```
/***********************/
q1_1.c
/***********************/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int a;
        a=fork();
        if(a<0)
        {
                printf("Child Process could not be Created");
                exit(-1);
        }
        else //Child Process Created Successfully
        {
                printf("My ID is:%d,My Parent ID is:%d\n",getpid(),getppid());
        }
        return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q1_1 q1_1.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q1_1
My ID is:5435,My Parent ID is:5411
My ID is:5436,My Parent ID is:5435
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q1_2.c
/***********************/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int a;
        a=fork();
        if(a<0)
        {
                printf("Child Process could not be Created");
                exit(-1);
        }
        else if(a==0) //In Child Process
        {
                execl("/bin/ls","ls\n",NULL);
        }
        else //Parent Process
        {
                printf("\nIn Parent Process\n");
        }
        return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q1_2 q1_2.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q1_2

In Parent Process
cyborg@cyborg:~/Desktop/OSprogs$ final   OSprogs.zip  q1_1    q1_2    q12.c   q13.c  q3.c  q5.c  q7.c  q9.c    track.txt
join.sh  q10.c       q1_1.c  q1_2.c  q1_3.c  q2.c   q4.c  q6.c  q8.c  run.sh

cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q1_3.c
/***********************/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int a;
        a=fork();
        if(a<0)
        {
                printf("Child Process could not be Created");
                exit(-1);
        }
        else if(a==0) //In Child Process
        {
                printf("\nIn Child Process\n");
        }
        else //Parent Process
        {
                wait(NULL);
                printf("\nIn Parent Process\n");
        }
        return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q1_3 q1_3.c
q1_3.c: In function ‘main’:
q1_3.c:19:3: warning: implicit declaration of function ‘wait’ [-Wimplicit-function-declaration]
   wait(NULL);
   ^
cyborg@cyborg:~/Desktop/OSprogs$ ./q1_3

In Child Process

In Parent Process
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q2.c
/***********************/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int a;
        printf("\nKernel Version:\n");
        system("cat /proc/sys/kernel/osrelease");
        printf("\nCPU TYPE & MODEL:\n");
        system("cat /proc/cpuinfo | awk 'NR==4,NR==5 {print}'");
        return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q2 q2.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q2

Kernel Version:
4.4.0-45-generic

CPU TYPE & MODEL:
Intel Core i7-6700K CPU @ 4.2GHz
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q3.c
/***********************/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int a;
        printf("\nKernel Version:\n");
        system("cat /proc/sys/kernel/osrelease");
        printf("\nInformation on Configured amount of free and Used Memory:\n");
        system("cat /proc/meminfo | awk 'NR==1,NR==2 {print}'");
        return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q3 q3.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q3

Kernel Version:
4.4.0-45-generic

Information on Configured amount of free and Used Memory:
MemTotal:       16251284 kB
MemFree:         1145832 kB
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q4.c
/***********************/

#include<stdio.h>
#include<sys/stat.h>
#include<time.h>
int main(int argc,char *argv[3])
{
        int i;
        struct stat buffer;
        printf("GIve File Name:");
        for(i=1;i<argc;i++)
        {
                printf("file=%s\n",argv[i]);
                if(stat(argv[i],&buffer)<0)
                        printf("Error in File Started");
                else
                {
                        printf("Ownered=%d\ngid=%d\n",buffer.st_uid,buffer.st_gid);
                        printf("Access Permission=%d\n",buffer.st_mode);
                        printf("Access Time=%d\n",(time(&(buffer.st_atime))));
                }
}
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ ./q4 q4.c
GIve File Name:file=q4.c
Ownered=1000
gid=1000
Access Permission=33204
Access Time=1478005512
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q5.c
/***********************/

#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
//#include<fcltl.h>
void copy(int,int);
void display(int);
main(int argc,char *argv[])
{
        int fold,fnew;
        if(argc!=3)
        {
                printf("Two Arguments Required");
                exit(1);
        }
        fold=open(argv[1],0);
        if(fold==-1)
        {
                printf("Unable to Open the File\n%s",argv[1]);
                exit(1);
        }
        fnew=creat(argv[2],0666);
        if(fnew==-1)
        {
                printf("Unable to Create the File%s\n",argv[2]);
                exit(1);
        }
        copy(fold,fnew);
        exit(0);
        close(fold);
        close(fnew);
        fnew=open(argv[2],0);
        printf("New File:\n");
        display(fnew);
        close(fnew);
        exit(0);
}
void copy(int old,int new)
{
        int count=0;
        char buffer[512];
        while((count=read(old,buffer,sizeof(buffer)))>0)
        {
```

```
                write(new,buffer,count);
        }
}
void display(int fnew)
{
                int count=0,i;
                char buffer[512];
                while((count=read(fnew,buffer,sizeof(buffer)))>0)
                {
                        for(i=0;i<count;i++)
                        {
                                printf("%c",buffer[i]);
                        }
                }
                for(i=0;i<count;i++)
                {
                        printf("%c",buffer[i]);
                }
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ cat demo.txt
this is a demo file.
cyborg@cyborg:~/Desktop/OSprogs$ cat newdemo.txt
cat: newdemo.txt: No such file or directory
cyborg@cyborg:~/Desktop/OSprogs$ ./q5 demo.txt newdemo.txt
cyborg@cyborg:~/Desktop/OSprogs$ cat newdemo.txt
this is a demo file.
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q6.c
/***********************/

//FCFS
#include<stdio.h>
int tim=0;
void main()
{
	int n,b[20],i,j,w[20],tw=0,taround[20],tt=0;
	float avw,avt;
	printf("Enter the No. of Processes=");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	{
		printf("Enter the Burst Time of %dth Process=",i);
		scanf("%d",&b[i]);
	}
	for(i=1;i<=n;i++)
	{
		w[i]=tim;
		for(j=1;j<=b[i];j++)
		{
			tim++;
			if(j==b[i])
				taround[i]=tim;
		}
	}

	for(i=1;i<=n;i++)
	{
		tw=tw+w[i];
	}
	avw=(float)tw/n;
	for(i=1;i<=n;i++)
	{
		tt=tt+taround[i];
	}
	avt=(float)tt/n;
	printf("\nWaiting & Turn Around Time");
	printf("\n**************************");
	for(i=1;i<=n;i++)
	{
		printf("\nProcess %d-Waiting=%d,Turn Around=%d",i,w[i],taround[i]);
	}
	printf("\nAverage Waiting Time=%f",avw);
	printf("\nAverage Turn Around Time=%f\n",avt);
```

}

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q6 q6.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q6
Enter the No. of Processes=5
Enter the Burst Time of 1th Process=3
Enter the Burst Time of 2th Process=5
Enter the Burst Time of 3th Process=2
Enter the Burst Time of 4th Process=4
Enter the Burst Time of 5th Process=2

Waiting & Turn Around Time
**************************
Process 1-Waiting=0,Turn Around=3
Process 2-Waiting=3,Turn Around=8
Process 3-Waiting=8,Turn Around=10
Process 4-Waiting=10,Turn Around=14
Process 5-Waiting=14,Turn Around=16
Average Waiting Time=7.000000
Average Turn Around Time=10.200000
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q7.c
/***********************/

//ROUND ROBIN
#include<stdio.h>
int main()
{
int count,j,n,time,remain,flag=0,time_quantum;
int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
printf("Enter Total Process:\t ");
scanf("%d",&n);
remain=n;
for(count=0;count<n;count++)
{
printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
scanf("%d",&at[count]);
scanf("%d",&bt[count]);
rt[count]=bt[count];
}
printf("Enter Time Quantum:\t");
scanf("%d",&time_quantum);
printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
for(time=0,count=0;remain!=0;)
{
if(rt[count]<=time_quantum && rt[count]>0)
{
time+=rt[count];
rt[count]=0;
flag=1;
}
else if(rt[count]>0)
{
rt[count]-=time_quantum;
time+=time_quantum;
}

if(rt[count]==0 && flag==1)
{
remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
wait_time+=time-at[count]-bt[count];
turnaround_time+=time-at[count];
flag=0;
}
if(count==n-1)
count=0;
else if(at[count+1]<=time)
```

```
count++;
else
count=0;
}
printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q7 q7.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q7
Enter Total Process:     5
Enter Arrival Time and Burst Time for Process Process Number 1 :0
3
Enter Arrival Time and Burst Time for Process Process Number 2 :2
5
Enter Arrival Time and Burst Time for Process Process Number 3 :4
2
Enter Arrival Time and Burst Time for Process Process Number 4 :5
4
Enter Arrival Time and Burst Time for Process Process Number 5 :9
2
Enter Time Quantum:     2


Process |Turnaround Time|Waiting Time

P[3]    |       2       |       0
P[1]    |       9       |       6
P[4]    |       8       |       4
P[5]    |       6       |       4
P[2]    |       14      |       9

Average Waiting Time= 4.600000
Avg Turnaround Time = 7.800000cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q8.c
/***********************/

#include<stdio.h>
void main()
{
int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);
printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1; //contains process number
}
//sorting burst time in ascending order using selection sort
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0; //waiting time for first process will be zero
//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=(float)total/n; //average waiting time
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
```

```
tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];
printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n; //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q8 q8.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q8
Enter number of process:5

Enter Burst Time:
p1:3
p2:5
p3:2
p4:4
p5:2

Process     Burst Time          Waiting Time    Turnaround Time
p3                2                   0                   2
p5                2                   2                   4
p1                3                   4                   7
p4                4                   7                   11
p2                5                   11                  16

Average Waiting Time=4.800000
Average Turnaround Time=8.000000
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q9.c
/***********************/

#include<stdio.h>
int main()
{
int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter Total Number of Process:");
scanf("%d",&n);
printf("\nEnter Burst Time and Priority\n");
for(i=0;i<n;i++)
{
printf("\nP[%d]\n",i+1);
printf("Burst Time:");
scanf("%d",&bt[i]);
printf("Priority:");
scanf("%d",&pr[i]);
p[i]=i+1; //contains process number
}
//sorting burst time, priority and process number in ascending order using selection sort
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(pr[j]<pr[pos])
pos=j;
}
temp=pr[i];
pr[i]=pr[pos];
pr[pos]=temp;
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0; //waiting time for first process is zero
//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
```

```
}
avg_wt=(float)total/n; //average waiting time
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];
printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n; //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q9 q9.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q9
Enter Total Number of Process:5

Enter Burst Time and Priority

P[1]
Burst Time:3
Priority:2

P[2]
Burst Time:5
Priority:1

P[3]
Burst Time:2
Priority:4

P[4]
Burst Time:4
Priority:5

P[5]
Burst Time:2
Priority:3

Process     Burst Time          Waiting Time    Turnaround Time
P[2]              5                 0                 5
P[1]              3                 5                 8
P[5]              2                 8                 10
P[3]              2                 10                12
P[4]              4                 12                16

Average Waiting Time=7.000000
Average Turnaround Time=10.200000
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q10.c
/***********************/

#include<stdio.h>
int main()
{
int i,j,n,time,sum_wait=0,sum_turnaround=0,smallest;
int at[10],bt[10],pt[10],rt[10],remain; //rt = remaining Time
printf("Enter no of Processes : ");
scanf("%d",&n);
remain=n;
for(i=0;i<n;i++)
{
printf("Enter arrival time, burst time and priority for process p%d :",i+1);
scanf("%d",&at[i]);
scanf("%d",&bt[i]);
scanf("%d",&pt[i]);
rt[i]=bt[i];
}
pt[9]=11;
printf("\n\nProcess\t|Turnaround time|waiting time\n");
for(time=0;remain!=0;time++)
{
smallest=9;
for(i=0;i<n;i++)
{
if(at[i]<=time && pt[i]<pt[smallest] && rt[i]>0)
{
smallest=i;
}
}
rt[smallest]--;
if(rt[smallest]==0)
{
remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",smallest+1,time+1-at[smallest],time+1-at[smallest]-bt[smallest]);
sum_wait+=time+1-at[smallest];
sum_turnaround+=time+1-at[smallest]-bt[smallest];
}
}
printf("\nAvg waiting time = %f\n",sum_wait*1.0/n);
printf("Avg turnaround time = %f",sum_turnaround*1.0/n);
return 0;
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q10 q10.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q10
Enter no of Processes : 5
Enter arrival time, burst time and priority for process p1 :0
3
2
Enter arrival time, burst time and priority for process p2 :2
5
1
Enter arrival time, burst time and priority for process p3 :4
2
4
Enter arrival time, burst time and priority for process p4 :5
4
5
Enter arrival time, burst time and priority for process p5 :9
2
3


Process |Turnaround time|waiting time
P[2]    |       5       |       0
P[1]    |       8       |       5
P[5]    |       2       |       0
P[3]    |       8       |       6
P[4]    |       11      |       7

Avg waiting time = 6.800000
Avg turnaround time = 3.600000cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q12.c
/***********************/

#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
int sum; //data shared among threads
void *runner(void *param); //the thread *I
int main(int argc,char *argv[])
{
        //thread calculates the sum of numbers from
        //1 to argv[1](an integer)
        pthread_t tid; //thread identifier *I
        pthread_attr_t attr; //set of thread attributes
        if(argc!=2)
        {
                fprintf(stderr,"usage:a.out<integervalue>\n");
                return -1;
        }
        if(atoi(argv[1])<0)
        {
                fprintf(stderr,"%d must be >=0\n",atoi(argv[1]));
                return -1;
        }
        pthread_attr_init(&attr); //get default attributes *I
        pthread_create(&tid,&attr,runner,argv[1]);
        pthread_join(tid,NULL); //wait for thread to exit *I
        printf("SUM=%d\n",sum);
        return 0;
}
void *runner(void *param)
{
        int i,upper=atoi(param);
        sum=0;
        for(i=1;i<=upper;i++)
                sum+=i;
        pthread_exit(0);
}
```

```
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q12 -pthread q12.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q12 5
SUM=15
cyborg@cyborg:~/Desktop/OSprogs$
```

```
/***********************/
q13.c
/***********************/

/* WRITE A MENU DRIVEN C PROGRAM TO IMPLEMENT MEMORY MANAGEMENT
ALGORITHMS. 1.FIRST FIT 2.BEST FIT 3.WORST FIT*/
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

//function to enter values in array

void accept(int a[],int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
}

//function to display array

void display(int a[],int n)
{
	int i;
	printf("\n\n");
	for(i=0;i<n;i++)
	{
		printf("\t%d ",a[i]);
	}
}

//function to sort given array

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

```
                }
        }
}

//reverse sort

void revsort(int a[],int n)
{
        int i,j,temp;
        for(i=0;i<n-1;i++)
        {
                for(j=0;j<n-1;j++)
                {
                        if(a[j]<a[j+1])
                        {
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                        }
                }
        }
}

// first fit algo

void first_fit(int psize[],int np,int msize[],int nm)
{
                int i,j,in_fr,ex_fr,flag[30]={0}; // in_fr : internal fragmebtation, ex_fr : external
fragmentation
                in_fr=ex_fr=0;
        for(i=0;i<np;i++) //loop to check space for process
        {
                for(j=0;j<nm;j++)
                {
/*
* the first fit option finds the "first largest hole" for a process that is given, hence if
* process size is smaller than the memory hole size, just place it and move to next process
block.
*/
                        if(flag[j]==0 && msize[j]>=psize[i]) //initially flag[] = NULL or 0, after every
condition fullfilment flag[j]=1
                        {
                                flag[j]=1;
                                in_fr=in_fr+msize[j]-psize[i]; //calculate internal fragmentation (left
over space after placing process)
                                break;
                        }
                }
```

```
		if(j==nm) //upper loop breaks if there is no space for a particular process
			printf("\n\nTHERE IS NO SPACE FOR PROCESS %d ",i);
	}
	for(i=0;i<nm;i++) //calculate external fragmenation, its is noting but the left over memory
hole in the memory block.
	{
		if(flag[i]==0)
			ex_fr=ex_fr+msize[i];
	}
	printf("\n\nPROCESSES::");
	display(psize,np);
	printf("\n\nMEMORY HOLES::");
	display(msize,nm);
	printf("\n\nTOTAL SUM OF INTERNAL FRAGMENTATION = %d ",in_fr);
	printf("\n\nTOTAL SUM OF EXTERNAL FRAGMENTATION = %d ",ex_fr);

}
void best_fit(int psize[],int np,int msize[],int nm)
{
	int i,j,in_fr,ex_fr,temp[30],flag[30]={0};
	in_fr=ex_fr=0;

	for(i=0;i<nm;i++)
		temp[i]=msize[i];
/*
* since we want to have best fit, sort the memory block, once
* sorted we can start placing the process where we first find space for it.
*/
	sort(temp,nm);
	for(i=0;i<np;i++)
	{
		for(j=0;j<nm;j++)
		{
/*
* the best fit option uses a sorted array of memory holes for a process that is given, hence if
* process size is smaller than the memory hole size, just place it and move to next process
block.
*/
			if(flag[j]==0 && temp[j]>=psize[i])
			{
				flag[j]=1;
				in_fr=in_fr+temp[j]-psize[i];
				break;
			}
		}
		if(j==nm) //rest same as first fit
			printf("\n\nTHERE IS NO SPACE FOR PROCESS %d ",i);
	}
```

```
	for(i=0;i<nm;i++)
	{
		if(flag[i]==0)
			ex_fr=ex_fr+temp[i];
	}
	printf("\n\nPROCESSES::");
	display(psize,np);
	printf("\n\nMEMORY HOLES::");
	display(temp,nm);
	printf("\n\nTOTAL SUM OF INTERNAL FRAGMENTATION = %d ",in_fr);
	printf("\n\nTOTAL SUM OF EXTERNAL FRAGMENTATION = %d ",ex_fr);
}
void worst_fit(int psize[],int np,int msize[],int nm)
{
	int i,j,in_fr,ex_fr,temp[30],flag[30]={0};
	in_fr=ex_fr=0;
	for(i=0;i<nm;i++)
		temp[i]=msize[i];
/*
* worst fit finds the biggest possible memory hole for the coming process,
* hence let's sort the memory block in reverse order. Now if 1st process comes,
* it will get the biggest possible(first memory hole in reversed array) memory hole
* and this will go on.
*/
	revsort(temp,nm);
	for(i=0;i<np;i++)
	{
/* The array is sorted, so just start placing ;) */
		for(j=0;j<nm;j++)
		{
			if(flag[j]==0 && temp[j]>=psize[i])
			{
				flag[j]=1;
				in_fr=in_fr+temp[j]-psize[i];
				break;
			}
		}
		if(j==nm) //rest same as first fit
			printf("\n\nTHERE IS NO SPACE FOR PROCESS %d ",i);
	}
	for(i=0;i<nm;i++)
	{
		if(flag[i]==0)
			ex_fr=ex_fr+temp[i];
	}
	printf("\n\nPROCESSES::");
	display(psize,np);
	printf("\n\nMEMORY HOLES::");
```

```
        display(temp,nm);
        printf("\n\nTOTAL SUM OF INTERNAL FRAGMENTATION = %d ",in_fr);
        printf("\n\nTOTAL SUM OF EXTERNAL FRAGMENTATION = %d ",ex_fr);

}

/*************************************************************************/
void main()
{
        int ch,np,nm,psize[30],msize[30];
        printf("\nENTER NO OF PROCESSES::");
        scanf("%d",&np);
        printf("\n\nENTER SIZES OF PROCESSES::");
        accept(psize,np);
        printf("\nENTER NO MEMORY HOLES::");
        scanf("%d",&nm);
        printf("\n\nENTER SIZES OF MEMORY HOLES::");
        accept(msize,nm);

        while(1)
        {
                printf("\n\n\t\t**MAIN MENU**");
                printf("\n\n\tMEMORY MANAGEMENT");
                printf("\n\n\t1.FIRST FIT");
                printf("\n\n\t2.BEST FIT");
                printf("\n\n\t3.WORST FIT");
                printf("\n\n\t4.QUIT");

                printf("\n\nENTER YOUR CHOICE::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                printf("\n\nFIRST FIT::\n");
                                first_fit(psize,np,msize,nm);
                                break;
                        case 2:
                                printf("\n\n\tBEST FIT::\n");
                                best_fit(psize,np,msize,nm);
                                break;
                        case 3:
                                printf("\n\n\tWORST FIT::\n");
                                worst_fit(psize,np,msize,nm);
                                break;
                        case 4:
                                exit(0);
                        default:
                                printf("\n\nPLEASE ENTER CORRECT CHOICE!!");
```

```
            }
        }
}
cyborg@cyborg:~/Desktop/OSprogs$ gcc -o q13 q13.c
cyborg@cyborg:~/Desktop/OSprogs$ ./q13
ENTER NO OF PROCESSES::4
ENTER SIZES OF PROCESSES::500 600 400 100
ENTER NO MEMORY HOLES::4
ENTER SIZES OF MEMORY HOLES::300 200 500 100

                **MAIN MENU**
        MEMORY MANAGEMENT
        1.FIRST FIT
        2.BEST FIT
        3.WORST FIT
        4.QUIT
ENTER YOUR CHOICE::1
        FIRST FIT::


THERE IS NO SPACE FOR PROCESS 1

THERE IS NO SPACE FOR PROCESS 2

PROCESSES::

        500     600     400     100

MEMORY HOLES::

        300     200     500     100

TOTAL SUM OF INTERNAL FRAGMENTATION = 200

TOTAL SUM OF EXTERNAL FRAGMENTATION = 300
                **MAIN MENU**
        MEMORY MANAGEMENT
        1.FIRST FIT
        2.BEST FIT
        3.WORST FIT
        4.QUIT
ENTER YOUR CHOICE::
```