1. Consider a system with 4 processes and 3 resources with the given resource matrices.

Claim matrix	Allocation matrix
3 2 2	1 0 0
6 1 3	6 1 2
3 1 4	2 1 1
4 2 2	0 0 2

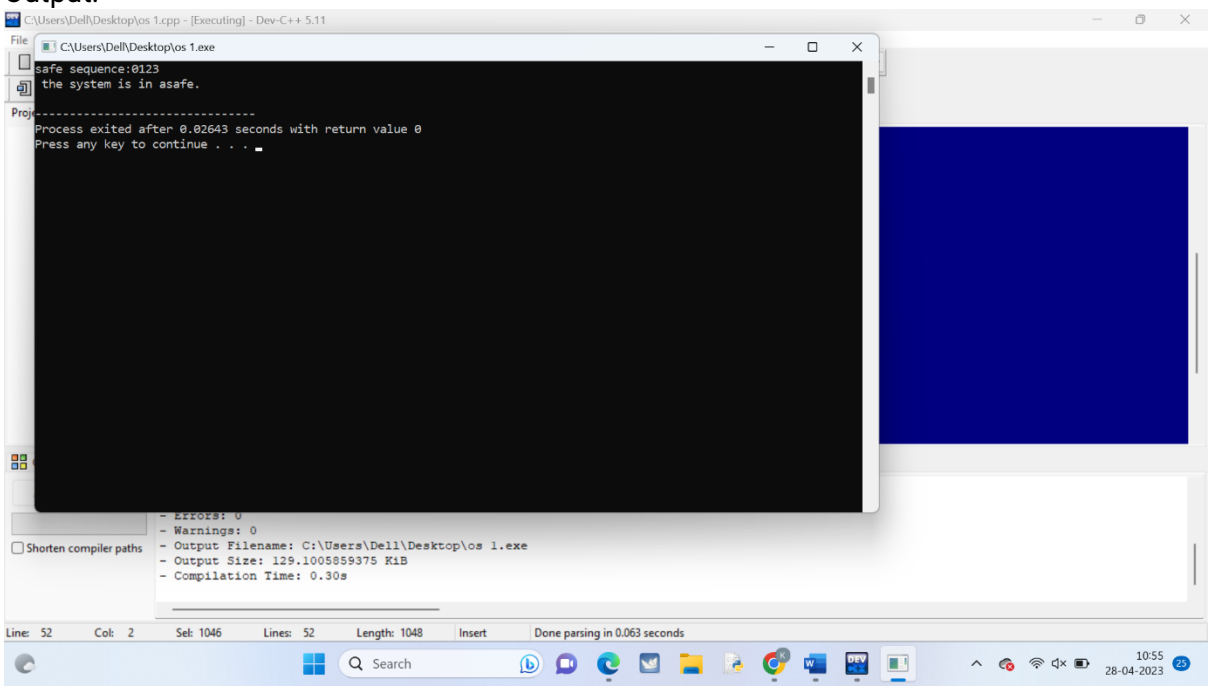
The resource vector is [9,3,6]. Write a C program to determine if the system is in safe or unsafe state.

Program:

```
#include<stdio.h>
int main()
{
	int claim[4][3]={{3,2,2},{6,1,3},{3,1,4},{4,2,2}};
	int allo[4][3]={{1,0,0},{6,1,2},{2,1,1},{0,0,2}};
	int res[3]={9,3,6};
	int ava[3]={0,0,0};
	for(int j=0;j<3;j++){
		for(int i=0;i<4;i++){
			ava[j]=allo[i][j];
		}
		ava[j]=res[j]-ava[j];
	}
	int finish[4]={0,0,0,0};
	int safe_seq[4];
	int num_fin=0;
	while(num_fin<4){
		int safe_found=0;
		for(int i=0;i<4;i++){
			if(!finish[i]){
				int can_finish=1;
				for(int j=0;j<3;j++){
					if(claim[i][j]-allo[i][j]>ava[j]){
						can_finish=0;
						break;
					}
				}
				if(can_finish){
					safe_seq[num_fin]=i;
					num_fin++;
```

```
                                        finish[i]=1;
                                        for(int j=0;j<3;j++){
                                                ava[j]+=allo[i][j];
                                        }
                                        safe_found=1;
                                }
                        }
                }
                if(!safe_found){
                        break;
                }       }
                if(num_fin==4){
                        printf("safe sequence:");
                        for(int i=0;i<4;i++){
                                printf("%d",safe_seq[i]);
                                                }
                                                printf("\n the system is in asafe.\n");
                }else{
                printf("\n the syastem is in an unsafe state\n");
        }
        return 0;
}
```

Output:



2. Write a C program to illustrate the FIFO method of page replacement and determine the number of page faults for the following test case:

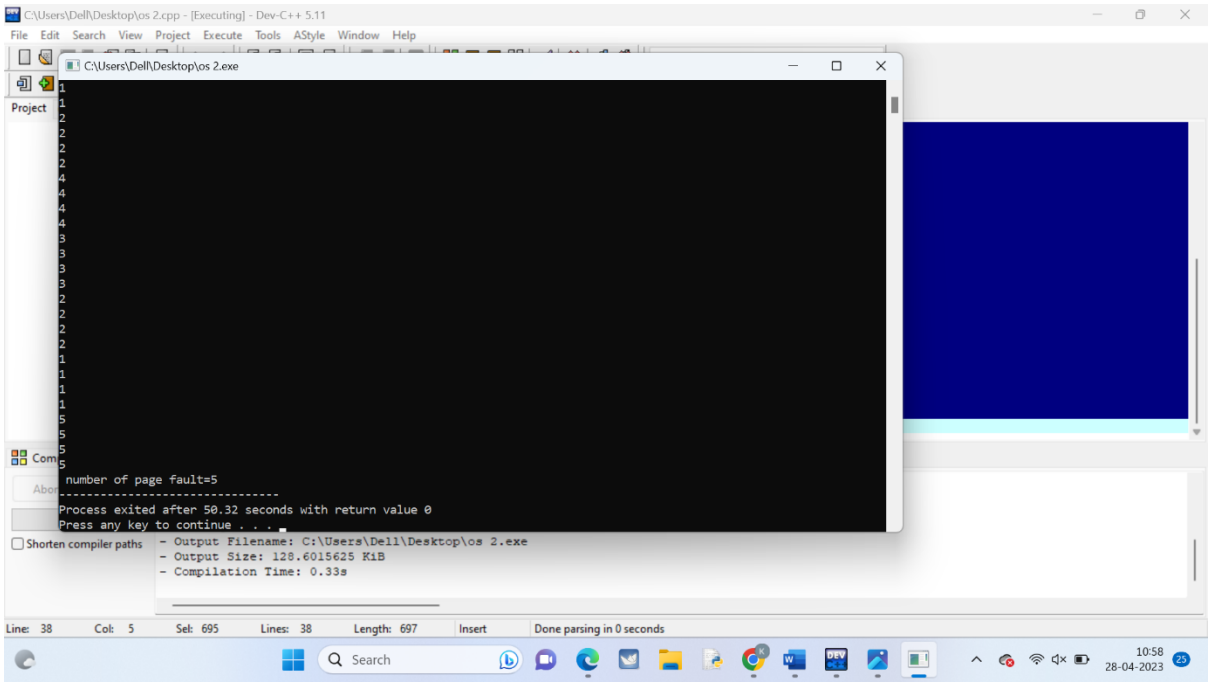
No of page frames: 3; Page reference sequence: 4, 1, 2, 4, 3, 2, 1 and 5.

Program:

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

```
int main()
{
	int n,frames[10],page[30],page_fault=0,first=0,last=0,found=0;
	printf("enter the number of page frame:");
	scanf("%d",&n);
	printf("enter the page reference sequence:");
	for(int i=0;i<8;i++)
	{
		scanf("%d",&page[i]);
	}
	for(int i=0;i<n;i++)
	{
		frames[i]=-1;
	}
	for(int i=0;i<8;i++){
		found=0;
		for(int j=0;j<n;j++){
			if(frames[j]==page[i])
			{
				found=1;
				break;
			}
		}
		if(found==0){
			page_fault++;
			frames[last]=page[i];
			last=(last+1)%n;
		}
		printf("\n%d\t",page[i]);
		for(int j=0;j<n;j++){
			printf("\n%d\t",page[i]);
					}
	}
	printf("\n number of page fault=%d",page_fault);
	return 0;
}
```

Output:



3. Write a program to compute the average waiting time and average turnaround time based on Non Preemptive Shortest-Job-First Scheduling for the following process with the given CPU burst times, (and the assumption that all jobs arrive at the same time.)

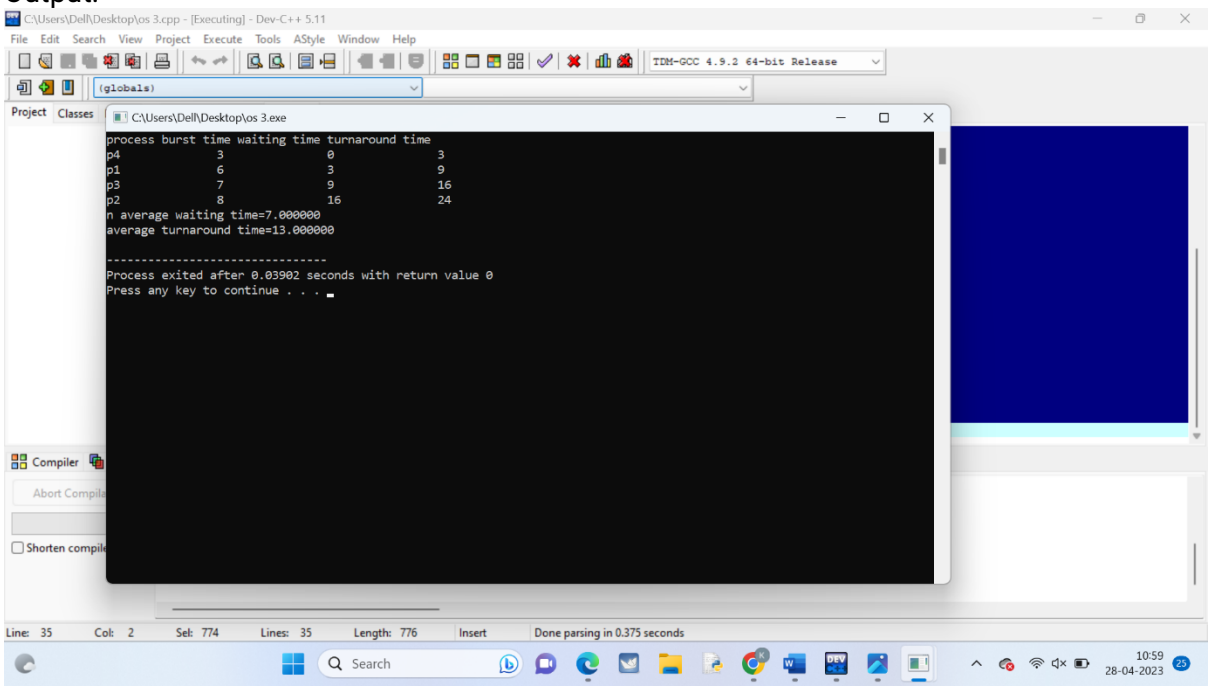
Process	Burst Time
P1	6
P2	8
P3	7
P4	3

Program:

```
#include<iostream>
int main(){
        int n=4;
        int bt[]={6,8,7,3};
        int p[]={1,2,3,4};
```

```
        int wt[n],tat[n],total_wt=0,total_tat=0;
        for(int i=0;i<n;i++){
                for(int j=i+1;j<n;j++){
                        if(bt[i]>bt[j]){
                                int temp_bt=bt[i];
                                bt[i]=bt[j];
                                bt[j]=temp_bt;
                                int temp_p=p[i];
                                p[i]=p[j];
                                p[j]=temp_p;
                        }
                }
        }
        wt[0]=0;
        for(int i=1;i<n;i++){
                wt[i]=wt[i-1]+bt[i-1];
        }
        for(int i=0;i<n;i++){
                tat[i]=bt[i]+wt[i];
        }
        printf("process burst time waiting time turnaround time\n");
        for(int i=0;i<n;i++){
        total_wt+=wt[i];
        total_tat+=tat[i];
        printf("p%d\t\t%d\t\t%d\t\t%d\n",p[i],bt[i],wt[i],tat[i]);
}
printf("n average waiting time=%2f\n",(float)total_wt/n);
printf("average turnaround time=%2f\n",(float)total_tat/n);
return 0;
}
```

Output:



4. Write a C program to implement the first-fit algorithm for memory management.

Test Case:

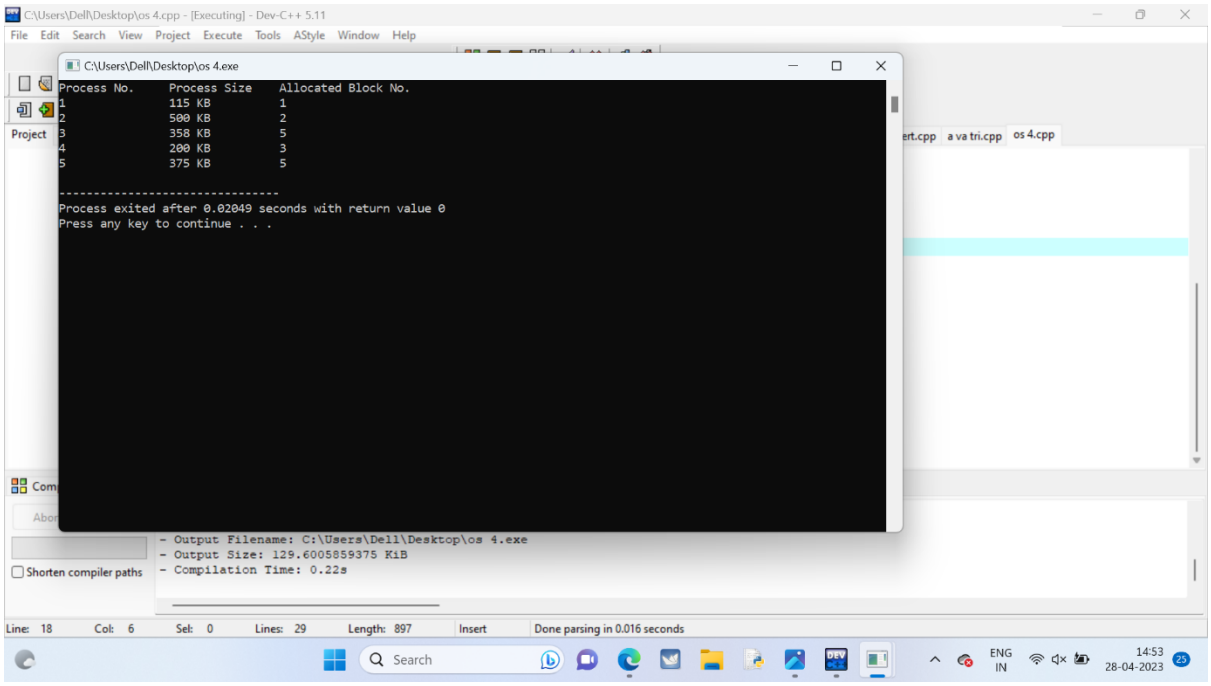
Memory partitions: 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order) Show the outcome for the test case with first-fit algorithms to place the processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)

Program:

```
#include <stdio.h>
int main() {
   int memory[] = {300, 600, 350, 200, 750, 125};
   int n = sizeof(memory)/sizeof(memory[0]);
   int process[] = {115, 500, 358, 200, 375};
         int m = sizeof(process)/sizeof(process[0]);
   int allocation[m];
   for (int i = 0; i < m; i++)
      allocation[i] = -1;
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         if (memory[j] >= process[i]) {
            allocation[i] = j;
            memory[j] -= process[i];
            break;
         }
      }
   }
   printf("Process No.\tProcess Size\tAllocated Block No.\n");
   for (int i = 0; i < m; i++) {
      printf("%d\t\t%d KB\t\t", i+1, process[i]);
      if (allocation[i] != -1)
         printf("%d\n", allocation[i]+1);
      else
         printf("Not Allocated\n");
   }

   return 0;
}
```

Output:


C:\Users\Dell\Desktop\os 4.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
C:\Users\Dell\Desktop\os 4.exe
Process No. Process Size Allocated Block No.
1 115 KB 1
2 500 KB 2
3 358 KB 5
4 200 KB 3
5 375 KB 5
Process exited after 0.02049 seconds with return value 0
Press any key to continue . . .
Project
os 4.cpp
- Output Filename: C:\Users\Dell\Desktop\os 4.exe
- Output Size: 129.6005859375 KiB
- Compilation Time: 0.22s
Shorten compiler paths
Line: 18 Col: 6 Sel: 0 Lines: 29 Length: 897 Insert
Done parsing in 0.016 seconds
Search
ENG
IN
14:53
28-04-2023