ascending order
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (request[i] > request[j]) {
            temp = request[i];
            request[i] = request[j];
            request[j] = temp;
         }
      }
   }
}

int main() {
   int n, i, total_head_movement = 0, current_position, direction;

   // Accept total number of disk blocks, request string, and current head position
   printf("Enter the total number of requests: ");
   scanf("%d", &n);

   int request[n];

   printf("Enter the disk request string: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &request[i]);
   }

   printf("Enter the start head position: ");
   scanf("%d", &current_position);

   printf("Enter the direction (1 for right, 0 for left): ");
   scanf("%d", &direction);

   // Sort the requests in ascending order
   sort_requests(request, n);
```

```
// Display the sorted request array
printf("\nSorted requests: ");
for (i = 0; i < n; i++) {
   printf("%d ", request[i]);
}

// Variables to track the requests that will be served
int right_requests[n], left_requests[n];
int right_count = 0, left_count = 0;

// Split requests into two arrays (right and left of the head)
for (i = 0; i < n; i++) {
   if (request[i] > current_position) {
      right_requests[right_count++] = request[i];
   } else {
      left_requests[left_count++] = request[i];
   }
}

total_head_movement = 0;
int previous_position = current_position;

// Process requests using C-SCAN algorithm
if (direction == 1) { // Right direction
   // Serve all requests to the right
   for (i = 0; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
   }

   // After reaching the farthest right, go back to the leftmost side
   total_head_movement += abs(previous_position - left_requests[left_count - 1]);
   previous_position = left_requests[left_count - 1];

   // Serve all requests to the left
   for (i = left_count - 2; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
   }
}
else { // Left direction
   // Serve all requests to the left
   for (i = left_count - 1; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
   }

   // After reaching the farthest left, go back to the rightmost side
```

```
        total_head_movement += abs(previous_position - right_requests[0]);
        previous_position = right_requests[0];

        // Serve all requests to the right
        for (i = 1; i < right_count; i++) {
            total_head_movement += abs(right_requests[i] - previous_position);
            previous_position = right_requests[i];
        }
    }

    // Display the order in which requests are served
    printf("\nOrder of requests served: ");
    if (direction == 1) { // Right direction first
        for (i = 0; i < right_count; i++) {
            printf("%d ", right_requests[i]);
        }
        for (i = left_count - 1; i >= 0; i--) {
            printf("%d ", left_requests[i]);
        }
    } else { // Left direction first
        for (i = left_count - 1; i >= 0; i--) {
            printf("%d ", left_requests[i]);
        }
        for (i = 0; i < right_count; i++) {
            printf("%d ", right_requests[i]);
        }
    }

    // Display total head movement
    printf("\nTotal head movements: %d\n", total_head_movement);

    return 0;
}
```

Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
23, 89, 132, 42, 187, 69, 36, 55
Start Head Position: 40
Direction: Left
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sort_requests(int request[], int n) {
    int temp;
```

```
    // Sorting the requests in ascending order
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (request[i] > request[j]) {
                temp = request[i];
                request[i] = request[j];
                request[j] = temp;
            }
        }
    }
}

int main() {
    int n, i, total_head_movement = 0, current_position, direction;

    // Accept total number of disk blocks, request string, and current head position
    printf("Enter the total number of requests: ");
    scanf("%d", &n);

    int request[n];

    printf("Enter the disk request string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &request[i]);
    }

    printf("Enter the start head position: ");
    scanf("%d", &current_position);

    printf("Enter the direction (1 for right, 0 for left): ");
    scanf("%d", &direction);

    // Sort the requests in ascending order
    sort_requests(request, n);

    // Display the sorted request array
    printf("\nSorted requests: ");
    for (i = 0; i < n; i++) {
        printf("%d ", request[i]);
    }

    // Variables to track the requests that will be served
    int right_requests[n], left_requests[n];
    int right_count = 0, left_count = 0;

    // Split requests into two arrays (right and left of the head)
    for (i = 0; i < n; i++) {
        if (request[i] > current_position) {
```

```
        right_requests[right_count++] = request[i];
    } else {
        left_requests[left_count++] = request[i];
    }
}

total_head_movement = 0;
int previous_position = current_position;

// Process requests using LOOK algorithm
if (direction == 1) { // Right direction
    // Serve all requests to the right
    for (i = 0; i < right_count; i++) {
        total_head_movement += abs(right_requests[i] - previous_position);
        previous_position = right_requests[i];
    }

    // After serving right requests, reverse direction and serve left requests
    for (i = left_count - 1; i >= 0; i--) {
        total_head_movement += abs(left_requests[i] - previous_position);
        previous_position = left_requests[i];
    }
}
else { // Left direction
    // Serve all requests to the left
    for (i = left_count - 1; i >= 0; i--) {
        total_head_movement += abs(left_requests[i] - previous_position);
        previous_position = left_requests[i];
    }

    // After serving left requests, reverse direction and serve right requests
    for (i = 0; i < right_count; i++) {
        total_head_movement += abs(right_requests[i] - previous_position);
        previous_position = right_requests[i];
    }
}

// Display the order in which requests are served
printf("\nOrder of requests served: ");
if (direction == 1) { // Right direction first
    for (i = 0; i < right_count; i++) {
        printf("%d ", right_requests[i]);
    }
    for (i = left_count - 1; i >= 0; i--) {
        printf("%d ", left_requests[i]);
    }
} else { // Left direction first
    for (i = left_count - 1; i >= 0; i--) {
```

```
        printf("%d ", left_requests[i]);
    }
    for (i = 0; i < right_count; i++) {
        printf("%d ", right_requests[i]);
    }
}

// Display total head movement
printf("\nTotal head movements: %d\n", total_head_movement);

return 0;
}
```

Write a simulation program for disk scheduling using SCAN algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
33, 99, 142, 52, 197, 79, 46, 65
Start Head Position: 72
Direction: Right
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sort_requests(int request[], int n) {
    int temp;
    // Sorting the requests in ascending order
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (request[i] > request[j]) {
                temp = request[i];
                request[i] = request[j];
                request[j] = temp;
            }
        }
    }
}

int main() {
    int n, i, total_head_movement = 0, current_position, direction;

    // Accept total number of disk blocks, request string, and current head position
    printf("Enter the total number of requests: ");
    scanf("%d", &n);

    int request[n];
```

```
printf("Enter the disk request string: ");
for (i = 0; i < n; i++) {
   scanf("%d", &request[i]);
}

printf("Enter the start head position: ");
scanf("%d", &current_position);

printf("Enter the direction (1 for right, 0 for left): ");
scanf("%d", &direction);

// Sort the requests in ascending order
sort_requests(request, n);

// Display the sorted request array
printf("\nSorted requests: ");
for (i = 0; i < n; i++) {
   printf("%d ", request[i]);
}

// Variables to track the requests that will be served
int right_requests[n], left_requests[n];
int right_count = 0, left_count = 0;

// Split requests into two arrays (right and left of the head)
for (i = 0; i < n; i++) {
   if (request[i] > current_position) {
      right_requests[right_count++] = request[i];
   } else {
      left_requests[left_count++] = request[i];
   }
}

total_head_movement = 0;
int previous_position = current_position;

// Process requests using SCAN algorithm
if (direction == 1) { // Right direction
   // Serve all requests to the right
   for (i = 0; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
   }

   // After reaching the farthest right, reverse direction and serve left requests
   for (i = left_count - 1; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
```

```
    }
  }
  else { // Left direction
    // Serve all requests to the left
    for (i = left_count - 1; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
    }

    // After reaching the farthest left, reverse direction and serve right requests
    for (i = 0; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
    }
  }

  // Display the order in which requests are served
  printf("\nOrder of requests served: ");
  if (direction == 1) { // Right direction first
    for (i = 0; i < right_count; i++) {
      printf("%d ", right_requests[i]);
    }
    for (i = left_count - 1; i >= 0; i--) {
      printf("%d ", left_requests[i]);
    }
  } else { // Left direction first
    for (i = left_count - 1; i >= 0; i--) {
      printf("%d ", left_requests[i]);
    }
    for (i = 0; i < right_count; i++) {
      printf("%d ", right_requests[i]);
    }
  }

  // Display total head movement
  printf("\nTotal head movements: %d\n", total_head_movement);

  return 0;
}
```

Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
176, 79, 34, 60, 92, 11, 41, 114
Starting Head Position: 65
Direction: Right

Program-

```
#include <stdio.h>
#include <stdlib.h>

void sort_requests(int request[], int n) {
   int temp;
   // Sorting the requests in ascending order
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (request[i] > request[j]) {
            temp = request[i];
            request[i] = request[j];
            request[j] = temp;
         }
      }
   }
}

int main() {
   int n, i, total_head_movement = 0, current_position, direction;

   // Accept total number of disk blocks, request string, and current head position
   printf("Enter the total number of requests: ");
   scanf("%d", &n);

   int request[n];

   printf("Enter the disk request string: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &request[i]);
   }

   printf("Enter the start head position: ");
   scanf("%d", &current_position);

   printf("Enter the direction (0 for left, 1 for right): ");
   scanf("%d", &direction);

   // Sort the requests in ascending order
   sort_requests(request, n);

   // Display the sorted request array
   printf("\nSorted requests: ");
   for (i = 0; i < n; i++) {
      printf("%d ", request[i]);
   }

   // Variables to track the requests that will be served
```

```
int right_requests[n], left_requests[n];
int right_count = 0, left_count = 0;

// Split requests into two arrays (right and left of the head)
for (i = 0; i < n; i++) {
   if (request[i] > current_position) {
      right_requests[right_count++] = request[i];
   } else {
      left_requests[left_count++] = request[i];
   }
}

total_head_movement = 0;
int previous_position = current_position;

// Process requests in the direction specified by the user
if (direction == 1) { // Right direction
   // Serve all requests to the right
   for (i = 0; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
   }

   // After reaching the farthest right, reverse the direction and serve left requests
   total_head_movement += abs(previous_position - left_requests[left_count - 1]);
   previous_position = left_requests[left_count - 1];

   // Serve all requests to the left
   for (i = left_count - 2; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
   }
}
else { // Left direction
   // Serve all requests to the left
   for (i = left_count - 1; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
   }

   // After reaching the farthest left, reverse the direction and serve right requests
   total_head_movement += abs(previous_position - right_requests[0]);
   previous_position = right_requests[0];

   // Serve all requests to the right
   for (i = 1; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
```

```
        }
    }

    // Display the order in which requests are served
    printf("\nOrder of requests served: ");
    if (direction == 1) { // Right direction first
        for (i = 0; i < right_count; i++) {
            printf("%d ", right_requests[i]);
        }
        for (i = left_count - 1; i >= 0; i--) {
            printf("%d ", left_requests[i]);
        }
    } else { // Left direction first
        for (i = left_count - 1; i >= 0; i--) {
            printf("%d ", left_requests[i]);
        }
        for (i = 0; i < right_count; i++) {
            printf("%d ", right_requests[i]);
        }
    }

    // Display total head movement
    printf("\nTotal head movements: %d\n", total_head_movement);

    return 0;
}
```

Write a simulation program for disk scheduling using C-LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
56, 59, 40, 19, 91, 161, 151, 39, 185
Start Head Position: 48
Direction: User Defined
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sort_requests(int request[], int n) {
    int temp;
    // Sorting the requests in ascending order
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (request[i] > request[j]) {
                temp = request[i];
                request[i] = request[j];
                request[j] = temp;
```

```
            }
        }
    }
}

int main() {
    int n, i, total_head_movement = 0, current_position, direction;

    // Accept total number of disk blocks, request string, and current head position
    printf("Enter the total number of requests: ");
    scanf("%d", &n);

    int request[n];

    printf("Enter the disk request string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &request[i]);
    }

    printf("Enter the start head position: ");
    scanf("%d", &current_position);

    printf("Enter the direction (0 for left, 1 for right): ");
    scanf("%d", &direction);

    // Sort the requests in ascending order
    sort_requests(request, n);

    // Display the sorted request array
    printf("\nSorted requests: ");
    for (i = 0; i < n; i++) {
        printf("%d ", request[i]);
    }

    // Variables to track the requests that will be served
    int right_requests[n], left_requests[n];
    int right_count = 0, left_count = 0;

    // Split requests into two arrays (right and left of the head)
    for (i = 0; i < n; i++) {
        if (request[i] > current_position) {
            right_requests[right_count++] = request[i];
        } else {
            left_requests[left_count++] = request[i];
        }
    }

    total_head_movement = 0;
```

```
int previous_position = current_position;

// Process requests in the direction specified by user
if (direction == 1) { // Right direction
   // Serve all requests to the right
   for (i = 0; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
   }

   // After reaching the farthest right, jump to the smallest left request
   total_head_movement += abs(previous_position - left_requests[left_count - 1]);
   previous_position = left_requests[left_count - 1];

   // Serve all requests to the left
   for (i = left_count - 2; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
   }
}
else { // Left direction
   // Serve all requests to the left
   for (i = left_count - 1; i >= 0; i--) {
      total_head_movement += abs(left_requests[i] - previous_position);
      previous_position = left_requests[i];
   }

   // After reaching the farthest left, jump to the largest right request
   total_head_movement += abs(previous_position - right_requests[0]);
   previous_position = right_requests[0];

   // Serve all requests to the right
   for (i = 1; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
   }
}

// Display the order in which requests are served
printf("\nOrder of requests served: ");
if (direction == 1) { // Right direction first
   for (i = 0; i < right_count; i++) {
      printf("%d ", right_requests[i]);
   }
   for (i = left_count - 1; i >= 0; i--) {
      printf("%d ", left_requests[i]);
   }
} else { // Left direction first
```

```
    for (i = left_count - 1; i >= 0; i--) {
        printf("%d ", left_requests[i]);
    }
    for (i = 0; i < right_count; i++) {
        printf("%d ", right_requests[i]);
    }
}

// Display total head movement
printf("\nTotal head movements: %d\n", total_head_movement);

return 0;
}
```

Write a simulation program for disk scheduling using C-LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.. [15]
80, 150, 60,135, 40, 35, 170
Starting Head Position: 70
Direction: Right
Program-

```
#include <stdio.h>
#include <stdlib.h>

void sort_requests(int request[], int n) {
    int temp;
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (request[i] > request[j]) {
                // Swap the elements
                temp = request[i];
                request[i] = request[j];
                request[j] = temp;
            }
        }
    }
}

int main() {
    int n, i, total_head_movement = 0, current_position, previous_position, direction;

    // Accept total number of disk blocks, request string, and current head position
    printf("Enter the total number of requests: ");
    scanf("%d", &n);

    int request[n];
```

```
printf("Enter the disk request string: ");
for (i = 0; i < n; i++) {
   scanf("%d", &request[i]);
}

printf("Enter the start head position: ");
scanf("%d", &current_position);

printf("Enter the direction (0 for left, 1 for right): ");
scanf("%d", &direction);

// Sort the requests in ascending order
sort_requests(request, n);

// Display the sorted request array
printf("\nSorted requests: ");
for (i = 0; i < n; i++) {
   printf("%d ", request[i]);
}

// Variables to track the requests that will be served
int right_requests[n], left_requests[n];
int right_count = 0, left_count = 0;

// Split requests into two arrays (right and left of the head)
for (i = 0; i < n; i++) {
   if (request[i] > current_position) {
      right_requests[right_count++] = request[i];
   } else {
      left_requests[left_count++] = request[i];
   }
}

total_head_movement = 0;
previous_position = current_position;

// Process requests in the right direction first
if (direction == 1) { // Right direction
   // Serve all requests to the right
   for (i = 0; i < right_count; i++) {
      total_head_movement += abs(right_requests[i] - previous_position);
      previous_position = right_requests[i];
   }

   // After reaching the farthest right, go to the smallest left request
   total_head_movement += abs(previous_position - left_requests[left_count - 1]);
   previous_position = left_requests[left_count - 1];
```

```
        // Serve all requests to the left
        for (i = left_count - 2; i >= 0; i--) {
            total_head_movement += abs(left_requests[i] - previous_position);
            previous_position = left_requests[i];
        }
    }
    else { // Left direction
        // Serve all requests to the left
        for (i = left_count - 1; i >= 0; i--) {
            total_head_movement += abs(left_requests[i] - previous_position);
            previous_position = left_requests[i];
        }

        // After reaching the farthest left, go to the largest right request
        total_head_movement += abs(previous_position - right_requests[0]);
        previous_position = right_requests[0];

        // Serve all requests to the right
        for (i = 1; i < right_count; i++) {
            total_head_movement += abs(right_requests[i] - previous_position);
            previous_position = right_requests[i];
        }
    }

    // Display the order in which requests are served
    printf("\nOrder of requests served: ");
    if (direction == 1) { // Right direction first
        for (i = 0; i < right_count; i++) {
            printf("%d ", right_requests[i]);
        }
        for (i = left_count - 1; i >= 0; i--) {
            printf("%d ", left_requests[i]);
        }
    } else { // Left direction first
        for (i = left_count - 1; i >= 0; i--) {
            printf("%d ", left_requests[i]);
        }
        for (i = 0; i < right_count; i++) {
            printf("%d ", right_requests[i]);
        }
    }

    // Display total head movement
    printf("\nTotal head movements: %d\n", total_head_movement);

    return 0;
}
```

Write a simulation program for disk scheduling using FCFS algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.
65, 95, 30, 91, 18, 116, 142, 44, 168
Start Head Position: 52
Program-

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n, i, total_head_movement = 0, current_position, previous_position;

   // Accept total number of disk blocks, request string, and current head position
   printf("Enter the total number of requests: ");
   scanf("%d", &n);

   int request[n];

   printf("Enter the disk request string: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &request[i]);
   }

   printf("Enter the start head position: ");
   scanf("%d", &current_position);

   previous_position = current_position;

   // Display the requests in the order they are served
   printf("\nOrder of requests served: ");
   for (i = 0; i < n; i++) {
      printf("%d ", request[i]);

      // Calculate the head movement for each request
      total_head_movement += abs(request[i] - previous_position);
      previous_position = request[i];
   }

   // Display total head movement
   printf("\nTotal head movements: %d\n", total_head_movement);

   return 0;
}
```