OS Lab File

NAME: Jaideep Singh

ROLL NO: 102103436

GROUP: 2C016

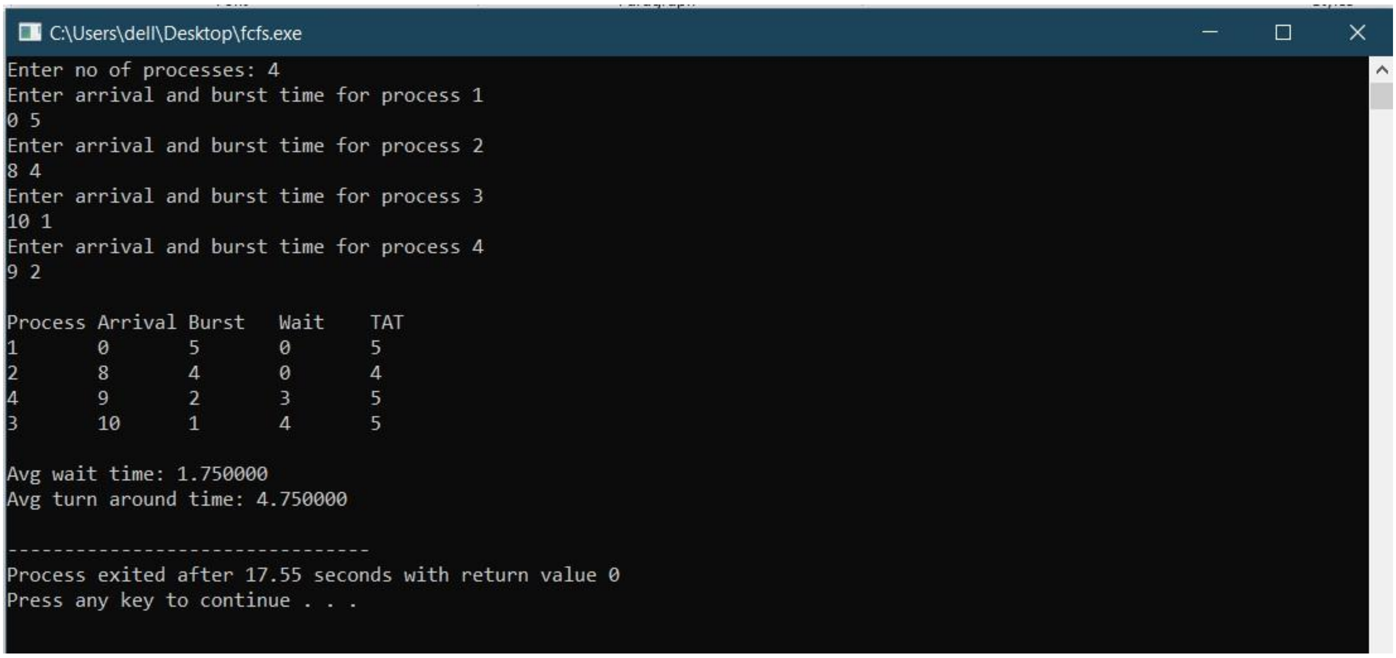
Lab Assignment 4

CPU Scheduling (FCFS):

```
#include <stdio.h>

void swap(int* a, int* b){
int temp = *a;     *a = *b;
   *b = temp;
}

void sort(int p[], int at[], int burstTIme[], int size){
   for(int i=0; i<size; i++){
for(int j=0; j<size-i-1; j++){
if(at[j]>at[j+1]){
swap(&at[j], &at[j+1]);
            swap(&burstTIme[j], &burstTIme[j+1]);
            swap(&p[j], &p[j+1]);
         }
      }
   }
}

double sum(int arr[], int size){
double sum=0;     for(int k=0;
k<size; k++)
sum+=arr[k];
   return sum;
}

int main(){
int N;
   printf("Enter no of processes: ");
scanf("%d", &N);

   //input
   int at[N], burstTIme[N], p[N], wait[N], turnAroundTime[N], ct[N], idle_time=0;
for(int i=0; i<N; i++){
      printf("Enter arrival and burst time for process %d\n", i+1);
scanf("%d\n%d", &at[i], &burstTIme[i]);
      p[i] = i+1;
   }
```

```
    sort(p, at, burstTIme, N);

    //fcfs
wait[0] = 0;
    ct[0] = at[0] + burstTIme[0];

    for(int i=0; i<N; i++){
idle_time=0;
        wait[i] = ct[i] - burstTIme[i] - at[i];
if(at[i+1] > ct[i]) idle_time += at[i+1] - ct[i];
ct[i+1] = ct[i] + burstTIme[i+1] + idle_time;
turnAroundTime[i] = wait[i] + burstTIme[i];
    }

    //display
    printf("\nProcess\tArrival\tBurst\tWait\tTURNAROUNDTIME\n");
for(int i=0; i<N; i++)
        printf("%d\t%d\t%d\t%d\t%d\n", p[i], at[i], burstTIme[i], wait[i], turnAroundTime[i]);

    printf("\nAvg wait time: %f", sum(wait, N)/N);
    printf("\nAvg turn around time: %f\n", sum(turnAroundTime, N)/N);
}
```

Output:



```
Enter no of processes: 4
Enter arrival and burst time for process 1
0 5
Enter arrival and burst time for process 2
8 4
Enter arrival and burst time for process 3
10 1
Enter arrival and burst time for process 4
9 2

Process Arrival Burst   Wait    TAT
1       0       5       0       5
2       8       4       0       4
4       9       2       3       5
3       10      1       4       5

Avg wait time: 1.750000
Avg turn around time: 4.750000

--------------------------------
Process exited after 17.55 seconds with return value 0
Press any key to continue . . .
```

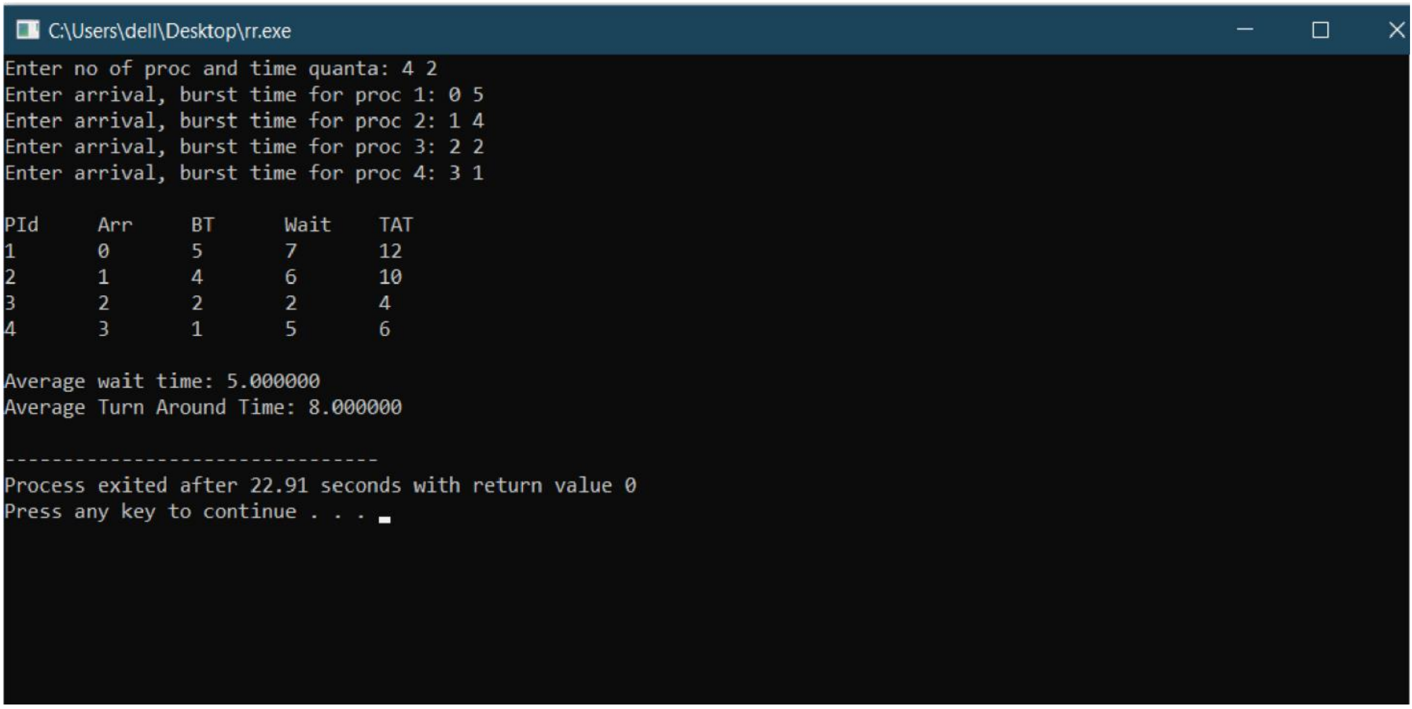
Round Robin:

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

void nextProcess(int queue[], int n, int selected_proc) {
int next;
    for (int i = 0; i < n; i++) {
if (queue[i] == 0) {
next = i;            break;
        }
    }
    queue[next] = selected_proc + 1;
}

void selectProcess(int queue[], int n) {     for (int i =
0; (i < n-1) && (queue[i+1] != 0) ; i++) {        int
temp = queue[i];        queue[i] = queue[i+1];
        queue[i+1] = temp;
    }
}

void newRequest(int curr_time, int at[], int n, int selected_proc, int queue[]) {
if (curr_time <= at[n-1]) {        bool newArrival = false;
       for (int j = selected_proc + 1; j < n; j++) {
if (at[j] <= curr_time) {                if
(selected_proc < j) {
selected_proc = j;
                    newArrival = true;
                }
           }
       }

       if (newArrival) {
          nextProcess(queue, n, selected_proc);
       }
    }
}

int main() {
    int n, tq, sumWait = 0, sumTURNAROUNDTIME = 0, selected_proc = 0;
    printf("Enter no of proc and time quanta: ");
scanf("%d %d", &n, &tq);

    int at[n], burstTIme[n], rt[n], waitTIme[n], turnAroundTime[n], queue[n], curr_time = 0,
idle_time = 0;     bool is_complete[n];

    for (int i = 0; i < n; i++) {
        printf("Enter arrival, burst time for proc %d: ", i+1);
        scanf("%d %d", &at[i], &burstTIme[i]);
        rt[i] = burstTIme[i];
queue[i] = 0;
        is_complete[i] = false;
    }

    while (curr_time != at[0]) {
curr_time++;
    }

    queue[0] = 1;

    while (true) {         bool
flag = true;        for (int i = 0;
i < n; i++) {            if (rt[i] !=
0) {                  flag = false;
break;
            }
        }
if (flag)
            break;

        for (int i = 0; (i < n) && (queue[i] != 0); i++) {
int ctr = 0;
            while ((ctr < tq) && (rt[queue[0]-1] > 0)) {
                rt[queue[0]-1]--;
curr_time++;
                ctr++;
                newRequest(curr_time, at, n, selected_proc, queue);
            }

            if ((rt[queue[0]-1] == 0) && (is_complete[queue[0]-1] == false)) {
turnAroundTime[queue[0]-1] = curr_time;                        is_complete[queue[0]-
1] = true;
            }

            bool idle = true;            if (queue[n-1] == 0)
{                 for (int i = 0; i < n && queue[i] != 0;
i++) {
                    if (is_complete[queue[i]-1] == false) {
                        idle = false;
                    }
                }            }
else {                idle
= false;
            }
if (idle) {
curr_time++;
                newRequest(curr_time, at, n, selected_proc, queue);
            }

            selectProcess(queue,n);
        }
    }
    for (int i = 0; i < n; i++) {
turnAroundTime[i] -= at[i];
        waitTIme[i] = turnAroundTime[i] - burstTIme[i];
    }

    printf("\nPId\tArr\tBURSTTIME\tWait\tTURNAROUNDTIME\n");
for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\n", i+1, at[i], burstTIme[i], waitTIme[i], turnAroundTime[i]);
    }

    for (int i = 0; i < n; i++) {
        sumWait += waitTIme[i];
        sumTURNAROUNDTIME += turnAroundTime[i];
    }

    printf("\nAverage wait time: %f", (float)sumWait/n);
    printf("\nAverage Turn Around Time: %f\n", (float)sumTURNAROUNDTIME/n);
}
```

Output:

SJF(without preemption)

```
#include <bits/stdc++.h> using

namespace std;
```



```
Enter no of proc and time quanta: 4 2
Enter arrival, burst time for proc 1: 0 5
Enter arrival, burst time for proc 2: 1 4
Enter arrival, burst time for proc 3: 2 2
Enter arrival, burst time for proc 4: 3 1

PId     Arr     BT      Wait    TAT
1       0       5       7       12
2       1       4       6       10
3       2       2       2       4
4       3       1       5       6

Average wait time: 5.000000
Average Turn Around Time: 8.000000

--------------------------------
Process exited after 22.91 seconds with return value 0
Press any key to continue . . .
```

```
#define SIZE 4
typedef struct proinfo {

        string pname; // process name

        int atime; // arrival time

        int btime; // burst time
```

```
} proinfo;

typedef struct cmpBtime {
    int operator()(const proinfo& a,
                    const proinfo& b)
    {
        return a.btime > b.btime;
    }
} cmpBtime; void
sjfNonpremetive(proinfo* arr)
{
    int index = 0;
    for (int i = 0; i < SIZE - 1; i++) {
        index = i;
        for (int j = i + 1; j < SIZE; j++) {
            if (arr[j].atime
        < arr[index].atime) {
                index = j;
            }
        }
        swap(arr[i], arr[index]);
    }
    int ctime = arr[0].atime;

    priority_queue<proinfo, vector<proinfo>,
                    cmpBtime>
    wait;

    int temp = arr[0].atime;



    wait.push(arr[0]);
    arr[0].atime = -1;
```

```
        cout << "Process id"
                << "\t";
        cout << "Arrival time"
                << "\t";
        cout << "Burst time"
                << "\t";

        cout << endl;

        while (!wait.empty()) {

                cout << "\t";
                cout << wait.top().pname << "\t\t";
                cout << wait.top().atime << "\t\t";
                cout << wait.top().btime << "\t\t";
                cout << endl;
                ctime += wait.top().btime;
                wait.pop();

                for (int i = 0; i < SIZE; i++) {
                        if (arr[i].atime <= ctime
                                && arr[i].atime != -1) {
                                wait.push(arr[i]);

                                arr[i].atime = -1;
                        }
                }
        }
}
int main()
{
        proinfo arr[SIZE];
        arr[0] = { "p1", 4, 3 };
        arr[1] = { "p2", 0, 8 };
        arr[2] = { "p3", 5, 4 };
        arr[3] = { "p4", 9, 2 };

        cout << "Process scheduling ";
        cout << "according to SJF is: \n"
                << endl;

        sjfNonpremetive(arr);
}
```

SJF(with preemption)

```
#include <bits/stdc++.h> using
namespace std;

struct Process {
        int pid; // Process ID
int bt; // Burst Time    int art;
// Arrival Time
};
void findWaitingTime(Process proc[], int n,int wt[])
{
        int rt[n];

        for (int i = 0; i < n; i++)
                rt[i] = proc[i].bt;

        int complete = 0, t = 0, minm = INT_MAX;
```

```
    int shortest = 0, finish_time;

    bool check = false;


    while (complete != n) {


        for (int j = 0; j < n; j++) {

            if ((proc[j].art <= t) &&

        (rt[j] < minm) && rt[j] > 0) {

                minm = rt[j];

        shortest = j;

check = true;

            }

        }


        if (check == false) {

            t++;

            continue;

        }




        rt[shortest]--;

minm = rt[shortest];            if

(minm == 0)

            minm = INT_MAX;

        if (rt[shortest] == 0) {



            complete++;

            check = false;
            finish_time = t + 1;
wt[shortest] = finish_time -
                          proc[shortest].bt -
                        proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }

        t++;
}
}

void findTurnAroundTime(Process proc[], int n,
                        int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                    total_tat = 0;
    findWaitingTime(proc, n, wt);
findTurnAroundTime(proc, n, wt, tat);      cout
<< " P\t\t"
        << "BT\t\t"
        << "WT\t\t"
        << "TAT\t\t\n";
    for (int i = 0; i < n; i++) {
        total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];          cout << " "
<< proc[i].pid << "\t\t"
            << proc[i].bt << "\t\t " << wt[i]
            << "\t\t " << tat[i] << endl;
    }

    cout << "\nAverage waiting time = "
<< (float)total_wt / (float)n;   cout <<
"\nAverage turn around time = "
        << (float)total_tat / (float)n;
}

int main()
{
    Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },
                    { 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };
    int n = sizeof(proc) / sizeof(proc[0]);

    findavgTime(proc, n);
    return 0;
}
```

PRIORITY ALGO

```
#include<bits/stdc++.h> using
namespace std;

struct Process
{
```

```
{

        int pid; // Process ID   int bt; // CPU
Burst time required     int priority; // Priority
of this process
};

bool comparison(Process a, Process b)
{
        return (a.priority > b.priority);
}
void findWaitingTime(Process proc[], int n,int wt[])
{
        wt[0] = 0;



        for (int i = 1; i < n ; i++ )
                wt[i] = proc[i-1].bt + wt[i-1] ;
}

void findTurnAroundTime( Process proc[], int n,int wt[], int tat[])
{
        for (int i = 0; i < n ; i++)
                tat[i] = proc[i].bt + wt[i];
}

void findavgTime(Process proc[], int n)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;

        findWaitingTime(proc, n, wt);
findTurnAroundTime(proc, n, wt, tat);          cout <<
```

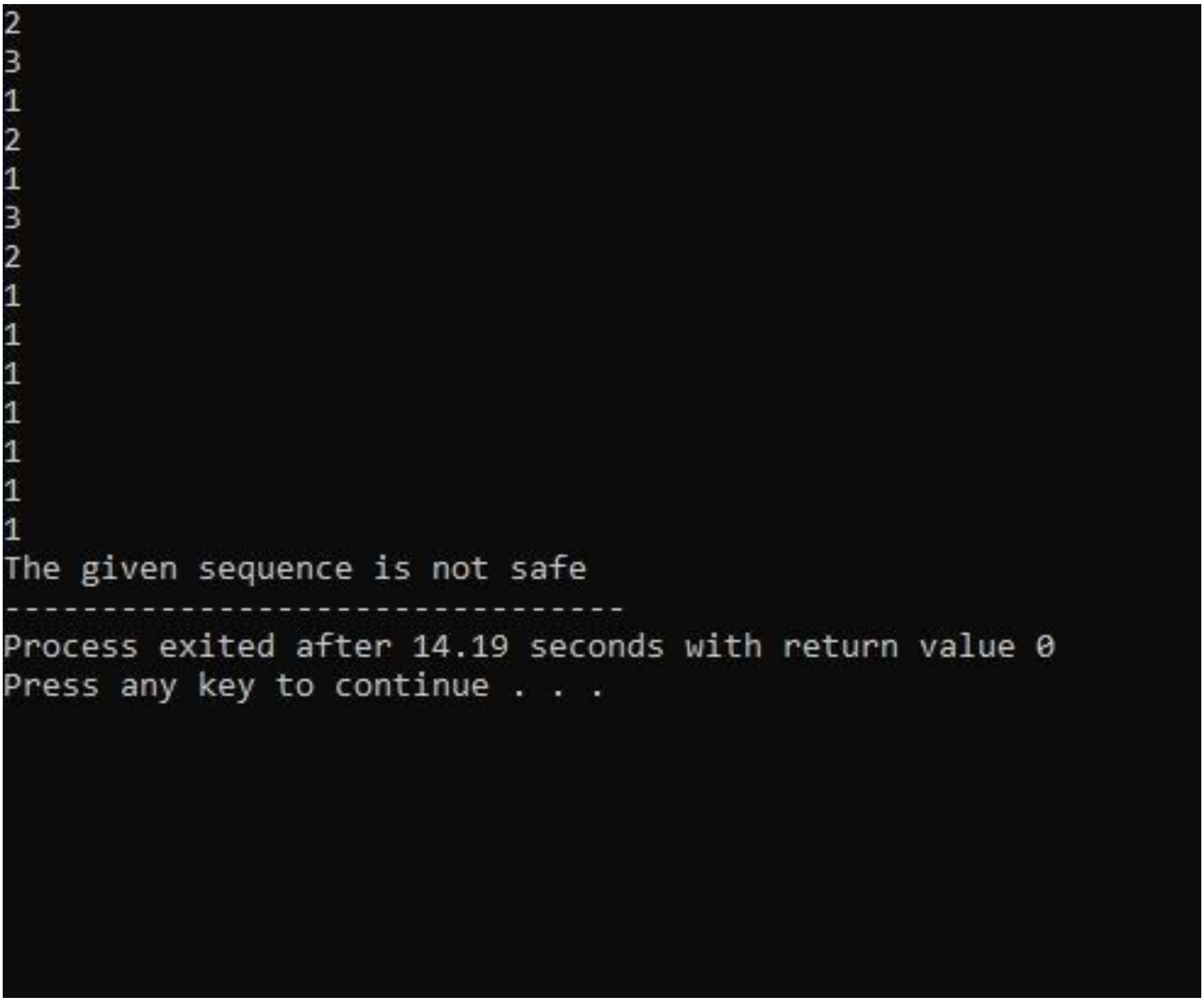
```
"\nProcesses "<< " Burst time "                    << " Waiting time "
<< " Turn around time\n";
            for (int i=0; i<n; i++)
            {
                        total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];               cout << " "
<< proc[i].pid << "\t\t"
                                    << proc[i].bt << "\t " << wt[i]
                                    << "\t\t " << tat[i] <<endl;
            }

            cout << "\nAverage waiting time = "
<< (float)total_wt / (float)n;   cout <<
"\nAverage turn around time = "
                        << (float)total_tat / (float)n;
}

void priorityScheduling(Process proc[], int n)
{
            sort(proc, proc + n, comparison);

            cout<< "Order in which processes gets executed \n";
            for (int i = 0 ; i < n; i++)
                        cout << proc[i].pid <<" " ;

            findavgTime(proc, n);
}
int main()
{
            Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
            int n = sizeof proc / sizeof proc[0];

            priorityScheduling(proc, n);

            return 0;

    }
```

Lab Assignment 5

BANKERS ALGORITHM

```
#include <iostream> using

namespace std;


int main()

{
int n, m, i, j, k;

cin>>n;//no. of processes cin>>m;//no.

of resources

int alloc[n][m] ; for(int

i=0;i<n;i++)

 {

            for(int j=0;j<m;j++)

            {

                        cin>>alloc[n][m];

             };

  };


int max[n][m] ; for(int

i=0;i<n;i++)

 {

            for(int j=0;j<m;j++)

            {

                        cout<<alloc[n][m]<<endl;

             };
  };

int avail[m] = { 3, 3, 2 };

 int f[n], ans[n], ind = 0;
  for (k = 0; k < n; k++) {
             f[k] = 0;
 }
 int need[n][m];
 for (i = 0; i < n; i++) { for (j =
              0; j < m; j++)
              need[i][j] = max[i][j] - alloc[i][j];
 }
 int y = 0; for (k = 0; k <5;
 k++) { for (i = 0; i < n; i++) {
              if (f[i] == 0) {

                           int flag = 0; for (j = 0; j
                           < m; j++) { if
                           (need[i][j] > avail[j]){
                           flag = 1;
                                        break;
                           }
                           }

                           if    (flag    ==    0)    {
                           ans[ind++]  =  i;  for  (y
                           = 0; y < m; y++)
                                        avail[y] += alloc[i][y];
                           f[i] = 1;
                           }
              }
              }
 }
```

```
int flag = 1;


// To check if sequence is safe or not for(int
i = 0;i<n;i++)
{
                    if(f[i]==0)
          {
                    flag = 0;
                    cout << "The given sequence is not safe";
                    break;
          }
}
if(flag==1)
{
          cout << "Following is the SAFE Sequence" << endl;
          for (i = 0; i < n - 1; i++)
                    cout << " P" << ans[i] << " ->";
          cout << " P" << ans[n - 1] <<endl;
}
return (0);
}
```



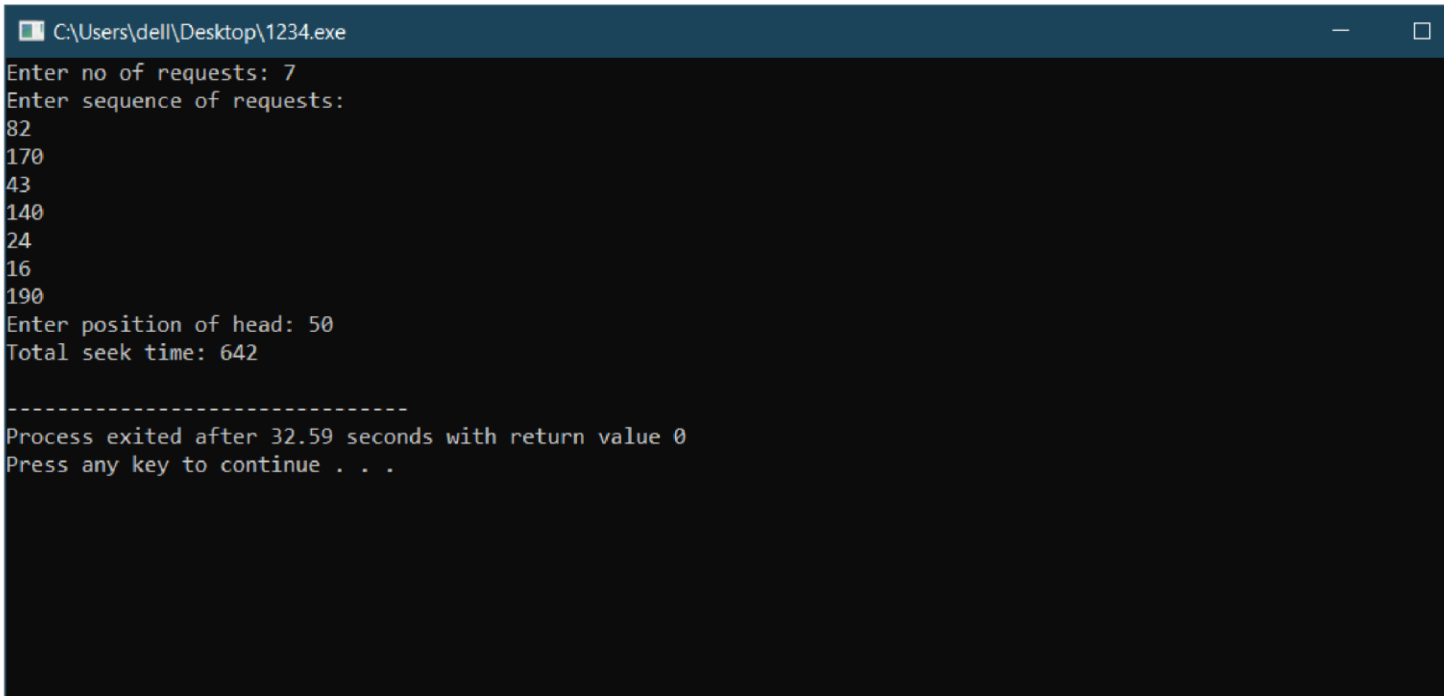
```
2
3
1
2
1
3
2
1
1
1
1
1
1
1
The given sequence is not safe
--------------------------------
Process exited after 14.19 seconds with return value 0
Press any key to continue . . .
```

Lab Assignment 8

Disk Scheduling (FCFS):

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

int seek_time(int arr[], int head, int len) {
int seek_time = abs(arr[0] - head);     for
(int i = 1; i < len; i++)
        seek_time += abs(arr[i] - arr[i-1]);

    return seek_time;
}

int main() {     int noOfReq,
head;     printf("Enter no of
requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq];     printf("Enter
sequence of requests:\n");     for (int i = 0;
i < noOfReq; i++)
        scanf("%d", &requests[i]);

    printf("Enter position of head: ");
    scanf("%d", &head);

    int ans = seek_time(requests, head, noOfReq);
    printf("Total seek time: %d\n", ans);
}
```

Output:



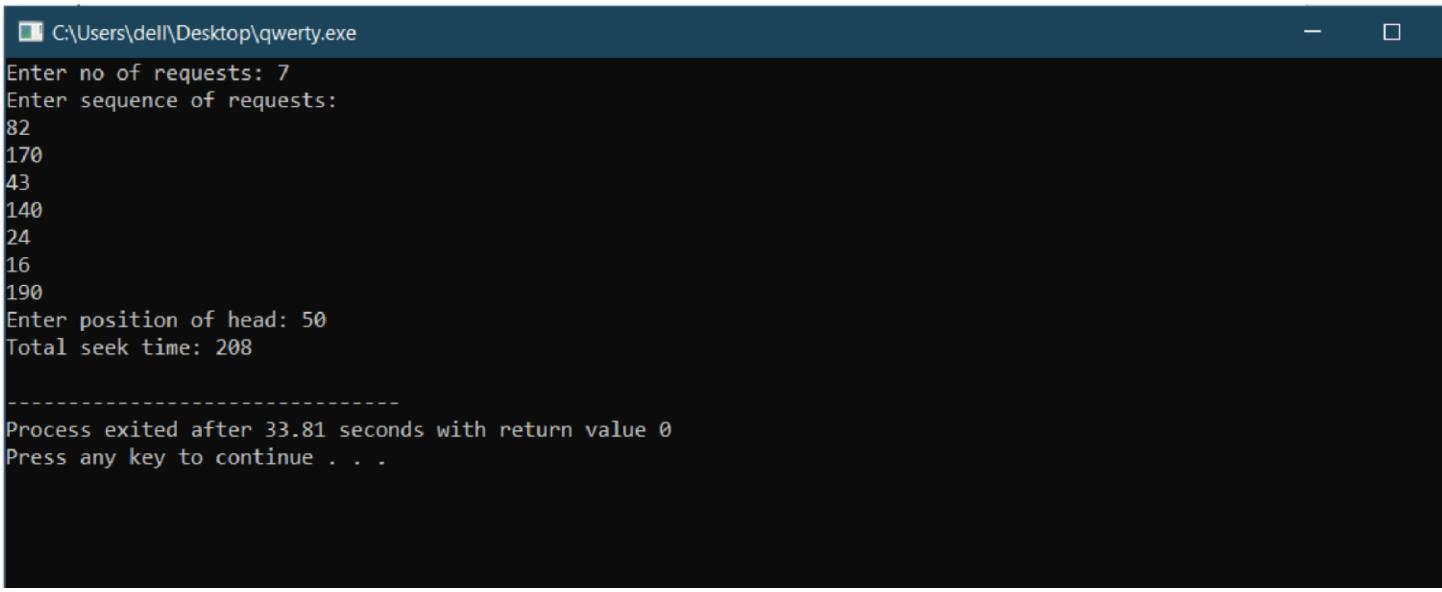
```
Enter no of requests: 7
Enter sequence of requests:
82
170
43
140
24
16
190
Enter position of head: 50
Total seek time: 642

--------------------------------
Process exited after 32.59 seconds with return value 0
Press any key to continue . . .
```

SSTF:

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

int min_index(int arr[], int len) {
int min = arr[0], index = 0;     for
(int i = 1; i < len; i++) {         if
(arr[i] <= min) {             min =
arr[i];
            index = i;
        }
    }

    return index;
}

int seek_time(int arr[], int head, int len) {
int seek_time = 0, req = len;     while
(req--) {         int closest[len];         for (int
i = 0; i < len; i++) {             closest[i] =
abs(arr[i] - head);
        }
        seek_time += abs(head - arr[min_index(closest, len)]);
head = arr[min_index(closest, len)];
```

```
        arr[min_index(closest, len)] = INT_MAX;
    }

    return seek_time;
}


int main() {     int noOfReq,
head;     printf("Enter no of
requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq];     printf("Enter
sequence of requests:\n");     for (int i = 0;
i < noOfReq; i++)
        scanf("%d", &requests[i]);

    printf("Enter position of head: ");
    scanf("%d", &head);

    int ans = seek_time(requests, head, noOfReq);
    printf("Total seek time: %d\n", ans);
}
```

Output:



```
Enter no of requests: 7
Enter sequence of requests:
82
170
43
140
24
16
190
Enter position of head: 50
Total seek time: 208

--------------------------------
Process exited after 33.81 seconds with return value 0
Press any key to continue . . .
```

SCAN:

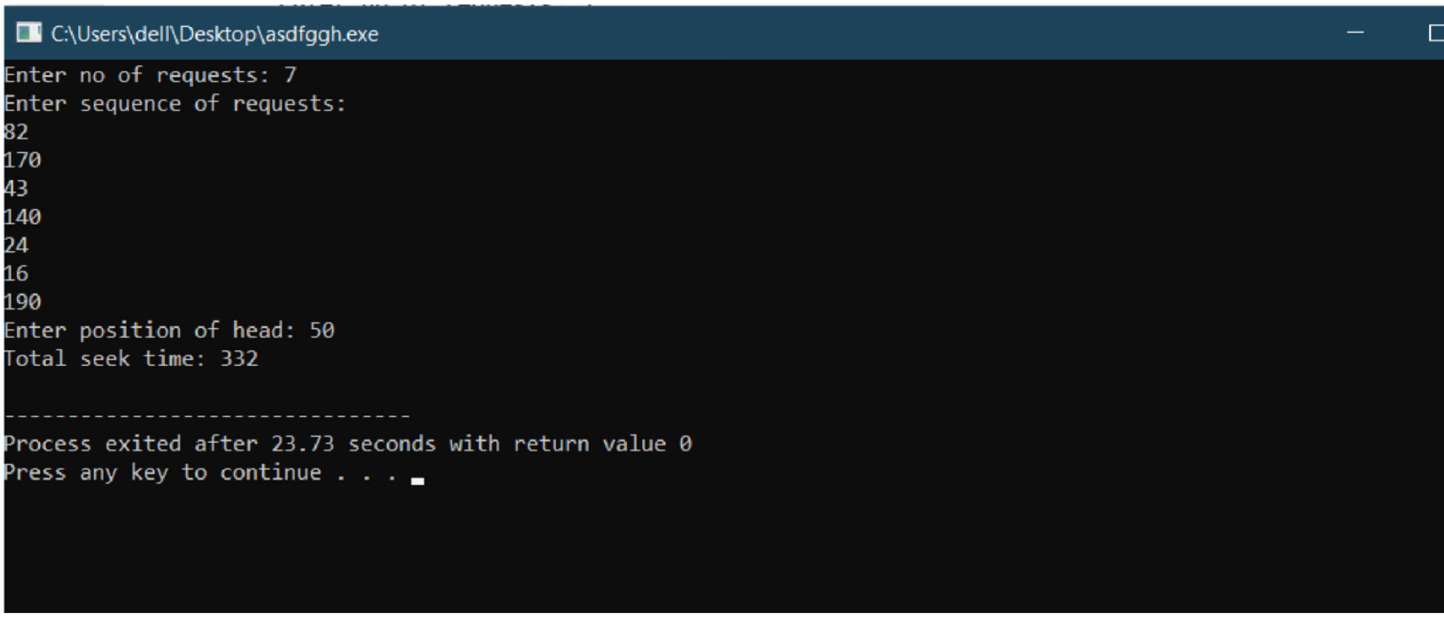
```
#include <stdio.h>
#include <stdlib.h>

int min(int arr[], int len) {
int min = arr[0];     for (int i
= 1; i < len; i++) {         if
(arr[i] <= min) {
            min = arr[i];
        }
    }

    return min;
}

int main() {     int noOfReq,
head;     printf("Enter no of
requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq], max_req = 0;
printf("Enter sequence of requests:\n");     for
(int i = 0; i < noOfReq; i++) {
scanf("%d", &requests[i]);
        max_req = (requests[i] >= max_req) ? requests[i] : max_req;
    }

    printf("Enter position of head: ");
scanf("%d", &head);

    int range_min = 0, range_max = max_req + (10-1);

    int ans = abs(range_max - head) + (range_max - min(requests, noOfReq));
printf("Total seek time: %d\n", ans);
}
```

Output:

CSCAN:

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

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



```
Enter no of requests: 7
Enter sequence of requests:
82
170
43
140
24
16
190
Enter position of head: 50
Total seek time: 332

--------------------------------
Process exited after 23.73 seconds with return value 0
Press any key to continue . . .
```

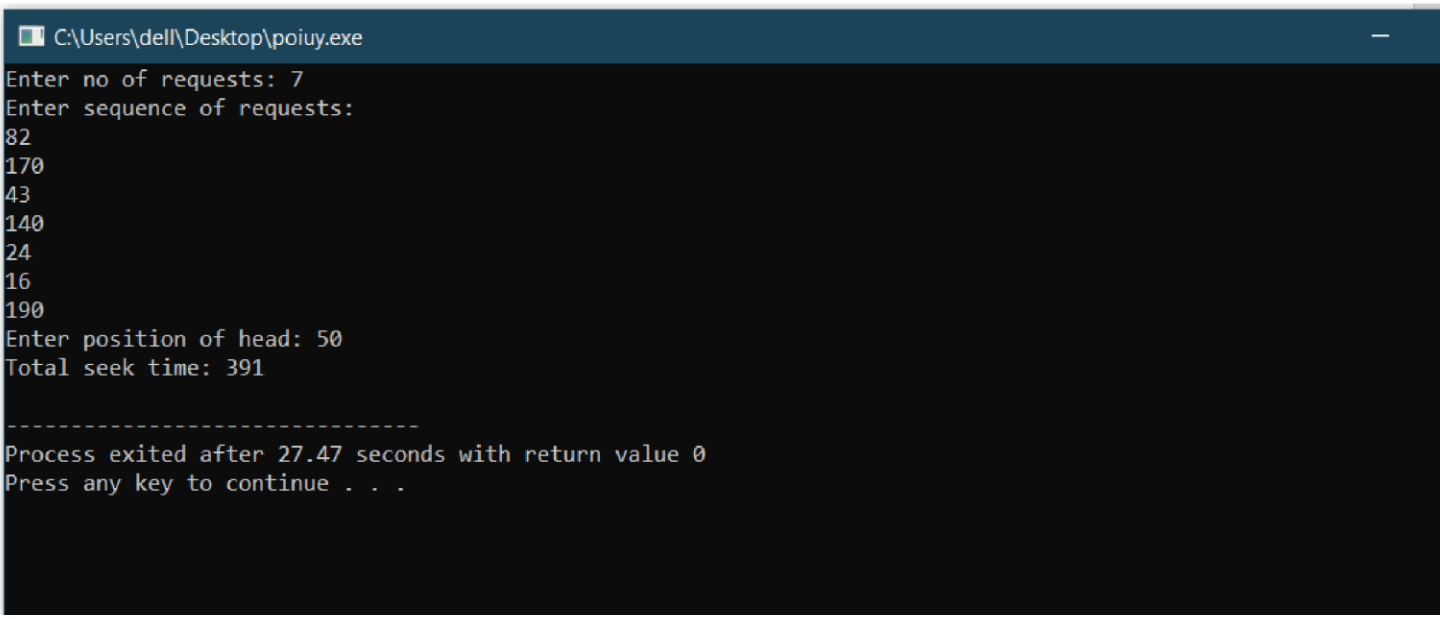
```
void sort(int arr[], int len) {     for
(int i = 0; i < len; i++) {         for
(int j = 0; j < len-i-1; j++) {
if (arr[j] > arr[j+1]) {                int
temp = arr[j+1];                arr[j+1]
= arr[j];
                arr[j] = temp;
            }
        }
    }
}

int min(int arr[], int len, int head) {
sort(arr, len);     int val = arr[0];
for (int i = 0; i < len; i++) {         if
(arr[i] <= head) {            val = arr[i];
} else {            break;
        }
    }

    return val;
}

int main() {     int noOfReq,
head;     printf("Enter no of
requests: ");
```

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

   int requests[noOfReq], max_req = 0;
printf("Enter sequence of requests:\n");     for
(int i = 0; i < noOfReq; i++) {
scanf("%d", &requests[i]);
       max_req = (requests[i] >= max_req) ? requests[i] : max_req;
   }

   printf("Enter position of head: ");
scanf("%d", &head);

   int range_min = 0, range_max = max_req + (10-1);

   int ans = abs(range_max - head) + (range_max - range_min) + (min(requests, noOfReq, head) -
range_min);
   printf("Total seek time: %d\n", ans);
}
```

Output:

LOOK:

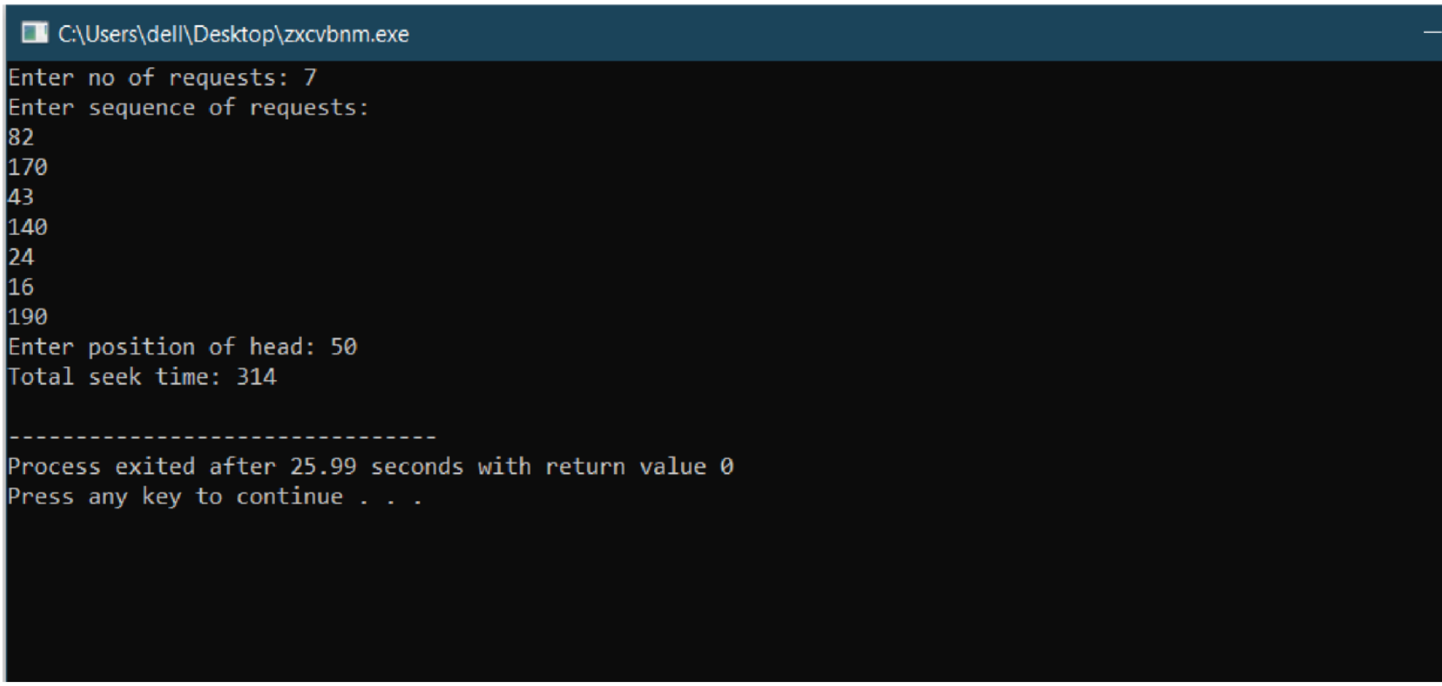
```
#include <stdio.h>
#include <stdlib.h>
```



```
Enter no of requests: 7
Enter sequence of requests:
82
170
43
140
24
16
190
Enter position of head: 50
Total seek time: 391

--------------------------------
Process exited after 27.47 seconds with return value 0
Press any key to continue . . .
```

```
int min(int arr[], int len) {
int min = arr[0], index = 0;
for (int i = 1; i < len; i++) {
if (arr[i] <= min) {
            min = arr[i];
        }
    }

    return min;
}

int main() {     int noOfReq,
head;     printf("Enter no of
requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq], max_req = 0;
printf("Enter sequence of requests:\n");     for
(int i = 0; i < noOfReq; i++) {
scanf("%d", &requests[i]);
        max_req = (requests[i] >= max_req) ? requests[i] : max_req;
    }

    printf("Enter position of head: ");
scanf("%d", &head);

    int ans = abs(max_req - head) + (max_req - min(requests, noOfReq));
    printf("Total seek time: %d\n", ans);
}
```

Output:



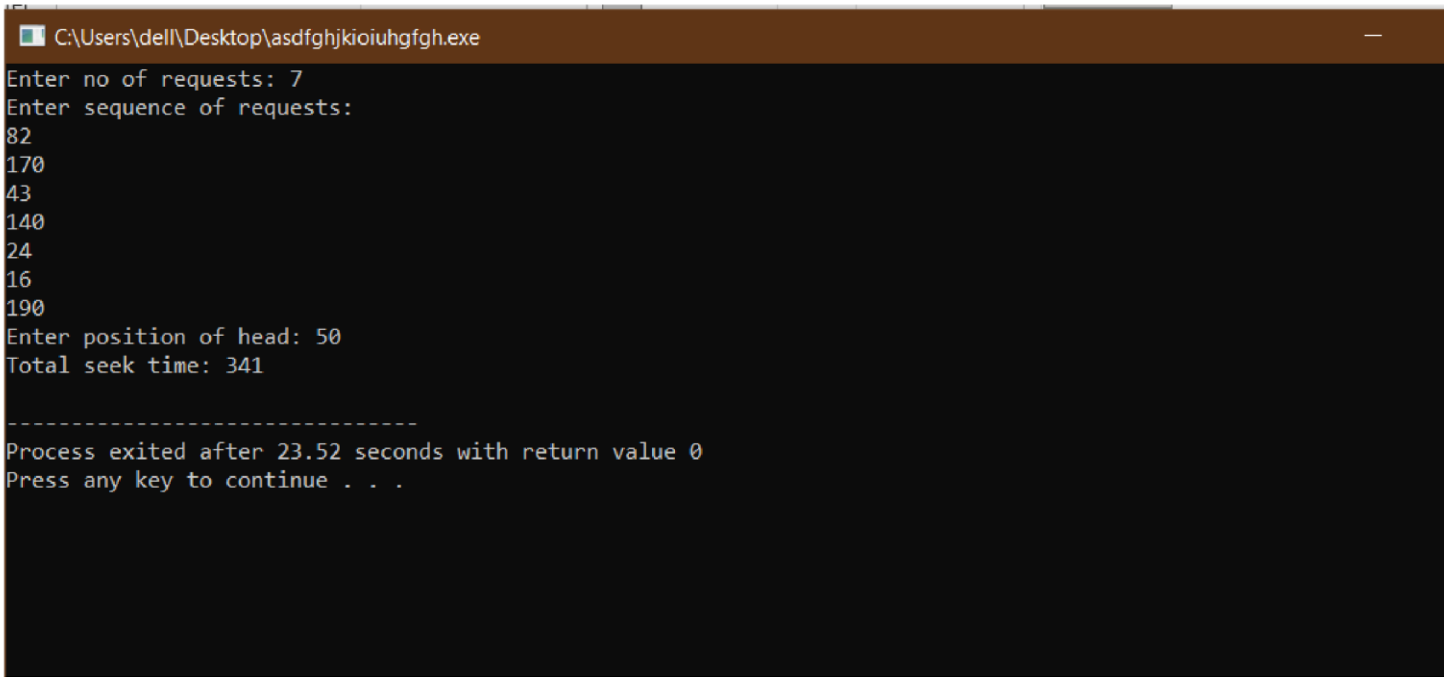
```
Enter no of requests: 7
Enter sequence of requests:
82
170
43
140
24
16
190
Enter position of head: 50
Total seek time: 314

--------------------------------
Process exited after 25.99 seconds with return value 0
Press any key to continue . . .
```

CLOOK:

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

void sort(int arr[], int len) {     for
(int i = 0; i < len; i++) {         for
(int j = 0; j < len-i-1; j++) {
if (arr[j] > arr[j+1]) {                 int
temp = arr[j+1];                 arr[j+1]
= arr[j];
                arr[j] = temp;
            }
        }
    }
}

int min(int arr[], int len, int head) {
    sort(arr, len);     int val =
arr[0];     for (int i = 0; i <
len; i++) {         if (arr[i] <=
head) {             val = arr[i];
} else {             break;
        }
    }

    return val;
}

int main() {     int noOfReq,
head;     printf("Enter no of
requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq], max_req = INT_MIN, min_req = INT_MAX;
printf("Enter sequence of requests:\n");
    for (int i = 0; i < noOfReq; i++) {
scanf("%d", &requests[i]);
        max_req = (requests[i] >= max_req) ? requests[i] : max_req;
min_req = (requests[i] <= min_req) ? requests[i] : min_req;
    }
```

```
   printf("Enter position of head: ");
scanf("%d", &head);

   int ans = abs(max_req - head) + (max_req - min_req) + (min(requests, noOfReq, head) - min_req);
printf("Total seek time: %d\n", ans);
}
```

Output:



```
Enter no of requests: 7
Enter sequence of requests:
82
170
43
140
24
16
190
Enter position of head: 50
Total seek time: 341

--------------------------------
Process exited after 23.52 seconds with return value 0
Press any key to continue . . .
```

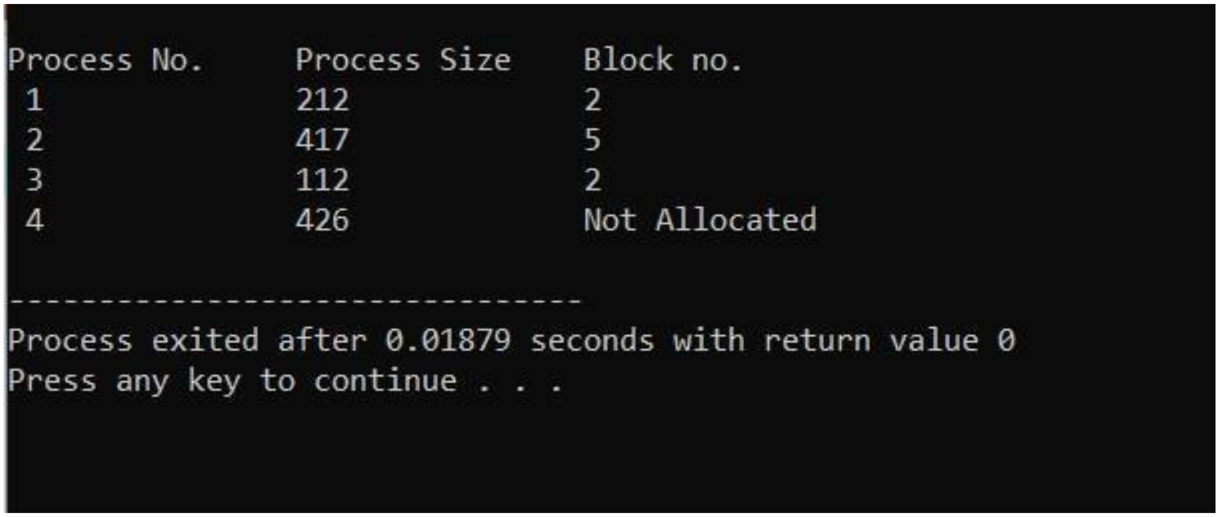
Lab Assignment 9

Contiguous Memory Allocation

First Fit Algorithm

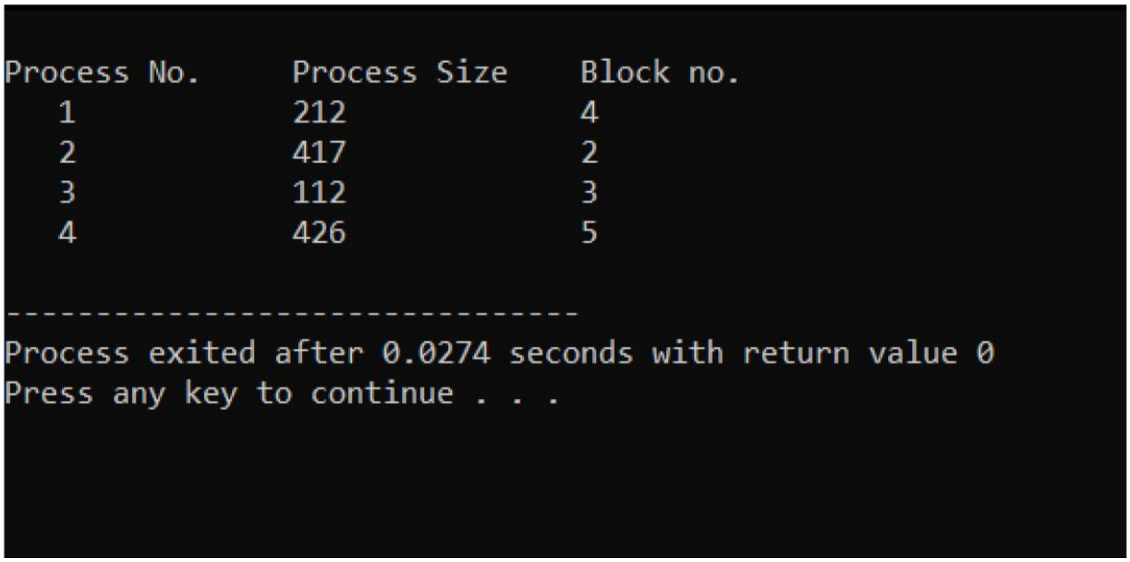
```
#include<iostream> using namespace std;

void firstFit(int blockSize[], int m,

                              int processSize[], int n)

{ int allocation[n]; for(int i=0;i<n;i++)

          { allocation[i]=-1;

          }

          for (int i = 0; i < n; i++)

          { for (int j = 0; j < m; j++)

                    { if (blockSize[j] >= processSize[i])

                              { allocation[i] = j; blockSize[j] -= processSize[i];


                                        break;

                              }

                    }

          }

          cout << "\nProcess No.\tProcess Size\tBlock no.\n"; for (int i = 0; i < n; i++)

          { cout << " " << i+1 << "\t\t"

                              <<  processSize[i] << "\t\t"; if (allocation[i] != -1)
                    cout << allocation[i] + 1; else cout << "Not Allocated";

                    cout << endl;

          }

}

int main()

{

          int blockSize[] = {100, 500, 200, 300, 600}; int processSize[] = {212, 417, 112 , 426};
          int m = sizeof(blockSize) / sizeof(blockSize[0]); int n = sizeof(processSize) /
          sizeof(processSize[0]); firstFit(blockSize, m, processSize, n);
          return 0 ;
}
```

Best Fit Algorithm



```
Process No.     Process Size    Block no.
 1              212             2
 2              417             5
 3              112             2
 4              426             Not Allocated

--------------------------------
Process exited after 0.01879 seconds with return value 0
Press any key to continue . . .
```

```
#include<iostream> using namespace std;

void bestFit(int blockSize[], int m, int processSize[], int n)
{ int allocation[n]; for(int i=0;i<n;i++)
          { allocation[i]=-1;
          }

    for (int i=0; i<n; i++)
    { int bestIdx = -1; for (int j=0; j<m; j++)
        { if (blockSize[j] >= processSize[i])
            { if (bestIdx == -1)
                    bestIdx = j;

                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            } }
        if (bestIdx != -1)
        { allocation[i] = bestIdx; blockSize[bestIdx] -= processSize[i];
        }

    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n"; for (int i = 0; i < n; i++)

    { cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t"; if (allocation[i] != -1)
            cout << allocation[i] + 1;

        else cout << "Not

        Allocated"; cout << endl;

    }

} int main()

{

    int blockSize[] = {100, 500, 200, 300, 600}; int processSize[] = {212, 417, 112 ,

    426}; int m = sizeof(blockSize)/sizeof(blockSize[0]); int n =

    sizeof(processSize)/sizeof(processSize[0]); bestFit(blockSize, m, processSize, n);
```



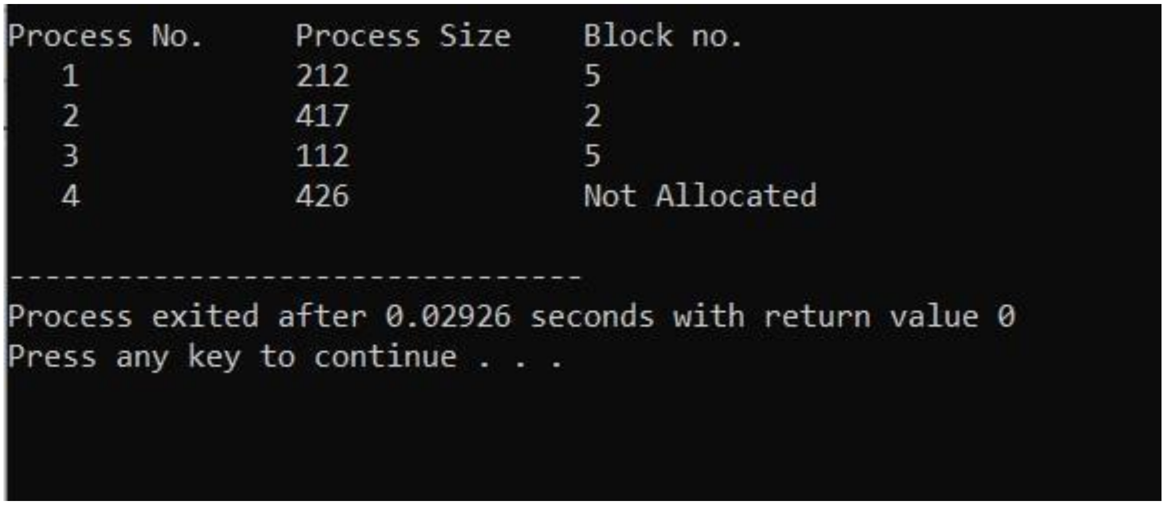
```
Process No.     Process Size    Block no.
   1            212             4
   2            417             2
   3            112             3
   4            426             5

--------------------------------
Process exited after 0.0274 seconds with return value 0
Press any key to continue . . .
```

```
    return 0 ;

}
```

Worst Fit Algorithm

```
#include<bits/stdc++.h> using namespace std;


void worstFit(int blockSize[], int m, int processSize[], int n)

{ int allocation[n]; for(int i=0;i<n;i++)
```

```
    { allocation[i]=-1;
          }

    for (int i=0; i<n; i++)
    { int wstIdx = -1; for (int j=0; j<m; j++)
        { if (blockSize[j] >= processSize[i])
            { if (wstIdx == -1) wstIdx
                    = j;
                                        else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            } }

        if (wstIdx != -1)
        { allocation[i] = wstIdx; blockSize[wstIdx] -= processSize[i];
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n"; for (int i = 0; i < n; i++)
    { cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1) cout << allocation[i] + 1; else
        cout << "Not Allocated"; cout << endl;

    }
}

int main()
{
    int blockSize[] = {100, 500, 200, 300, 600}; int processSize[] = {212, 417, 112 ,
    426}; int m = sizeof(blockSize)/sizeof(blockSize[0]); int n =
    sizeof(processSize)/sizeof(processSize[0]); worstFit(blockSize, m, processSize,
    n);
    return 0 ;
}
```



```
Process No.     Process Size    Block no.
   1            212             5
   2            417             2
   3            112             5
   4            426             Not Allocated

--------------------------------
Process exited after 0.02926 seconds with return value 0
Press any key to continue . . .
```