Program

```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <errno.h>

#define BUFFER_SIZE 1024

int main(int argc, char *argv[])
{

    if (argc < 3)
    {
        printf("Usage: %s <source> <destination>\n", argv[0]);
        exit(1);
    }

    FILE *fp1, *fp2;
    char buf[BUFFER_SIZE];

    fp1 = fopen(argv[1], "r");
    fp2 = fopen(argv[2], "w");

    while (fgets(buf, BUFFER_SIZE, fp1) != NULL)
        fputs(buf, fp2);

    return 0;
}
```

Output

```
os-lab> cat a.txt
Graph Theory
Operating Systems
Computer Architecture and Organization
DBMS
Constitution of India
Professional Ethics
os-lab> cat b.txt
os-lab> ./a.out a.txt b.txt
os-lab> cat b.txt
Graph Theory
Operating Systems
Computer Architecture and Organization
DBMS
Constitution of India
Professional Ethics
```

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <errno.h>

int main(int argc, char *argv[])
{
    DIR *p;
    struct dirent *d;

    p = opendir(argv[1]);
    if (p == NULL)
    {
        printf("Error: %s\n", strerror(errno));
        exit(1);
    }

    while ((d = readdir(p)) != NULL)
        printf("%s\t", d->d_name);
    printf("\n");

    return 0;
}
```

Output

```
os-lab> ./a.out /
boot    media   dev     opt     var     lib64   bin     home    libx32
lib32   .       etc     root    srv     sbin    proc    init    lost+found
sys     usr     Docker  lib     snap    ..      run     mnt     tmp
os-lab> ./a.out .
.       ..      a.out   cp.c    grep.c  ls.c
```

Program

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

#define BUFFER_SIZE 1024

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        printf("Usage: %s <pattern> <filename>\n", argv[0]);
        exit(1);
    }

    char pattern[BUFFER_SIZE];
    strncpy(pattern, argv[1], BUFFER_SIZE);

    char buf[BUFFER_SIZE];

    FILE *fp;

    fp = fopen(argv[2], "r");

    if (fp == NULL)
    {
        printf("File not found\n");
        exit(1);
    }

    while (fgets(buf, BUFFER_SIZE, fp) != NULL)
    {
        if (strstr(buf, pattern) != NULL)
            printf("%s", buf);
    }

    fclose(fp);

    return 0;
}
```

Output

```
os-lab> cat a.txt
Graph Theory
Operating Systems
Computer Architecture and Organization
DBMS
Constitution of India
Professional Ethics
os-lab> .\a.out and a.txt
Computer Architecture and Organization
```

Program

```
echo "Enter the value of n"
read n

a=0
b=1

echo "$a"
echo "$b"

for ((i=2; i<n; i++))
do
    c=$((a + b))
    echo "$c"

    a=$b
    b=$c
done
```

Output

```
Enter the value of n
10
0
1
1
2
3
5
8
13
21
34
```

Program

```
echo "Enter a number: "
read n

for((i=2; i<=n/2; i++))
do
    if [ $((n%i)) -eq 0 ]
    then
        echo "$n is not a prime number."
        exit
    fi
done

echo "$n is a prime number."
```

Output

```
Enter a number:
5
5 is a prime number.

Enter a number:
6
6 is not a prime number.
```

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    int r = fork();

    if (r == -1)
    {
        printf("Error in process\n");
        exit(1);
    }

    else if (r == 0)
    {
        int pid = getpid();
        printf("Successfully forked process\n");
        printf("PID: %d\n", pid);
    }

    printf("Program to demonstrate fork()\n");

    return 0;
}
```

Output

```
Program to demonstrate fork()
Successfully forked process
PID: 750
Program to demonstrate fork()
```

Program

```
#include <stdio.h>

typedef struct
{
    int pid;
    int at;
    int bt;
    int ct;
    int wt;
    int tat;
} process;

int main()
{
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    process a[n];

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++)
    {
        a[i].pid = i + 1;
        scanf("%d %d", &a[i].at, &a[i].bt);
    }

    int completed = 0;
    int twt = 0, ttat = 0, current_time = 0;

    for (int i = 0; i < n; i++)
    {
        a[i].wt = 0;
        a[i].ct = 0;
    }

    // Sort the processes based on arrival time (FCFS)
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j].at > a[j + 1].at)
            {
                process temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        if (current_time < a[i].at)
```

```
            current_time = a[i].at;

        a[i].ct = current_time + a[i].bt;
        a[i].tat = a[i].ct - a[i].at;
        a[i].wt = a[i].tat - a[i].bt;

        twt += a[i].wt;
        ttat += a[i].tat;

        current_time = a[i].ct;
    }

    int avg_tat = ttat / n;
    int avg_wt = twt / n;

    printf("\nPID\tArrival\tBurst\tCompletion\tWait\tTurnaround\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\n", a[i].pid, a[i].at, a[i].bt,
a[i].ct, a[i].wt, a[i].tat);
    }

    printf("\nTotal Turnaround Time: %d\n", ttat);
    printf("Total Waiting Time: %d\n", twt);

    printf("Average Turnaround Time: %d\n", avg_tat);
    printf("Average Waiting Time: %d\n", avg_wt);

    return 0;
}
```

Output

```
Enter the number of processes: 6
Enter arrival time and burst time for each process:
0 8
1 4
2 2
3 1
4 3
5 2

PID     Arrival Burst   Completion      Wait    Turnaround
1       0       8       8               0       8
2       1       4       12              7       11
3       2       2       14              10      12
4       3       1       15              11      12
5       4       3       18              11      14
6       5       2       20              13      15

Total Turnaround Time: 72
Total Waiting Time: 52
Average Turnaround Time: 12
Average Waiting Time: 8
```

Program

```
#include <stdio.h>

typedef struct
{
    int pid;
    int p;
    int at;
    int bt;
    int ct;
    int rt;
    int wt;
    int tat;
} process;

int main()
{
    int n;
    printf("Enter the number of processes:");
    scanf("%d", &n);

    process a[n];

    printf("Enter arrival time, burst time and priority for each
process:\n");
    for (int i = 0; i < n; i++)
    {
        a[i].pid = i + 1;
        scanf("%d %d %d", &a[i].at, &a[i].bt, &a[i].p);
    }

    int completed = 0;
    int twt = 0, ttat = 0, current_time = 0;

    for (int i = 0; i < n; i++)
    {
        a[i].rt = a[i].bt;
        a[i].wt = 0;
    }

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j].p < a[j + 1].p)
            {
                process temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        if (current_time < a[i].at)
            current_time = a[i].at;

        a[i].ct = current_time + a[i].bt;
        a[i].tat = a[i].ct - a[i].at;
        a[i].wt = a[i].tat - a[i].bt;

        twt += a[i].wt;
        ttat += a[i].tat;

        current_time = a[i].ct;
    }

    int avg_tat = ttat / n;
    int avg_wt = twt / n;

    printf("PID\tBurst\tArrrival\tPriority\tCompletion\tWait\tTurn
Aruund\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t\t%d\t\t%d\t\t%d\t%d\n", a[i].pid, a[i].bt,
a[i].at, a[i].p, a[i].ct, a[i].wt, a[i].tat);

    printf("Total Turn Around Time: %d\n", ttat);
    printf("Total Waiting Time: %d\n", twt);

    printf("Average Turn Around Time: %d\n", avg_tat);
    printf("Average Waiting Time: %d\n", avg_wt);

    return 0;
}
```

Output

```
Enter the number of processes:3
Enter arrival time, burst time and priority for each process:
0 3 8
2 4 2
3 1 1
PID     Burst   Arrival Priority Completion Wait    Turnaround
1       3       0       8        3          0       3
2       4       2       2        7          1       5
3       1       3       1        8          4       5
Total Turn Around Time: 13
Total Waiting Time: 5
Average Turn Around Time: 4
Average Waiting Time: 1
```

Program

```c
#include <stdio.h>

typedef struct
{
    int pid;
    int at;
    int bt;
    int ct;
    int rt;
    int wt;
    int tat;
} process;

int main()
{
    int n;
    printf("Enter the number of processes:");
    scanf("%d", &n);

    process a[n];

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++)
    {
        a[i].pid = i + 1;
        scanf("%d %d", &a[i].at, &a[i].bt);
    }

    int completed = 0;
    int twt = 0, ttat = 0, current_time = 0;

    for (int i = 0; i < n; i++)
    {
        a[i].rt = a[i].bt;
        a[i].wt = 0;
    }

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j].bt > a[j + 1].bt)
            {
                process temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        if (current_time < a[i].at)
```

```
            current_time = a[i].at;

        a[i].ct = current_time + a[i].bt;
        a[i].tat = a[i].ct - a[i].at;
        a[i].wt = a[i].tat - a[i].bt;

        twt += a[i].wt;
        ttat += a[i].tat;

        current_time = a[i].ct;
    }

    int avg_tat = ttat / n;
    int avg_wt = twt / n;

    printf("PID\tBurst\tArrrival\tCompletion\tWait\tTurn Aruund\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t\t%d\t\t%d\t%d\n", a[i].pid, a[i].bt, a[i].at,
a[i].ct, a[i].wt, a[i].tat);

    printf("Total Turn Around Time: %d\n", ttat);
    printf("Total Waiting Time: %d\n", twt);

    printf("Average Turn Around Time: %d\n", avg_tat);
    printf("Average Waiting Time: %d\n", avg_wt);

    return 0;
}
```

Output

```
Enter the number of processes:3
Enter arrival time and burst time for each process:
0 3
2 4
3 1
PID     Burst   Arrival Completion      Wait    Turnaround
3       1       3       4               0       1
1       3       0       7               4       7
2       4       2       11              5       9
Total Turn Around Time: 17
Total Waiting Time: 9
Average Turn Around Time: 5
Average Waiting Time: 3
```

Program

```
#include <stdio.h>

typedef struct
{
    int pid;
    int at;
    int bt;
    int ct;
    int rt;
    int wt;
    int tat;
} process;

int main()
{
    int n;
    printf("Enter the number of processes:");
    scanf("%d", &n);

    process a[n];

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++)
    {
        a[i].pid = i + 1;
        scanf("%d %d", &a[i].at, &a[i].bt);
    }

    int completed = 0;
    int twt = 0, ttat = 0, current_time = 0;

    for (int i = 0; i < n; i++)
    {
        a[i].rt = a[i].bt;
        a[i].wt = 0;
    }

    while (completed < n)
    {
        int shortest_job = -1;
        int shortest_time = 99999;

        for (int i = 0; i < n; i++)
        {
            if (a[i].at <= current_time && a[i].rt < shortest_time &&
a[i].rt > 0)
            {
                shortest_job = i;
                shortest_time = a[i].rt;
            }

            if (shortest_job == -1)
                current_time++;
            else
            {
                a[shortest_job].rt--;
                current_time++;
                if (a[shortest_job].rt == 0)
                {
                    completed++;
                    a[shortest_job].ct = current_time;
                    a[shortest_job].tat = a[shortest_job].ct -
a[shortest_job].at;
                    a[shortest_job].wt = a[shortest_job].tat -
a[shortest_job].bt;
                    twt += a[shortest_job].wt;
                    ttat += a[shortest_job].tat;
                }
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        a[i].wt = a[i].tat - a[i].bt;
        twt += a[i].wt;
        ttat += a[i].tat;
    }

    int avg_tat = ttat / n;
    int avg_wt = twt / n;

    printf("PID\tBurst\tArrrival\tCompletion\tWait\tTurn Around\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t\t%d\t\t%d\t%d\n", a[i].pid, a[i].bt, a[i].at,
a[i].ct, a[i].wt, a[i].tat);

    printf("Total Turn Around Time: %d\n", ttat);
    printf("Total Waiting Time: %d\n", twt);

    printf("Average Turn Around Time: %d\n", avg_tat);
    printf("Average Waiting Time: %d\n", avg_wt);

    return 0;
}
```

Outline

```
Enter the number of processes:3
Enter arrival time and burst time for each process:
0 3
2 4
3 1
PID     Burst   Arrival Completion      Wait    Turnaround
1       3       0       3               0       3
2       4       2       11              5       9
3       1       3       6               2       3
Total Turn Around Time: 30
Total Waiting Time: 14
Average Turn Around Time: 10
Average Waiting Time: 4
```

Program

```
#include <stdio.h>

typedef struct
{
    int pid;
    int at;
    int bt;
    int ct;
    int rt;
    int wt;
    int tat;
} process;

int main()
{
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    process a[n];

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++)
    {
        a[i].pid = i + 1;
        scanf("%d %d", &a[i].at, &a[i].bt);
        a[i].rt = a[i].bt;
    }

    int quantum;
    printf("Enter the time quantum: ");
    scanf("%d", &quantum);

    // Sort the processes based on arrival time (FCFS)
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j].at > a[j + 1].at)
            {
                process temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

    int completed = 0;
    int twt = 0, ttat = 0, current_time = 0;

    while (completed < n)
    {
        for (int i = 0; i < n; i++)
        {
            if (a[i].rt > 0)
            {
                if (a[i].rt <= quantum)
                {
                    current_time += a[i].rt;
                    a[i].rt = 0;
                    a[i].ct = current_time;
                    a[i].tat = a[i].ct - a[i].at;
                    a[i].wt = a[i].tat - a[i].bt;
                    completed++;
                    twt += a[i].wt;
                    ttat += a[i].tat;
                }
                else
                {
                    current_time += quantum;
                    a[i].rt -= quantum;
                }
            }
        }
    }

    int avg_tat = ttat / n;
    int avg_wt = twt / n;

    printf("\nPID\tArrival\tBurst\tCompletion\tWait\tTurnaround\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\n", a[i].pid, a[i].at, a[i].bt,
a[i].ct, a[i].wt, a[i].tat);
    }

    printf("\nTotal Turnaround Time: %d\n", ttat);
    printf("Total Waiting Time: %d\n", twt);

    printf("Average Turnaround Time: %d\n", avg_tat);
    printf("Average Waiting Time: %d\n", avg_wt);

    return 0;
}
```

Output

```
Enter the number of processes: 3
Enter arrival time and burst time for each process:
0 3
2 4
3 1
Enter the time quantum: 2

PID     Arrival Burst   Completion      Wait    Turnaround
1       0       3       6               3       6
2       2       4       8               2       6
3       3       1       5               1       2
```

```
Total Turnaround Time: 14
Total Waiting Time: 6
Average Turnaround Time: 4
Average Waiting Time: 2
```

Program

```
#include <stdio.h>

#define MAX_BLOCKS 100
#define MAX_PROCESS 100

typedef struct
{
    int size;
    int allocated;
} mem_block;

typedef struct
{
    int size;
    int block;
} process;

void first_fit(mem_block mem_blocks[], int m, process p[], int n)
{
    int i, j;

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            if (mem_blocks[j].allocated == 0 && mem_blocks[j].size >=
p[i].size)
            {
                mem_blocks[j].allocated = 1;
                p[i].block = j;
                break;
            }
        }
    }
}

void best_fit(mem_block mem_blocks[], int m, process p[], int n)
{
    int i, j, best_block;

    for (i = 0; i < n; i++)
    {
        best_block = -1;
        for (j = 0; j < m; j++)
        {
            if (mem_blocks[j].allocated == 0 && mem_blocks[j].size >=
p[i].size)
            {
                if (best_block == -1)
                    best_block = j;
                else if (mem_blocks[best_block].size > mem_blocks[j].size)
                    best_block = j;
            }
        }

        if (best_block != -1)
        {
            mem_blocks[best_block].allocated = 1;
            p[i].block = best_block;
        }
    }
}

void worst_fit(mem_block mem_blocks[], int m, process p[], int n)
{
    int i, j, worst_block;

    for (i = 0; i < n; i++)
    {
        worst_block = -1;
        for (j = 0; j < m; j++)
        {
            if (mem_blocks[j].allocated == 0 && mem_blocks[j].size >=
p[i].size)
            {
                if (worst_block == -1)
                    worst_block = j;
                else if (mem_blocks[j].size > mem_blocks[worst_block].size)
                    worst_block = j;
            }
        }

        if (worst_block != -1)
        {
            mem_blocks[worst_block].allocated = 1;
            p[i].block = worst_block;
        }
    }
}

void print_allocation(mem_block mem_blocks[], int m, process p[], int n)
{
    printf("\nProc\tProc Size\tBlock\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t", i + 1, p[i].size);
        if (p[i].block != -1)
            printf("%d\n", p[i].block + 1);
        else
            printf("Not Allocated\n");
    }
}

int main()
{
    int m, n;
    mem_block mem_blocks[MAX_BLOCKS];
    process p[MAX_PROCESS];
    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);

    printf("Enter the size of each block: ");
    for (int i = 0; i < m; i++)
    {
        scanf("%d", &mem_blocks[i].size);
        mem_blocks[i].allocated = 0;
    }

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the size of each process: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &p[i].size);
        p[i].block = -1;
    }

    printf("1. First Fit\n");
    printf("2. Best Fit\n");
    printf("3. Worst Fit\n");
    printf("Enter your choice: ");

    int choice;
    scanf("%d", &choice);

    switch (choice)
    {
    case 1:
        first_fit(mem_blocks, m, p, n);
        break;
    case 2:

        best_fit(mem_blocks, m, p, n);
        break;
    case 3:
        worst_fit(mem_blocks, m, p, n);
        break;
    default:
        printf("Invalid choice\n");
        return 0;
    }

    print_allocation(mem_blocks, m, p, n);

    return 0;
}
```

Output

```
First fit
Enter the number of memory blocks: 5
Enter the size of each block: 20 100 40 200 10
Enter the number of processes: 4
Enter the size of each process: 90 50 30 40
1. First Fit
2. Best Fit
3. Worst Fit
Enter your choice: 1

Proc    Proc Size       Block
1       90              2
2       50              4
3       30              3
4       40              Not Allocated

Best Fit
Enter the number of memory blocks: 5
Enter the size of each block: 20 100 40 200 10
Enter the number of processes: 4
Enter the size of each process: 90 50 30 40
1. First Fit
2. Best Fit
3. Worst Fit
Enter your choice: 2

Proc    Proc Size       Block
1       90              2
2       50              4
3       30              3
4       40              Not Allocated

Worst Fit
Enter the number of memory blocks: 5
Enter the size of each block: 20 100 40 200 10
Enter the number of processes: 4
Enter the size of each process: 90 50 30 40
1. First Fit
2. Best Fit
3. Worst Fit
Enter your choice: 3

Proc    Proc Size       Block
1       90              4
2       50              2
3       30              3
4       40              Not Allocated
```