EXPERIMENT 5

20CP209P – Design and Analysis of Algorithm Lab

Aim:

Implement interval scheduling algorithm. Given n events with their starting and ending times, find a schedule that includes as many events as possible. It is not possible to select an event partially. For example, consider the following example:

Code:

Interval-Scheduling:

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

typedef struct Process {
   int id;
   int start;
   int finish;
   int duration;
} Process;

int comp_fin(const void* a, const void* b);
int comp_st(const void* a, const void* b);
int comp_dur(const void* a, const void* b);
void earl_st(Process processes[], int n);
void sjf(Process processes[], int n);
void earl_fin(Process processes[], int n);

int main(void)
{
   Process processes1[] = {
      {1, 1, 4, 4 - 1},
      {2, 3, 5, 5 - 3},
      {3, 0, 6, 6 - 0},
      {4, 5, 7, 7 - 5},
      {5, 3, 9, 9 - 3},
      {6, 5, 9, 9 - 5},
      {7, 6, 10, 10 - 6},
      {8, 8, 11, 11 - 8},
      {9, 8, 12, 12 - 8},
      {10, 2, 14, 14 - 2}
   };

   int n = sizeof(processes1) / sizeof(processes1[0]);

   earl_fin(processes1, n);
   earl_st(processes1, n);
```

```
    sjf(processes1, n);
    printf("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

    // As per Cormen example
    Process processes2[] = {
        {1, 1, 4, 4 - 1},
        {2, 3, 5, 5 - 3},
        {3, 0, 6, 6 - 0},
        {4, 5, 7, 7 - 5},
        {5, 3, 9, 9 - 3},
        {6, 5, 9, 9 - 5},
        {7, 6, 10, 10 - 6},
        {8, 8, 11, 11 - 8},
        {9, 8, 12, 12 - 8},
        {10, 2, 14, 14 - 2},
        {11, 12, 16, 16 - 12}
    };

    int n2 = sizeof(processes2) / sizeof(processes2[0]);

    earl_fin(processes2, n2);
    earl_st(processes2, n2);
    sjf(processes2, n2);

    return 0;
}

// Greedy activity selection - cormen pg. 424 - pdf pg. 446
void earl_fin(Process processes[], int n)
{
    qsort(processes, n, sizeof(Process), comp_fin);

    printf("Selected processes -> Earliest Finish Time\n(printed instead of added in set)\n");
    printf("As per Cormen Greedy Approach\n");
    int last_fin_time = 0;
    for (int i = 0; i < n; i++)
    {
        if (processes[i].start >= last_fin_time)
        {
            printf("Process %d -> Start: %d, Finish: %d, Duration: %d\n", processes[i].id, processes[i].start,
processes[i].finish, processes[i].duration);
            last_fin_time = processes[i].finish;
        }
    }
    return;
}

void earl_st(Process processes[], int n)
{
    qsort(processes, n, sizeof(Process), comp_st);
```

```
    printf("Selected processes -> Earliest Start Time\n(printed instead of added in set)\n");
    printf("As per Cormen Greedy Approach\n");
    int last_fin_time = 0;
    for (int i = 0; i < n; i++)
    {
        if (processes[i].start >= last_fin_time)
        {
            printf("Process %d -> Start: %d, Finish: %d, Duration: %d\n", processes[i].id, processes[i].start,
processes[i].finish, processes[i].duration);
            last_fin_time = processes[i].finish;
        }
    }
    return;
}

void sjf(Process processes[], int n)
{
    qsort(processes, n, sizeof(Process), comp_dur);

    printf("Selected processes -> Shortest Job first\n(printed instead of added in set)\n");
    printf("As per Cormen Greedy Approach\n");
    int last_fin_time = 0;
    for (int i = 0; i < n; i++)
    {
        if (processes[i].start >= last_fin_time)
        {
            printf("Process %d -> Start: %d, Finish: %d, Duration: %d\n", processes[i].id, processes[i].start,
processes[i].finish, processes[i].duration);
            last_fin_time = processes[i].finish;
        }
    }
    return;
}

int comp_fin(const void* a, const void* b)
{
    return (((Process *)a)->finish - ((Process *)b)->finish);
}

int comp_st(const void* a, const void* b)
{
    return (((Process *)a)->start - ((Process *)b)->start);
}

int comp_dur(const void* a, const void* b)
{
    return (((Process *)a)->duration - ((Process *)b)->duration);
}
```

// Details for qsort function
//
https://www.w3schools.com/c/ref_stdlib_qsort.php#:~:text=The%20qsort()%20function%20sorts,h%3E%20header%20file.

Output:

```
PS B:\sem4\23bcp153_daa\lab5> ./intsched
Selected processes -> Earliest Finish Time
(printed instead of added in set)
As per Cormen Greedy Approach
Process 1 -> Start: 1, Finish: 4, Duration: 3
Process 4 -> Start: 5, Finish: 7, Duration: 2
Process 8 -> Start: 8, Finish: 11, Duration: 3
Selected processes -> Earliest Start Time
(printed instead of added in set)
As per Cormen Greedy Approach
Process 3 -> Start: 0, Finish: 6, Duration: 6
Process 7 -> Start: 6, Finish: 10, Duration: 4
Selected processes -> Shortest Job first
(printed instead of added in set)
As per Cormen Greedy Approach
Process 4 -> Start: 5, Finish: 7, Duration: 2
Process 8 -> Start: 8, Finish: 11, Duration: 3

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Selected processes -> Earliest Finish Time
(printed instead of added in set)
As per Cormen Greedy Approach
Process 1 -> Start: 1, Finish: 4, Duration: 3
Process 4 -> Start: 5, Finish: 7, Duration: 2
Process 8 -> Start: 8, Finish: 11, Duration: 3
Process 11 -> Start: 12, Finish: 16, Duration: 4
Selected processes -> Earliest Start Time
(printed instead of added in set)
As per Cormen Greedy Approach
Process 3 -> Start: 0, Finish: 6, Duration: 6
Process 7 -> Start: 6, Finish: 10, Duration: 4
Process 11 -> Start: 12, Finish: 16, Duration: 4
Selected processes -> Shortest Job first
(printed instead of added in set)
As per Cormen Greedy Approach
Process 2 -> Start: 3, Finish: 5, Duration: 2
Process 4 -> Start: 5, Finish: 7, Duration: 2
Process 8 -> Start: 8, Finish: 11, Duration: 3
Process 11 -> Start: 12, Finish: 16, Duration: 4
```

Code:

Interval-Partitioning:

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

typedef struct Process {
    int id;
    int start;
    int finish;
    int duration;
} Process;

typedef struct  Node {
    Process process;
    struct Node* next;
} Node;

typedef struct TT {
    Node** classes;
    int n;
    int filled;
} TT;

int comp_fin(const void* a, const void* b);
int comp_st(const void* a, const void* b);
int comp_dur(const void* a, const void* b);
TT* init_tt(int n);
void add_proc_to_class(Node** head, Process p);
void earl_st(Process processes[], int n, TT* mytt);
void earl_fin(Process processes[], int n, TT* mytt);
void sjf(Process processes[], int n, TT* mytt);
void print_tt(TT* mytt);

int main(void)
{
    Process processes1[] = {
        {1, 1, 2, 2 - 1},
        {2, 1, 3, 3 - 1},
        {3, 1, 4, 4 - 1},
        {4, 2, 4, 4 - 2},
        {5, 3, 5, 5 - 3},
        {6, 4, 6, 6 - 4},
        {7, 4, 6, 6 - 4},
        {8, 6 , 7, 7 - 6},
        {9, 6, 8, 8 - 6},
        {10, 6, 8, 8 - 6}
```

```
    };

    int n = sizeof(processes1) / sizeof(processes1[0]);
    TT* mytt;

    printf("Earliest Finish Time Partitioning:\n");
    mytt = init_tt(n);
    earl_fin(processes1, n, mytt);
    print_tt(mytt);
    free(mytt->classes);
    free(mytt);

    printf("\nEarliest Start Time Partitioning:\n");
    mytt = init_tt(n);
    earl_st(processes1, n, mytt);
    print_tt(mytt);
    free(mytt->classes);
    free(mytt);

    printf("\nShortest Job First Partitioning:\n");
    mytt = init_tt(n);
    sjf(processes1, n, mytt);
    print_tt(mytt);
    free(mytt->classes);
    free(mytt);

    return 0;
}

TT* init_tt(int n)
{
    TT* mytt = (TT *)malloc(sizeof(TT));
    mytt->classes = (Node**)malloc(sizeof(Node *) * n);
    mytt->n = n;
    mytt->filled = 0;
    for (int i = 0; i < n; i++)
    {
        mytt->classes[i] = NULL;
    }

    return mytt;
}

void add_proc_to_class(Node** head, Process p)
{
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->process = p;
    new_node->next = *head;
    *head = new_node;
    return;
```

```
}

int can_place_in_class(Node* head, Process p)
{
    Node* curr = head;
    while(curr)
    {
        if (p.start < curr->process.finish && p.finish > curr->process.start)
        {
            return 0;
        }
        curr = curr->next;
    }
    return 1;
}

void earl_st(Process processes[], int n, TT* mytt)
{
    qsort(processes, n, sizeof(Process), comp_st);
    for (int i = 0; i < n; i++)
    {
        int placed = 0;
        for (int j = 0; j < mytt->filled; j++)
        {
            if (can_place_in_class(mytt->classes[j], processes[i]))
            {
                add_proc_to_class(&mytt->classes[j], processes[i]);
                placed = 1;
                break;
            }
        }
        if(!placed)
        {
            add_proc_to_class(&mytt->classes[mytt->filled], processes[i]);
            mytt->filled++;
        }
    }
    return;
}

void earl_fin(Process processes[], int n, TT* mytt)
{
    qsort(processes, n, sizeof(Process), comp_fin);
    for (int i = 0; i < n; i++)
    {
        int placed = 0;
        for (int j = 0; j < mytt->filled; j++)
        {
            if (can_place_in_class(mytt->classes[j], processes[i]))
            {
```

```
                add_proc_to_class(&mytt->classes[j], processes[i]);
                placed = 1;
                break;
            }
        }
        if(!placed)
        {
            add_proc_to_class(&mytt->classes[mytt->filled], processes[i]);
            mytt->filled++;
        }
    }
    return;
}

void sjf(Process processes[], int n, TT* mytt)
{
    qsort(processes, n, sizeof(Process), comp_dur);
    for (int i = 0; i < n; i++)
    {
        int placed = 0;
        for (int j = 0; j < mytt->filled; j++)
        {
            if (can_place_in_class(mytt->classes[j], processes[i]))
            {
                add_proc_to_class(&mytt->classes[j], processes[i]);
                placed = 1;
                break;
            }
        }
        if(!placed)
        {
            add_proc_to_class(&mytt->classes[mytt->filled], processes[i]);
            mytt->filled++;
        }
    }
    return;
}

void print_tt(TT* mytt)
{
    for (int i = 0; i < mytt->filled; i++)
    {
        printf("Class no.: %d\n\t", i);
        Node* curr = mytt->classes[i];
        while (curr)
        {
            printf("P%d - (%d-%d)   ", curr->process.id, curr->process.start, curr->process.finish);
            curr = curr->next;
        }
        printf("\n");
```

```
    }
    printf("\nTotal number of classes used: %d\n", mytt->filled);
}

int comp_fin(const void* a, const void* b)
{
    return (((Process *)a)->finish - ((Process *)b)->finish);
}

int comp_st(const void* a, const void* b)
{
    return (((Process *)a)->start - ((Process *)b)->start);
}

int comp_dur(const void* a, const void* b)
{
    return (((Process *)a)->duration - ((Process *)b)->duration);
}

// can implement using the below strategy
// https://leetcode.com/problems/divide-intervals-into-minimum-number-of-groups/editorial/
```

Output:

```
PS B:\sem4\23bcp153_daa\lab5> gcc intpart.c -o intpart
PS B:\sem4\23bcp153_daa\lab5> ./intpart
Earliest Finish Time Partitioning:
Class no.: 0
        P8 - (6-7)   P6 - (4-6)   P4 - (2-4)   P1 - (1-2)
Class no.: 1
        P10 - (6-8)   P5 - (3-5)   P2 - (1-3)
Class no.: 2
        P9 - (6-8)   P7 - (4-6)   P3 - (1-4)

Total number of classes used: 3

Earliest Start Time Partitioning:
Class no.: 0
        P10 - (6-8)   P5 - (3-5)   P2 - (1-3)
Class no.: 1
        P9 - (6-8)   P6 - (4-6)   P3 - (1-4)
Class no.: 2
        P8 - (6-7)   P7 - (4-6)   P4 - (2-4)   P1 - (1-2)

Total number of classes used: 3

Shortest Job First Partitioning:
Class no.: 0
        P6 - (4-6)   P4 - (2-4)   P8 - (6-7)   P1 - (1-2)
Class no.: 1
        P2 - (1-3)   P5 - (3-5)   P9 - (6-8)
Class no.: 2
        P3 - (1-4)   P10 - (6-8)   P7 - (4-6)

Total number of classes used: 3
PS B:\sem4\23bcp153_daa\lab5>
```