**Assignment 9**

**Name** : Komal Potdar

**Roll No**.: 92

**PRN No**.: 12320165

**Div**: CS B SY

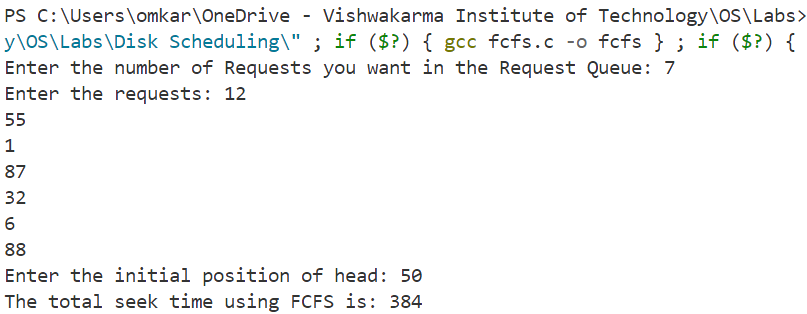
**Batch**: 3

**Implement disk scheduling Algorithm**

1. **FIFO**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h> // for abs()  int fcfs\_diskScheduling(int \*requests, int n, int initial\_Position) {      int total\_seek\_time = 0;      int current\_Position = initial\_Position;      for (int i = 0; i < n; i++) {          int seek\_distance = abs(requests[i] - current\_Position);          total\_seek\_time += seek\_distance;          current\_Position = requests[i];      }      return total\_seek\_time;  }  int main() {      int n;      printf("Enter the number of Requests you want in the Request Queue: ");      scanf("%d", &n);      int \*requests = (int \*)malloc(n \* sizeof(int));      if (requests == NULL) {          printf("Memory allocation failed!\n");          return 1;      }      printf("Enter the requests: ");      for (int i = 0; i < n; i++) {          scanf("%d", &requests[i]);      }      int initial\_Position;      printf("Enter the initial position of head: ");      scanf("%d", &initial\_Position);      int seek\_time = fcfs\_diskScheduling(requests, n, initial\_Position);      printf("The total seek time using FCFS is: %d\n", seek\_time);      free(requests); // Free dynamically allocated memory      return 0;  } |

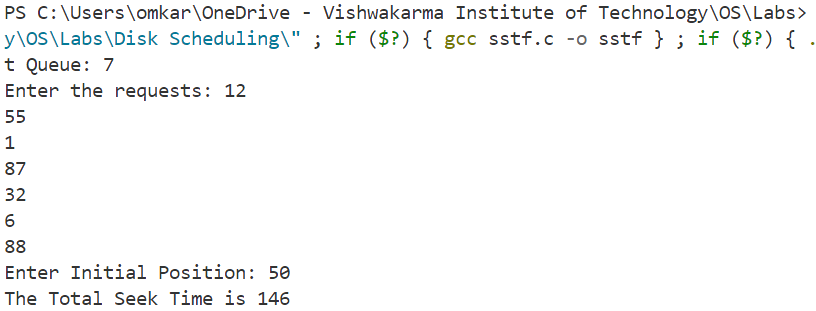
**Output:**



1. **SSTF**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int find\_closest\_request(int \*requests, int n, int current\_Position)  {      int min\_Distance = 999;      int closest\_Request = -1;      for (int i = 0; i < n; i++)      {          int seek\_Distance = abs(requests[i] - current\_Position);          if (seek\_Distance < min\_Distance)          {              min\_Distance = seek\_Distance;              closest\_Request = requests[i];          }      }      return closest\_Request;  }  int sstf(int \*requests, int n, int initial\_Position)  {      int total\_seek\_time = 0;      int current\_Position = initial\_Position;      while(n > 0)      {          int closest\_Request = find\_closest\_request(requests, n, current\_Position);          int seek\_distance = abs(closest\_Request - current\_Position);          total\_seek\_time += seek\_distance;          current\_Position = closest\_Request;          // Remove the processed request from the array          for (int i = 0; i < n; i++) {              if (requests[i] == closest\_Request) {                  // Shift elements to left to overwrite the processed request                  for (int j = i; j < n - 1; j++) {                      requests[j] = requests[j + 1];                  }                  n--; // Decrement the count of requests                  break; // Exit the loop once the request is found and removed              }          }      }      return total\_seek\_time;  }  int main()  {      int n;      printf("Enter the number of requests you want in the Request Queue: ");      scanf("%d", &n);      int \*requests = (int \*)malloc(n \* sizeof(int));      if (requests == NULL) {          printf("Memory allocation failed!\n");          return 1;      }      printf("Enter the requests: ");      for (int i = 0; i < n; i++) {          scanf("%d", &requests[i]);      }      int initial\_Position;      printf("Enter Initial Position: ");      scanf("%d", &initial\_Position);      int seekTime = sstf(requests, n, initial\_Position);      printf("The Total Seek Time is %d\n", seekTime);      free(requests);      return 0;  } |

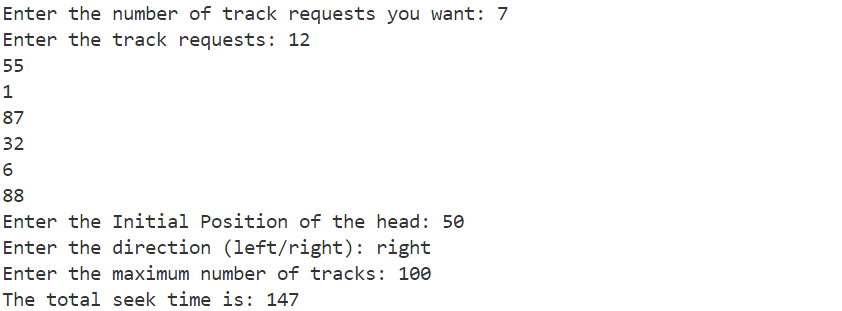
**Output:**



1. **SCAN**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int compare(const void \*a, const void \*b) {      return (\*(int\*)a - \*(int\*)b);  }  int scan(int \*requests, int n, int initial\_Position, char \*direction, int track\_size)  {      int total\_seek\_time = 0;      int current\_Position = initial\_Position;      int \*left = NULL;      int \*right = NULL;      int left\_count = 0, right\_count = 0;      // Allocate memory for left and right arrays      left = (int \*)malloc(n \* sizeof(int));      if (left == NULL) {          printf("Memory allocation failed!\n");          return -1;      }      right = (int \*)malloc(n \* sizeof(int));      if (right == NULL) {          printf("Memory allocation failed!\n");          free(left); // Free the previously allocated memory          return -1;      }      // Append the values to left and right arrays depending on the direction      if (strcmp(direction, "left") == 0)          left[left\_count++] = 0;      else if (strcmp(direction, "right") == 0)          right[right\_count++] = track\_size - 1;      for (int i = 0; i < n; i++)      {          if (requests[i] < initial\_Position)              left[left\_count++] = requests[i];          else              right[right\_count++] = requests[i];      }      // Sort left and right arrays      qsort(left, left\_count, sizeof(int), compare);      qsort(right, right\_count, sizeof(int), compare);      // Run the loop twice      int run = 2;      while(run--)      {          if (strcmp(direction, "left") == 0)          {              for (int i = left\_count - 1; i >= 0; i--)              {                  int distance = abs(left[i] - current\_Position);                  total\_seek\_time += distance;                  current\_Position = left[i];              }              strcpy(direction, "right");          }          else if (strcmp(direction, "right") == 0)          {              for (int i = 0; i < right\_count; i++)              {                  int distance = abs(right[i] - current\_Position);                  total\_seek\_time += distance;                  current\_Position = right[i];              }              strcpy(direction, "left");          }      }      free(left); // Free dynamically allocated memory      free(right);      return total\_seek\_time;  }  int main()  {      int n;      printf("Enter the number of track requests you want: ");      scanf("%d", &n);      int \*requests = (int \*)malloc(n \* sizeof(int));      if (requests == NULL) {          printf("Memory allocation failed!\n");          return 1;      }      printf("Enter the track requests: ");      for (int i = 0; i < n; i++)      {          scanf("%d", &requests[i]);      }      int initial\_Position;      printf("Enter the Initial Position of the head: ");      scanf("%d", &initial\_Position);      char direction[10];      printf("Enter the direction (left/right): ");      scanf("%s", direction);      int track\_size;      printf("Enter the maximum number of tracks: ");      scanf("%d", &track\_size);      int seekTime = scan(requests, n, initial\_Position, direction, track\_size);      printf("The total seek time is: %d\n", seekTime);      free(requests);      return 0;  } |

**Output:**



1. **C-SCAN**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  int compare(const void \*a, const void \*b) {      return (\*(int\*)a - \*(int\*)b);  }  int cscan(int \*requests, int n, int initial\_Position, char \*direction, int track\_size)  {      int total\_seek\_time = 0;      int \*left = NULL;      int \*right = NULL;      int flag1 = 0, flag2 = 0; //to know if to add track\_size-1 or not      int currentPosition = initial\_Position;      // Allocate memory for left and right arrays      left = (int \*)malloc(n \* sizeof(int));      if (left == NULL) {          printf("Memory allocation failed!\n");          return -1;      }      right = (int \*)malloc(n \* sizeof(int));      if (right == NULL) {          printf("Memory allocation failed!\n");          free(left);          return -1;      }      left[0] = 0;      right[0] = track\_size - 1;      int left\_count = 1, right\_count = 1;      for (int i = 0; i < n; i++)      {          if (requests[i] < initial\_Position)          {              left[left\_count++] = requests[i];          }          else          {              right[right\_count++] = requests[i];          }      }      // Sort left and right arrays      qsort(left, left\_count, sizeof(int), compare);      qsort(right, right\_count, sizeof(int), compare);      int run = 2;      while (run--)      {          if (strcmp(direction, "right") == 0)          {              for (int i = 0; i < right\_count; i++)              {                  int distance = abs(right[i] - currentPosition);                  total\_seek\_time += distance;                  currentPosition = right[i];              }              strcpy(direction, "left");              flag1 = 1;              currentPosition = 0;          }          else if (strcmp(direction, "left") == 0)          {              for (int i = 0; i < left\_count; i++)              {                  int distance = abs(left[i] - currentPosition);                  total\_seek\_time += distance;                  currentPosition = left[i];              }              strcpy(direction, "right");              currentPosition = track\_size - 1;              flag2 = 1;          }      }      if (flag1 == 1 && flag2 == 1)      {          total\_seek\_time += track\_size - 1;      }      free(left); // Free dynamically allocated memory      free(right);      return total\_seek\_time;  }  int main()  {      int n;      printf("Enter the number of track requests you want: ");      scanf("%d", &n);      int \*requests = (int \*)malloc(n \* sizeof(int));      if (requests == NULL) {          printf("Memory allocation failed!\n");          return 1;      }      printf("Enter the track requests: ");      for (int i = 0; i < n; i++)      {          scanf("%d", &requests[i]);      }      int initial\_Position;      printf("Enter the Initial Position of the head: ");      scanf("%d", &initial\_Position);      char direction[10];      printf("Enter the direction (left/right): ");      scanf("%s", direction);      int track\_size;      printf("Enter the maximum number of tracks: ");      scanf("%d", &track\_size);      int seekTime = cscan(requests, n, initial\_Position, direction, track\_size);      printf("The total seek time is: %d\n", seekTime);      free(requests);      return 0;  } |

**Output:**

