**Name: Brijesh Rameshbhai Rohit**

**Admission number: U19CS009**

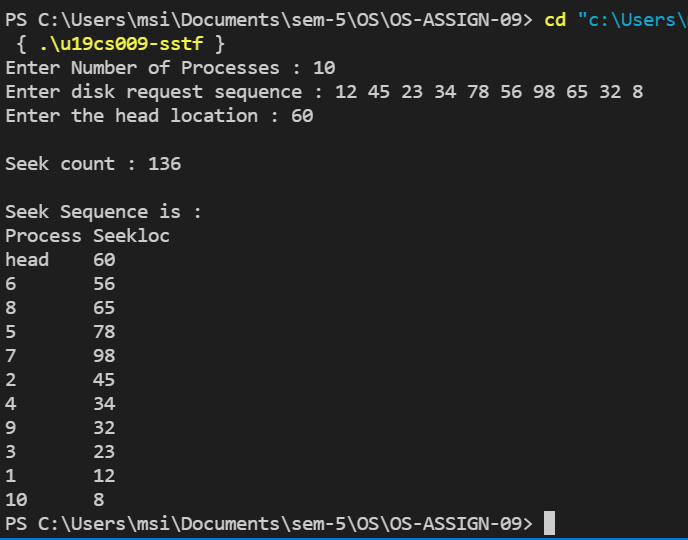
**OS-ASSIGNMENT-09**

**1.To implement Shortest Seek Time First (SSTF) Disk scheduling algorithm.**

**Code:**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;    class process  {  private:  int id;  int seekloc;    public:  process(int id, int a)  {  this->id = id;  this->seekloc = a;  }  int getSeekloc()  {  return this->seekloc;  }  void setSeekloc(int loc)  {  this->seekloc = loc;  }  int getId()  {  return this->id;  }  };    //FUNCTION TO GET THE PROCESS CLOSEST TO THE HEAD  int nextnearprocess(vector<process> &p, int l, int head)  {  int n = p.size();  int min = INT\_MAX;  int index = -1;  for (int i = l; i < p.size(); i++)  {  int dis = abs(p[i].getSeekloc() - head);  if (dis < min)  {  min = dis;  index = i;  }  }  return index;  }    //FUNCTION TO IMPLEMENT SSTF  void sstf(vector<process> &p, int n, int headpos)  {  int curr = 0, itr = 0;  int seekcount = 0, h = headpos; //loop till all processes are completed  while (curr < n)  {  int closestInd = nextnearprocess(p, curr, headpos); //getting Index of closest process  if (closestInd == -1)  {  break;  }  seekcount += abs(p[closestInd].getSeekloc() - headpos);  headpos = p[closestInd].getSeekloc();  swap(p[closestInd], p[itr]); //swapping inorder to remove the process that are completed and then increasing the itr  itr++;  curr++;  }  cout << endl << "Seek count : " << seekcount << endl;  cout << endl << "Seek Sequence is : " << endl;  cout << "Process\tSeekloc" << endl;  cout << "head \t" << h << endl;  for (int i = 0; i < n; i++)  {  cout << p[i].getId() << "\t" << p[i].getSeekloc() << endl;  }  }    int main()  {  int n;  cout << "Enter Number of Processes : ";  cin >> n;    vector<process> p;    cout << "Enter disk request sequence : ";    for (int i = 0; i < n; i++)  {  int a;  cin >> a;  p.push\_back(process(i + 1, a));  }    int headpos;  cout << "Enter the head location : ";  cin >> headpos;    sstf(p, n, headpos);    return 0;  } |

**Output:**

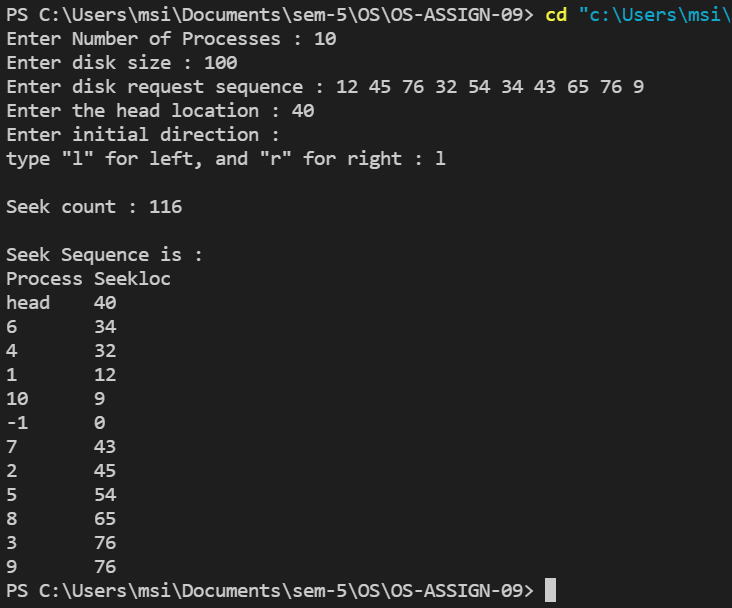


**2. To implement Scan Disk scheduling algorithm.**

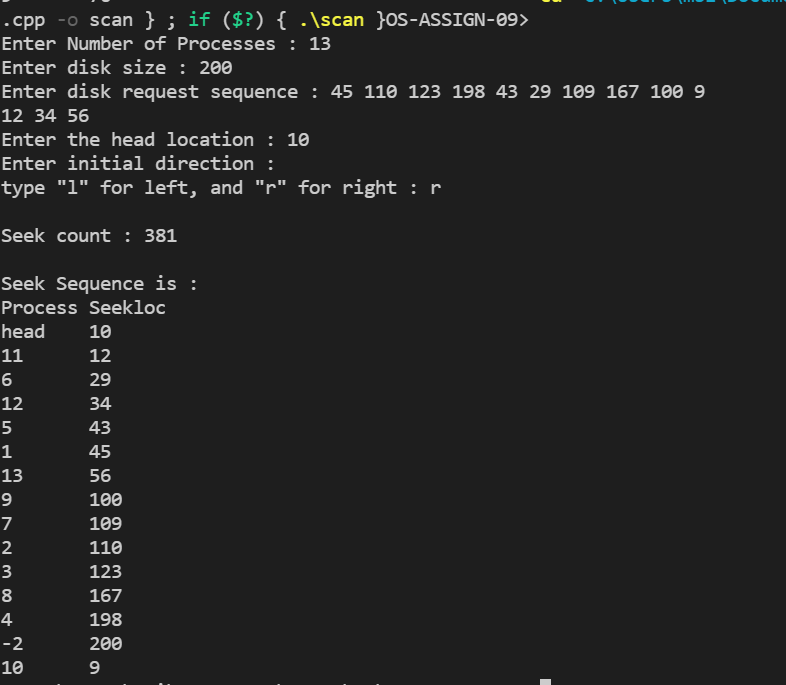
**Code:**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;    class process  {  private:  int id;  int seekloc;  int complete;    public:  process(int id, int a)  {  this->id = id;  this->seekloc = a;  this->complete = 0;  }  int getSeekloc() { return this->seekloc; }  void setSeekloc(int loc) { this->seekloc = loc; }  int getId() { return this->id; }  void setComplete() { this->complete = 1; }  int getComplete() { return this->complete; }  };    //FUNCTION TO CALCULATE DISTANCES IN A ONE DIRECTION FROM HEAD TO SELECT PROCESSES  vector<pair<int, int>> calcd(vector<process> &p, int flag, int head)  {  int n = p.size();  if (flag == 0) //LEFT DIRECTION  {  vector<pair<int, int>> p1;  for (int i = 0; i < n; i++)  {  if (p[i].getSeekloc() <= head && p[i].getComplete() != 1)  {  p1.push\_back({i, abs(p[i].getSeekloc() - head)});  }  }  return p1;  }  else //RIGHT DIRECTION  {  vector<pair<int, int>> p1;  for (int i = 0; i < n; i++)  {  if (p[i].getSeekloc() >= head && p[i].getComplete() != 1)  {  p1.push\_back({i, abs(p[i].getSeekloc() - head)});  }  }  return p1;  }  }    bool compare(pair<int, int> p1, pair<int, int> p2)  {  return p1.second < p2.second;  }    void scan(vector<process> &p, int n, int headloc, char dir)  {  int curr = 0, itr = 0;  int seekcount = 0, h = headloc;  vector<pair<int, int>> output;  while (curr < n + 1)  {  if (dir == 'l') //IF LEFT  {  vector<pair<int, int>> dis;  dir = 'r'; //ALL THE PROCESSES IN THE LEFT OF THE HEAD  dis = calcd(p, 0, h);  sort(dis.begin(), dis.end(), compare);  for (auto i : dis)  { //PROCESS INCOMPLETE THEN ALLOCATE  if (p[i.first].getComplete() != 1)  {  seekcount += abs(p[i.first].getSeekloc() - h);  h = p[i.first].getSeekloc();  output.push\_back({p[i.first].getId(), p[i.first].getSeekloc()});  p[i.first].setComplete();  }  curr++;  if (curr >= n + 1)  break; //ALL PROCESS COMPLETE THEN BREAK;  }  if (curr >= n + 1)  break; //ALL PROCESS COMPLETE THEN BREAK;  }  else if (dir == 'r') //IF RIGHT  {  vector<pair<int, int>> dis;  dir = 'l'; //ALL THE PROCESSES IN THE RIGHT OF THE HEAD  dis = calcd(p, 1, h);  sort(dis.begin(), dis.end(), compare);  for (auto i : dis)  { //PROCESS INCOMPLETE THEN ALLOCATE  if (p[i.first].getComplete() != 1)  {  seekcount += abs(p[i.first].getSeekloc() - h);  h = p[i.first].getSeekloc();  output.push\_back({p[i.first].getId(), p[i.first].getSeekloc()});  p[i.first].setComplete();  }  curr++;  if (curr >= n + 1)  break; //ALL PROCESS COMPLETE THEN BREAK;  }  if (curr >= n + 1)  break; //ALL PROCESS COMPLETE THEN BREAK;  }  }  cout << "\nSeek count : " << seekcount << endl;  cout << "\nSeek Sequence is : " << endl;  cout << "Process\tSeekloc" << endl;  cout << "head \t" << headloc << endl;  for (int i = 0; i < output.size(); i++)  {  cout << output[i].first << "\t" << output[i].second << endl;  }  }  int main()  {  int n, size;  cout << "Enter Number of Processes : ";  cin >> n;  cout << "Enter disk size : ";  cin >> size;    vector<process> p;  cout << "Enter disk request sequence : ";  for (int i = 0; i < n; i++)  {  int a;  cin >> a;  p.push\_back(process(i + 1, a));  }    int headloc;  cout << "Enter the head location : ";  cin >> headloc;    char dir;  cout << "Enter initial direction : " << endl;  cout << "type \"l\" for left, and \"r\" for right : ";  cin >> dir;    p.push\_back(process(-1, 0));    p.push\_back(process(-2, size));    scan(p, n, headloc, dir);    return 0;  } |

**Output:**



**-1 refers to left end of the Disk, meaning at which head location is 0**



**-1 refers to the right end of the disk meaning at which head location would be disk size [here 200].**