**Program :**

#include <stdio.h>

typedef struct process{

    int process\_id;

    int priority;

    int burst\_time;

} process;

void swap(process \*a, process \*b){

    process t = \*a;

    \*a = \*b;

    \*b = t;

}

void sort\_by\_priority(process \*arr, int n){

    int swp = 0;

    do{

        for(int i = 1; i < n; i++){

            if(arr[i].priority < arr[i-1].priority){

                swap(&arr[i], &arr[i-1]);

                swp = 1;

            }

        }

        if(!swp) break;

        else swp = 0;

    } while(1);

}

int main(){

    int n;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    process arr[n];

    for(int i = 0; i < n; i++){

        arr[i].process\_id = i+1;

        printf("\nProcess\_id %d\n", i+1);

        printf("Enter priority: ");

        scanf("%d", &arr[i].priority);

        printf("Enter burst time: ");

        scanf("%d", &arr[i].burst\_time);

    }

    sort\_by\_priority(arr, n);

    int wait = 0, turn = 0, tot\_wait = 0;

    printf("The process are :\n");

    printf("ID\tFrom\tTo\tWT\tTAT\n");

    for(int i = 0; i < n; i++){

        printf("%d\t%d\t%d\t%d\t%d\n",arr[i].process\_id,wait,wait+arr[i].burst\_time,wait, wait+arr[i].burst\_time);

        if(i != n-1) {

            tot\_wait += wait+arr[i].burst\_time;

        }

        turn += wait+arr[i].burst\_time;

        wait += arr[i].burst\_time;

    }

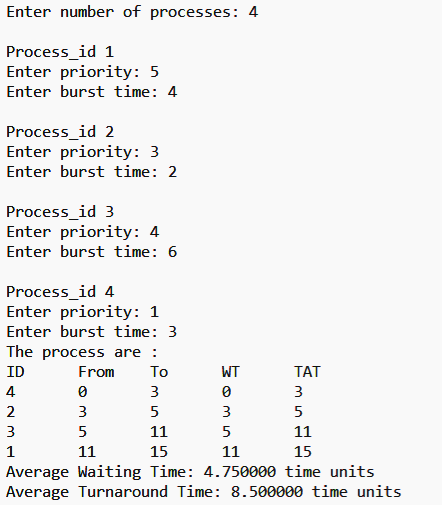
    printf("Average Waiting Time: %f time units\n", (float)tot\_wait/n);

    printf("Average Turnaround Time: %f time units\n", (float)turn/n);

    return 0;

}

**Output :**

****

**Program :**

#include<bits/stdc++.h>

using namespace std;

int main(void) {

    cout << "Enter page size:\n";

    int p;

    cin >> p;

    vector<int> pages;

    cout << "Enter string length:\n";

    int n;

    cin >> n;

    vector<int> a(n);

    cout << "Enter page string:\n";

    for(int i=0; i<n; i++) {

        cin >> a[i];

    }

    cout << endl;

    for(int i=0; i<n; i++) {

        if(pages.size() != p) {

            pages.push\_back(a[i]);

        }

        else {

            int index = 0;

            for(int pos=0; pos<p; pos++) {

                if(pages[pos] == a[i]) {

                    index = pos;

                }

            }

            pages.erase(pages.begin()+index);

            pages.push\_back(a[i]);

        }

        for(int itr=0; itr<pages.size(); itr++) {

            cout << pages[itr] << ' ';

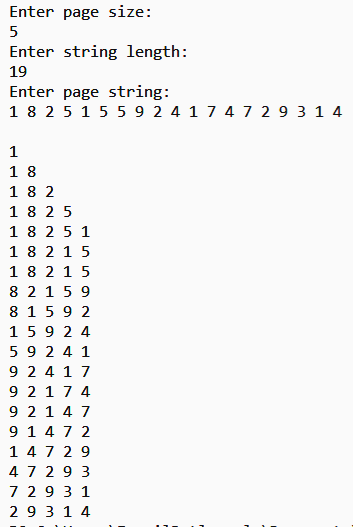
        }

        cout << endl;

    }

}

**Output :**

****

**Program :**

import java.util.Scanner;

public class BestFit{

    static void bestFit(int blockSize[], int m, int processSize[], int n){

        int allocation[] = new int[n]; // Stores block id of the block allocated to a process

        for (int i = 0; i < allocation.length; i++) // Initially no block is assigned to any process

        allocation[i] = -1;

        System.out.println("\nProcess no.\tProcess Size\tBlock no.\tBlock size remaining");

        for (int i=0; i<n; i++){

            int bestIdx = -1;  // Find the best fit block for current process

            boolean allocated = false;

            for (int j=0; j<m; j++){

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

                    if (allocated == false){ //to assign block first time

                        bestIdx = j;

                        allocated=true;

                    }

                    else if (blockSize[bestIdx] > blockSize[j])

                        bestIdx = j;

                }

            }

            if (allocated == true){      //if allocated update allocation list and reduce corresp block size

                allocation[i] = bestIdx; // allocate block j to p[i] process

                blockSize[bestIdx] -= processSize[i]; // Reduce available memory in this block.

            }

            System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t" );//printing output

            if (allocation[i] != -1)

                System.out.print(allocation[i] + 1 + "\t\t" + blockSize[allocation[i]]);

            else

                System.out.print("Not Allocated");

            System.out.println();

        }

    }

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of processes : ");

        int n = sc.nextInt();

        int processSize[] = new int[n];

        System.out.println("Enter the processes : ");

        for(int i=0;i<n;i++){

            System.out.print("Process No:"+ (i+1) + " =");

            processSize[i]=sc.nextInt();

        }

        System.out.print("\nEnter the number of memory blocks : ");

        int m = sc.nextInt();

        int blockSize[] = new int[m];

        System.out.println("Enter the memory blocks : ");

        for(int i=0;i<m;i++){

            System.out.print("Block No:"+ (i+1) + " =");

            blockSize[i]=sc.nextInt();

        }

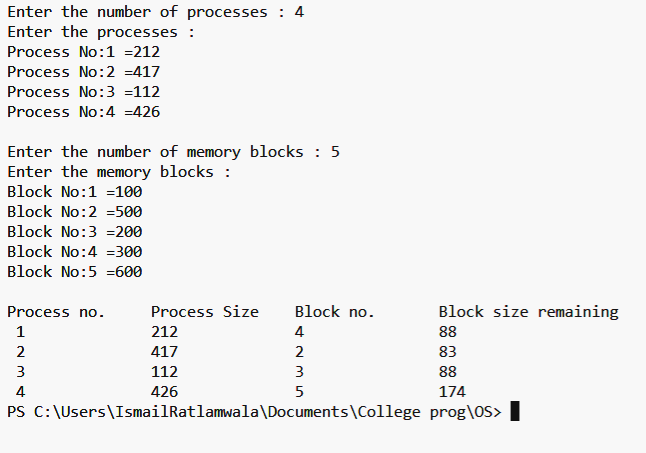
        bestFit(blockSize, m, processSize, n);

        sc.close();

    }

}

**Output :**

****

**Program :**

#include <bits/stdc++.h>

using namespace std;

int bin2dec(char \*s, int n){

   int res = 0;

   for(int i = n-1; i >= 0; i--)

       res += (s[i]-'0')\*(1<<i);

   return res;

}

int checkTable(int page, int pt[][3]){

   return pt[page][2];

}

int main(){

   int processSize, pageSize, physicalMem;

   cout << "Enter process size in KB:\n";

   cin >> processSize;

   cout << "Enter page size in bytes:\n";

   cin >> pageSize;

   cout << "Enter size of physical memory in MB:\n";

   cin >> physicalMem;

   int frames = (physicalMem\*(1<<20))/pageSize;

   printf("\nNo. of frames in memory: %d (i.e. 2^%.0f)\n", frames, log2(frames));

   int n = (processSize\*(1<<10))/pageSize;

   printf("No. of entries in page table: %d (i.e. 2^%.0f)\n", n, log2(n));

   float phy\_add\_bits = log2(physicalMem\*(1<<20));

   printf("No. of bits in physical address: %0.f\n", phy\_add\_bits);

   float log\_add\_bits = log2(processSize\*(1<<10));

   printf("No. of bits in logical address: %0.f\n", log\_add\_bits);

   printf("\nPage Segment: %0.f bits\tOffset: %0.f bits\n", log2(n), log\_add\_bits-log2(n));

   int pt[n][3];

   cout << "\nInput page table: (Page No | Frame No | Valid Bit)\n";

   for(int i = 0; i < 4; i++) {

       scanf("%d %d %d", &pt[i][0], &pt[i][1], &pt[i][2]);

   }

   int repeat=1;

   while(repeat){

      char logAdd[(int)log\_add\_bits];

      printf("Input logical address: ");

      scanf("%s", logAdd);

      int page = bin2dec(logAdd, (int)log2(n));

      printf("%s", (checkTable(page, pt) ? "Page Hit\n":"Page Fault\n"));

      cout<<"\nTo continue enter 1 else 0 : ";

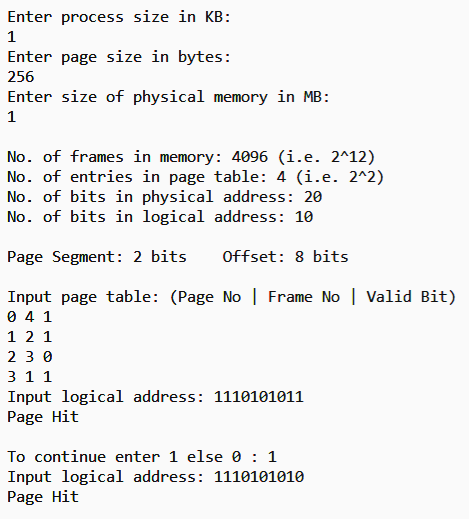
      cin>>repeat;

   }

   return 0;

}

**Output :**

****

**Program :**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    vector<pair<int,bool>> request;

    cout<<"Enter number of request : ";

    int n,inp;

    cin>>n;

    cout<<"Enter the requests :"<<endl;

    for(int i=0;i<n;i++) {

        cin>>inp;

        request.push\_back({inp,false});

    }

    cout<<"Enter the position of head : ";

    int head;

    cin>>head;

    int totalMov=0;

    cout<<"Movement of head : \n"<<head;

    for(int j=0;j<n;j++){

        int minDiff=INT\_MAX, indx;

        for(int i=0;i<n;i++){

            if(!request[i].second && (abs(head-request[i].first)<minDiff)){

                minDiff=abs(head-request[i].first);

                indx=i;

            }

        }

        cout<<" => "<<request[indx].first;

        head=request[indx].first;

        request[indx].second=true;

        totalMov += minDiff ;

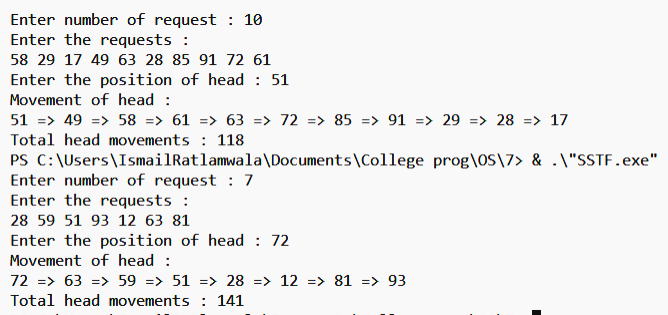
    }

    cout<<"\nTotal head movements : "<<totalMov<<endl;

    return 0;

}

**Output :**



**Program :**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    vector<int> request;

    cout<<"Enter the size of disk : ";

    int size;

    cin>>size;

    size--;

    cout<<"Enter number of request : ";

    int n;

    cin>>n;

    cout<<"Enter the requests :"<<endl;

    for(int i=0;i<n;i++) {

        int inp;

        cin>>inp;

        request.push\_back(inp);

    }

    cout<<"Enter the position of head : ";

    int head;

    cin>>head;

    bool fromLeft=false;

    sort(request.begin(),request.end());

    int cutoffInd;

    for(cutoffInd=0; cutoffInd<n; cutoffInd++)

        if(request[cutoffInd]>head) break;

    cout<<"Movement of head : \n"<<head;

    if(fromLeft){

        for(int i=cutoffInd-1;i>=0;i--) cout<<" => "<<request[i];

        for(int i=cutoffInd; i<n; i++) cout<<" => "<<request[i];

        cout<<"\nTotal head movements : "<<request[n-1]+head;

    }

    else{

        for(int i=cutoffInd; i<n; i++) cout<<" => "<<request[i];

        cout<<" => "<<size;

        for(int i=cutoffInd-1;i>=0;i--) cout<<" => "<<request[i];

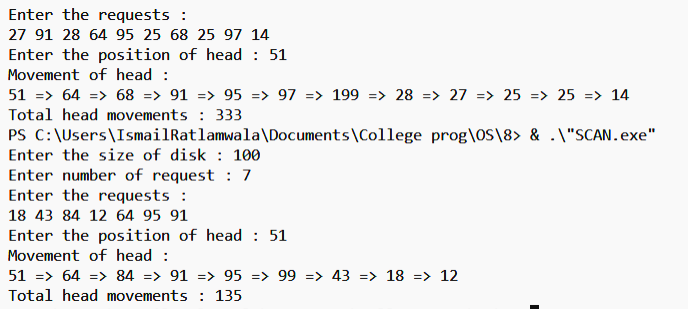
        cout<<"\nTotal head movements : "<<2\*size-head-request[0];

    }

    return 0;

}

**Output :**



**Program :**

#include <bits/stdc++.h>

using namespace std;

int main(void) {

    vector<vector<int>> allocation(5, vector<int>(3)),

                        maxNeed(5, vector<int>(3)),

                        remNeed(5, vector<int>(3));

    cout << "Enter number of resources of type A, B, C:\n";

    int a, b, c;

    cin >> a >> b >> c;

    cout << "\nEnter allocation matrix:\n";

    cout << "A B C\n";

    vector<int> totalAllocated(3);

    for(int i=0; i<5; i++) {

        for(int j=0; j<3; j++) {

            cin >> allocation[i][j];

            totalAllocated[j] += (allocation[i][j]);

        }

    }

    cout << "\nEnter max need matrix:\n";

    cout << "A B C\n";

    for(int i=0; i<5; i++) {

        for(int j=0; j<3; j++) {

            cin >> maxNeed[i][j];

        }

    }

    for(int i=0; i<5; i++) {

        for(int j=0; j<3; j++) {

            remNeed[i][j] = maxNeed[i][j] - allocation[i][j];

        }

    }

    vector<int> available(3);

    available[0] = a - totalAllocated[0];

    available[1] = b - totalAllocated[1];

    available[2] = c - totalAllocated[2];

    for(int it=0; it<5; it++) {

        bool deadlock = true;

        bool canExecute = true;

        for(int i=0; i<5; i++) {

            bool canExecute = true;

            for(int j=0; j<3; j++) {

                if(remNeed[i][j] > available[j]) {

                    canExecute = false;

                }

            }

            if(canExecute) {

                deadlock = false;

                for(int j=0; j<3; j++) {

                    available[j] += allocation[i][j];

                }

            }

        }

        if(deadlock) {

            cout << "Deadlock occurs!";

            return 0;

        }

    }

    cout << "\nRemaining need matrix:\n";

    for(int i=0; i<5; i++) {

        for(int j=0; j<3; j++) {

            cout << remNeed[i][j] << ' ';

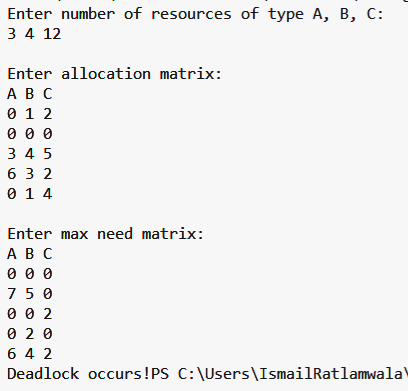
        }

        cout << endl;

    }

}

**Output :**

****

**Program :**

#include <bits/stdc++.h>

using namespace std;

int main(void) {

    cout << "Enter track size:\n";

    int t;

    cin >> t;

    cout << "\nEnter size of request queue:\n";

    int n;

    cin >> n;

    vector<int> a(n);

    cout << "\nEnter request queue:\n";

    for(int i=0; i<n; i++) {

        cin >> a[i];

    }

    sort(a.begin(), a.end());

    cout << "\nEnter starting position:\n";

    int starting;

    cin >> starting;

    cout << "\nChoose starting direction:\n1.Right\n2.Left\n";

    cout << "\nEnter choice:  ";

    int choice;

    cin >> choice;

    int total = 0;

    vector<int> ans;

    ans.push\_back(starting);

    if(choice == 1) {

        for(int i=0; i<n; i++) {

            if(a[i] >= starting) {

                ans.push\_back(a[i]);

            }

        }

        for(int i=n-1; i>=0; i--) {

            if(a[i] <= starting) {

                ans.push\_back(a[i]);

            }

        }

        total += (a[n-1] - starting + a[n-1] - a[0]);

    }

    if(choice == 2) {

        for(int i=n-1; i>=0; i--) {

            if(a[i] <= starting) {

                ans.push\_back(a[i]);

            }

        }

        for(int i=0; i<n; i++) {

            if(a[i] >= starting) {

                ans.push\_back(a[i]);

            }

        }

        total += (starting - a[0] + a[n-1] - a[0]);

    }

    cout << "\nHead movement is:\n";

    for(int i=0; i<ans.size(); i++) {

        cout << ans[i] << " -> ";

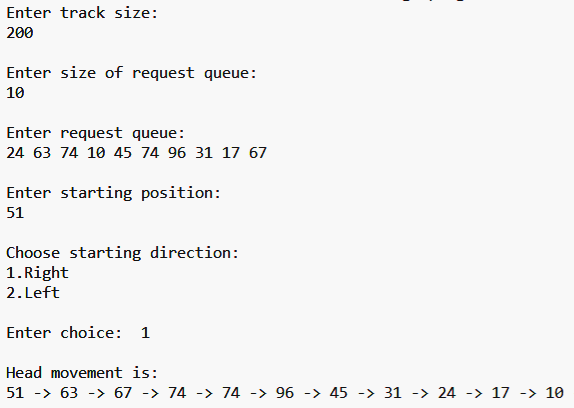
    }

    cout << "Stop\n";

    cout << "\nTotal head movement = " << total;

}

**Output :**

****