**FCFS**

package fcfs;

import java.util.\*;

import java.util.Scanner;

class Fcfs{

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

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

int n=sc.nextInt(); //Total processes

int pid[]= new int[n]; //Process ids

int at[]= new int[n];

int bt[]= new int[n];

int ct[]= new int[n];

int ta[]= new int[n];

int wt[]= new int[n];

int temp;

float avgwt=0,avgta=0;

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

{

System.out.print("Enter the process "+(i+1)+" arrival time : ");

at[i]=sc.nextInt();

System.out.print("Enter the process "+(i+1)+" Burst time: ");

bt[i]=sc.nextInt();

pid[i]=i+1;

}

//sort the process according to atrival Time

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

{

for(int j=0;j<n-(i+1);j++)

{

if(at[j]>at[j+1])

{

temp=at[j];

at[j]=at[j+1];

at[j + 1] = temp;

temp=bt[j];

bt[j]=bt[j+1];

bt[j+1]=temp;

temp=pid[j];

pid[j]=pid[j+1];

pid[j+1]=temp;

}

}

}

//find the completion time

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

{

if (i == 0)

ct[i] = at[i] + bt[i];

else {

if (at[i] > ct[i - 1]) {

ct[i] = at[i] + bt[i];

} else

ct[i] = ct[i - 1] + bt[i];

}

ta[i] = ct[i] - at[i];

wt[i] = ta[i] - bt[i];

avgta = avgta + ta[i];

avgwt = avgwt + wt[i];

}

System.out.println("\nPID AT BT CT TAT WT ");

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

System.out.println(pid[i] + "\t" + at[i] + "\t" + bt[i] + "\t" + ct[i] + "\t" + ta[i] + "\t" + wt[i]);

}

System.out.println(avgta/n);

System.out.println(avgwt/n);

sc.close();

}

}

**SJF**

package SJF;

import java.util.\*;

public class Sjf1 {

public static void main(String args[])

{

Scanner sc = new Scanner(System.***in***);

System.***out***.println ("enter no of process:");

int n = sc.nextInt();

int pid[] = new int[n];

int at[] = new int[n]; // at means arrival time

int bt[] = new int[n]; // bt means burst time

int ct[] = new int[n]; // ct means complete time

int ta[] = new int[n]; // ta means turn around time

int wt[] = new int[n]; //wt means waiting time

int f[] = new int[n]; // f means it is flag it checks process is completed or not

int st=0, tot=0;

float avgwt=0, avgta=0;

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

{

System.***out***.println ("enter process " + (i+1) + " arrival time:");

at[i] = sc.nextInt();

System.***out***.println ("enter process " + (i+1) + " brust time:");

bt[i] = sc.nextInt();

pid[i] = i+1;

f[i] = 0;

}

boolean a = true;

while(true)

{

int c=n, min=999;

if (tot == n) // total no of process = completed process loop will be terminated

break;

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

{

/\*

\* If i'th process arrival time <= system time and its flag=0 and burst<min

\* That process will be executed first\*/

if ((at[i] <= st) && (f[i] == 0) && (bt[i]<min))

{

min=bt[i];

c=i;

}

}

/\* If c==n means c value can not updated because no process arrival time< system time so we increase the system time \*/

if (c==n)

st++;

else

{

ct[c]=st+bt[c];

st+=bt[c];

ta[c]=ct[c]-at[c];

wt[c]=ta[c]-bt[c];

f[c]=1;

tot++;

}

}

System.***out***.println("\npid arrival brust complete turn waiting");

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

{

avgwt+= wt[i];

avgta+= ta[i];

System.***out***.println(pid[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+ta[i]+"\t"+wt[i]);

}

System.***out***.println ("\naverage tat is "+ (float)(avgta/n));

System.***out***.println ("average wt is "+ (float)(avgwt/n));

sc.close();

}

}

**Round Robin**

package roundRobin;

import java.util.Scanner;

public class RR {

public static void main(String args[])

{

int n,i,qt,count=0,temp,sq=0,bt[],wt[],tat[],rem\_bt[];

float awt=0,atat=0;

bt = new int[10];

wt = new int[10];

tat = new int[10];

rem\_bt = new int[10];

Scanner s=new Scanner(System.*in*);

System.*out*.print("Enter the number of process (maximum 10) = ");

n = s.nextInt();

System.*out*.print("Enter the burst time of the process\n");

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

{

System.*out*.print("P"+i+" = ");

bt[i] = s.nextInt();

rem\_bt[i] = bt[i];

}

System.*out*.print("Enter the quantum time: ");

qt = s.nextInt();

while(true)

{

for (i=0,count=0;i<n;i++)

{

temp = qt;

if(rem\_bt[i] == 0)

{

count++;

continue;

}

if(rem\_bt[i]>qt)

rem\_bt[i]= rem\_bt[i] - qt;

else

if(rem\_bt[i]>=0)

{

temp = rem\_bt[i];

rem\_bt[i] = 0;

}

sq = sq + temp;

tat[i] = sq;

}

if(n == count)

break;

}

System.*out*.print("--------------------------------------------------------------------------------");

System.*out*.print("\nProcess\t Burst Time\t Turnaround Time\t Waiting Time\n");

System.*out*.print("--------------------------------------------------------------------------------");

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

{

wt[i]=tat[i]-bt[i];

awt=awt+wt[i];

atat=atat+tat[i];

System.*out*.print("\n "+(i+1)+"\t "+bt[i]+"\t\t "+tat[i]+"\t\t "+wt[i]+"\n");

}

awt=awt/n;

atat=atat/n;

System.*out*.println("\nAverage waiting Time = "+awt+"\n");

System.*out*.println("Average turnaround time = "+atat);

}

}

**PRIORITY**

package priority;

import java.util.Scanner;

public class Priority {

public static void main(String args[]) {

Scanner s = new Scanner(System.***in***);

int x,n,p[],pp[],bt[],w[],t[],awt,atat,i;

p = new int[10];

pp = new int[10];

bt = new int[10];

w = new int[10];

t = new int[10];

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

n = s.nextInt();

System.***out***.print("\n\t Enter burst time : time priorities \n");

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

{

System.***out***.print("\nProcess["+(i+1)+"]:");

bt[i] = s.nextInt();

pp[i] = s.nextInt();

p[i]=i+1;

}

//sorting on the basis of priority

for(i=0;i<n-1;i++)

{

for(int j=i+1;j<n;j++)

{

if(pp[i]>pp[j])

{

x=pp[i];

pp[i]=pp[j];

pp[j]=x;

x=bt[i];

bt[i]=bt[j];

bt[j]=x;

x=p[i];

p[i]=p[j];

p[j]=x;

}

}

}

w[0]=0;

awt=0;

t[0]=bt[0];

atat=t[0];

for(i=1;i<n;i++)

{

w[i]=t[i-1];

awt+=w[i];

t[i]=w[i]+bt[i];

atat+=t[i];

}

//Displaying the process

System.***out***.print("\n\nProcess \t Burst Time \t Wait Time \t Turn Around Time Priority \n");

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

System.***out***.print("\n "+p[i]+"\t\t "+bt[i]+"\t\t "+w[i]+"\t\t "+t[i]+"\t\t "+pp[i]+"\n");

awt/=n;

atat/=n;

System.***out***.print("\n Average Wait Time : "+awt);

System.***out***.print("\n Average Turn Around Time : "+atat);

}

}

**FIRST FIT**

package firstFit;

import java.util.Arrays;

import java.util.Scanner;

public class FirstFit {

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

{

int allocation[] = new int[n];

for (int i = 0; i < allocation.length; 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;

}

}

}

System.*out*.println("\nProcess No.\tProcess Size\tBlock no.");

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

{

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

if (allocation[i] != -1)

System.*out*.print(allocation[i] + 1);

else

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

System.*out*.println();

}

}

public static void main(String[] args){

FirstFit first = new FirstFit();

Scanner scan = new Scanner(System.*in*);

System.*out*.println();

System.*out*.println("Enter the number of Blocks: ");

int m = scan.nextInt();

System.*out*.println("Enter the number of Processes: ");

int n = scan.nextInt();

int blockSize[] = new int[m];

int processSize[] = new int[n];

System.*out*.println("Enter the Size of all the blocks: ");

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

blockSize[i] = scan.nextInt();

}

System.*out*.println("Enter the size of all processes: ");

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

processSize[i] = scan.nextInt();

}

first.firstFit(blockSize,m,processSize,n);

}

}

**BEST FIT**

package BestFit;

import java.util.Arrays;

import java.util.Scanner;

public class best\_fit {

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

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++)

allocation[i] = -1;

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

// Find the best fit block for current process

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 we could find a block for current process

if (bestIdx != -1)

{

// allocate block j to p[i] process

allocation[i] = bestIdx;

// Reduce available memory in this block.

blockSize[bestIdx] -= processSize[i];

}

}

System.***out***.println("\nProcess No.\tProcess Size\tBlock no.");

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

{

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

if (allocation[i] != -1)

System.***out***.print(allocation[i] + 1);

else

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

System.***out***.println();

}

}

public static void main(String[] args){

best\_fit best = new best\_fit();

Scanner scan = new Scanner(System.***in***);

System.***out***.println();

System.***out***.println("Enter the number of Blocks: ");

int m = scan.nextInt();

System.***out***.println("Enter the number of Processes: ");

int n = scan.nextInt();

int blockSize[] = new int[m];

int processSize[] = new int[n];

System.***out***.println("Enter the Size of all the blocks: ");

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

blockSize[i] = scan.nextInt();

}

System.***out***.println("Enter the size of all processes: ");

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

processSize[i] = scan.nextInt();

}

best.bestFit(blockSize,m,processSize,n);

}

}

**WORST FIT**

package worstfit;

import java.util.Arrays;

import java.util.Scanner;

public class worst\_fit {

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

{

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++)

allocation[i] = -1;

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

// Find the best fit block for current process

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 we could find a block for current process

if (wstIdx != -1)

{

// allocate block j to p[i] process

allocation[i] = wstIdx;

// Reduce available memory in this block.

blockSize[wstIdx] -= processSize[i];

}

}

System.*out*.println("\nProcess No.\tProcess Size\tBlock no.");

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

{

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

if (allocation[i] != -1)

System.*out*.print(allocation[i] + 1);

else

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

System.*out*.println();

}

}

public static void main(String[] args){

worst\_fit worst = new worst\_fit();

Scanner scan = new Scanner(System.*in*);

System.*out*.println();

System.*out*.println("Enter the number of Blocks: ");

int m = scan.nextInt();

System.*out*.println("Enter the number of Processes: ");

int n = scan.nextInt();

int blockSize[] = new int[m];

int processSize[] = new int[n];

System.*out*.println("Enter the Size of all the blocks: ");

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

blockSize[i] = scan.nextInt();

}

System.*out*.println("Enter the size of all processes: ");

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

processSize[i] = scan.nextInt();

}

worst.worstFit(blockSize,m,processSize,n);

}

}

**NEXT FIT**

// Java program for next fit

// memory management algorithm

import java.util.Arrays;

public class GFG {

// Function to allocate memory to blocks as per Next fit

// algorithm

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

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n], j = 0;

// Initially no block is assigned to any process

Arrays.fill(allocation, -1);

// pick each process and find suitable blocks

// according to its size ad assign to it

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

// Do not start from beginning

int count =0;

while (j < m) {

count++; //makes sure that for every process we traverse through entire array maximum once only.This avoids the problem of going into infinite loop if memory is not available

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

// allocate block j to p[i] process

allocation[i] = j;

// Reduce available memory in this block.

blockSize[j] -= processSize[i];

break;

}

// mod m will help in traversing the blocks from

// starting block after we reach the end.

j = (j + 1) % m;

}

}

System.out.print("\nProcess No.\tProcess Size\tBlock no.\n");

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

System.out.print( i + 1 + "\t\t" + processSize[i]

+ "\t\t");

if (allocation[i] != -1) {

System.out.print(allocation[i] + 1);

} else {

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

}

System.out.println("");

}

}

// Driver program

static public void main(String[] args) {

int blockSize[] = {5, 10, 20};

int processSize[] = {10, 20, 5};

int m = blockSize.length;

int n = processSize.length;

NextFit(blockSize, m, processSize, n);

}

}

**COMBINE ( All Fits )**

import java.util.Arrays;  
class First  
{  
// Method to allocate memory to  
// blocks as per First fit algorithm  
static void firstFit(int blockSize[], int m, int processSize[], int n)

{  
// Stores block id of the  
// block allocated to a process  
int allocation[] = new int[n];  
// Initially no block is assigned to any process  
for (int i = 0; i < allocation.length; i++)  
allocation[i] = -1;  
// pick each process and find suitable blocks  
// according to its size ad assign to it  
for (int i = 0; i < n; i++)  
{  
for (int j = 0; j < m; j++)  
{  
if (blockSize[j] >= processSize[i])  
{  
// allocate block j to p[i] process  
allocation[i] = j;  
// Reduce available memory in this block.  
blockSize[j] -= processSize[i];  
break;  
}  
}}  
System.out.println("\nProcess No.\tProcess Size\tBlock no."); for (int i = 0; i < n; i++)  
{  
System.out.print(" " + (i+1) + "\t\t" +  
processSize[i] + "\t\t");  
if (allocation[i] != -1)  
System.out.print(allocation[i] + 1);  
else  
System.out.print("Not Allocated");  
System.out.println();  
}  
}

static void bestFit(int blockSize[], int m, int processSize[], int n)  
{  
// Stores block id of the block allocated to a

// process  
int allocation[] = new int[n];  
// Initially no block is assigned to any process  
for (int i = 0; i < allocation.length; i++)  
allocation[i] = -1;  
// pick each process and find suitable blocks  
// according to its size ad assign to it  
for (int i=0; i<n; i++)  
{  
// Find the best fit block for current process  
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 we could find a block for current process  
if (bestIdx != -1)  
{  
// allocate block j to p[i] process  
allocation[i] = bestIdx;  
// Reduce available memory in this block.  
blockSize[bestIdx] -= processSize[i];  
}  
}  
System.out.println("\nProcess No.\tProcess Size\tBlock no."); for (int i = 0; i < n; i++)  
{  
System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t"); if (allocation[i] != -1)  
System.out.print(allocation[i] + 1);  
else  
System.out.print("Not Allocated");  
System.out.println();  
}  
}  
static void worstFit(int blockSize[], int m, int processSize[], int n)  
{  
// Stores block id of the block allocated to a  
// process  
int allocation[] = new int[n];  
// Initially no block is assigned to any process  
for (int i = 0; i < allocation.length; i++)  
allocation[i] = -1;  
// pick each process and find suitable blocks

// according to its size ad assign to it  
for (int i=0; i<n; i++)  
{  
// Find the best fit block for current process 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 we could find a block for current process  
if (wstIdx != -1)  
{  
// allocate block j to p[i] process  
allocation[i] = wstIdx;  
// Reduce available memory in this block.  
blockSize[wstIdx] -= processSize[i];  
}  
}  
System.out.println("\nProcess No.\tProcess Size\tBlock no.");  
for (int i = 0; i < n; i++)  
{  
System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");  
if (allocation[i] != -1)  
System.out.print(allocation[i] + 1);  
else  
System.out.print("Not Allocated");  
System.out.println();  
}  
}  
static void NextFit(int blockSize1[], int m1, int processSize1[], int n1) {  
// Stores block id of the block allocated to a  
// process  
int allocation[] = new int[n1], j = 0;  
// Initially no block is assigned to any process  
Arrays.fill(allocation, -1);  
// pick each process and find suitable blocks  
// according to its size ad assign to it  
for (int i = 0; i < n1; i++) {  
// Do not start from beginning  
int count =0;  
while (j < m1) {  
count++; //makes sure that for every process we traverse through entire  
array maximum once only.This avoids the problem of going into infinite loop if memory is not available  
if (blockSize1[j] >= processSize1[i]) {  
// allocate block j to p[i] process  
allocation[i] = j;  
// Reduce available memory in this block.  
blockSize1[j] -= processSize1[i];

break;  
}  
// mod m will help in traversing the blocks from  
// starting block after we reach the end.  
j = (j + 1) % m1;  
}  
}  
System.out.print("\nProcess No.\tProcess Size\tBlock no.\n"); for (int i = 0; i < n1; i++) {  
System.out.print( i + 1 + "\t\t" + processSize1[i]  
+ "\t\t");  
if (allocation[i] != -1) {  
System.out.print(allocation[i] + 1);  
} else {  
System.out.print("Not Allocated");  
}  
System.out.println("");  
}  
}  
// Driver Code  
public static void main(String[] args)  
{  
System.out.println("....First Fit....");  
int blockSize[] = {100, 500, 200, 300, 600};  
int processSize[] = {212, 417, 112, 426};  
int m = blockSize.length;  
int n = processSize.length;  
firstFit(blockSize, m, processSize, n);  
/\* int blockSize[] = {100, 500, 200, 300, 600};  
int processSize[] = {212, 417, 112, 426};  
int m = blockSize.length;  
int n = processSize.length;\*/  
System.out.println(" ");  
System.out.println("....Best Fit....");  
bestFit(blockSize, m, processSize, n);  
System.out.println(" ");  
System.out.println("....Worst Fit....");  
worstFit(blockSize, m, processSize, n);  
System.out.println(" ");  
System.out.println("....Next Fit....");  
int blockSize1[] = {5, 10, 20};  
int processSize1[] = {10, 20, 5};  
int m1 = blockSize1.length;  
int n1 = processSize1.length;  
NextFit(blockSize1, m1, processSize1, n1);  
}  
}