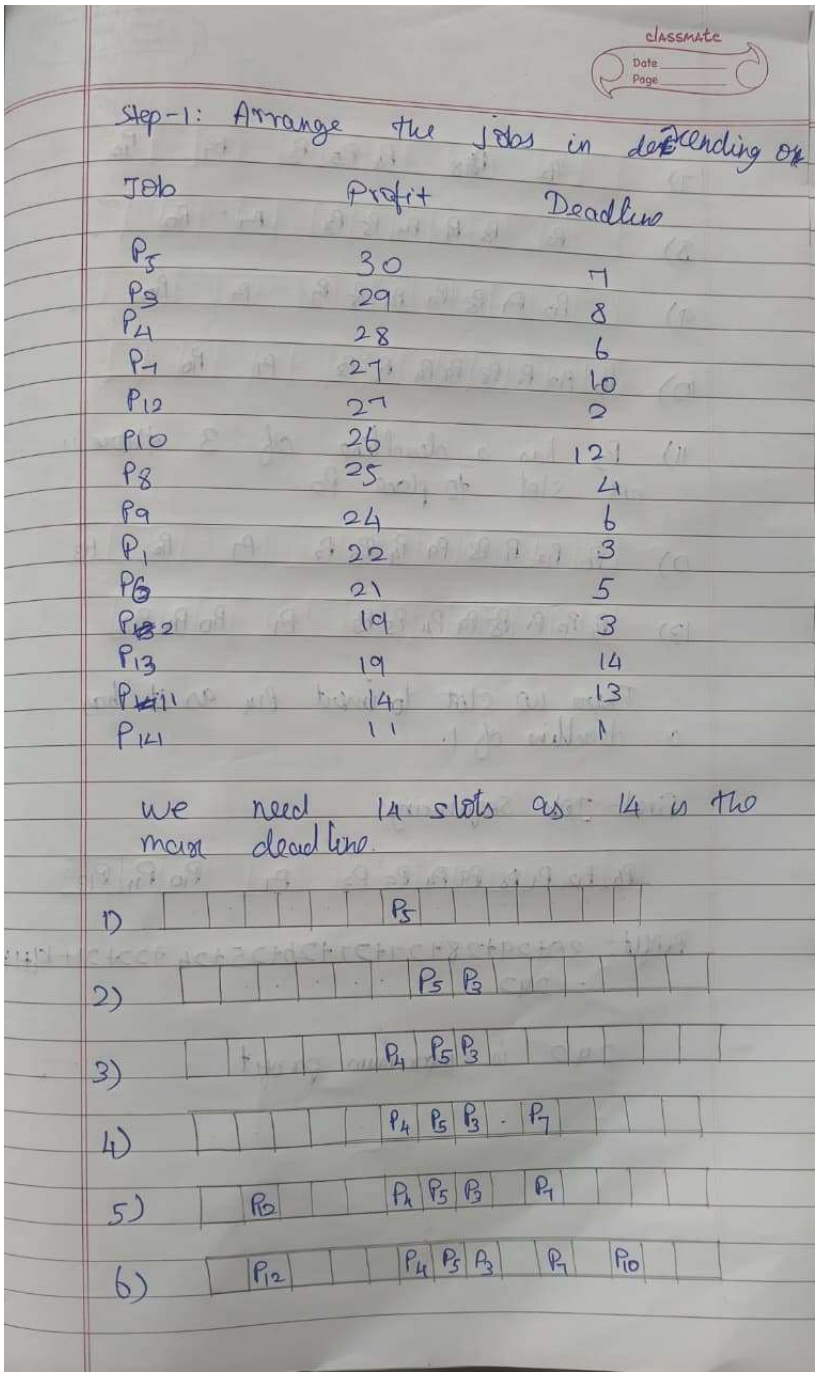
Design Analysis and Algorithm – Lab Work

Week 7

Write a program to implement job sequencing.

METHOD:



Step-1: Arrange the jobs in descending order

Job	Profit	Deadline
P_5	30	7
P_3	29	8
P_4	28	6
P_7	27	10
P_{12}	27	2
P_{10}	26	12
P_8	25	4
P_9	24	6
P_1	22	3
P_6	21	5
$P_{\cancel{13}2}$	19	3
P_{13}	19	14
$P_{\cancel{11}}$	14	13
P_{14}	11	1

we need 14 slots as 14 is the max deadline

1) P_5

2) P_5 P_3

3) P_4 P_5 P_3

4) P_4 P_5 P_3 P_7

5) P_{12} P_4 P_5 P_3 P_7

6) P_{12} P_4 P_5 P_3 P_7 P_{10}

7) | P_{12} | P_8 | P_4 | P_5 | P_3 | P_7 | P_{10} |

8) | P_{12} | P_8 | P_9 | P_4 | P_5 | P_3 | P_7 | P_{10} |

9) | P_{12} | P_1 | P_8 | P_9 | P_4 | P_5 | P_3 | P_7 | P_{10} |

10) | P_6 | P_{12} | P_1 | P_8 | P_9 | P_4 | P_5 | P_3 | P_7 | P_{10} |

11) P_2 has a deadline of 3 there is not slot to place P_2.

12) | P_6 | P_{12} | P_1 | P_8 | P_9 | P_4 | P_5 | P_3 | P_7 | P_{10} | P_{13} |

13) | P_6 | P_{12} | P_1 | P_8 | P_9 | P_4 | P_5 | P_3 | P_7 | P_{10} | P_{11} | P_{13} |

There no slot to insert P_{14} as it has a deadline of 1.

final Job sequencing:

| P_6 | P_{12} | P_1 | P_8 | P_9 | P_4 | P_5 | P_3 | P_7 | P_{10} | P_{11} | P_{13} |

Profit = 30+29+28+27+27+26+25+24+22+21+6(H.
= 292

292 is maximum profit.

CODE:

```c
#include <stdio.h>
#define MAX 50

struct Job{
    char id[5];
    int deadline;
    int profit;
};

void sortJobs(struct Job jobs[], int n)
{
    int i, j;
    struct Job temp;

    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (jobs[i].profit < jobs[j].profit)
            {
                temp = jobs[i];
                jobs[i] = jobs[j];
                jobs[j] = temp;
            }
        }
    }
}

int main()
{
    struct Job jobs[MAX];
    int n, i, j, maxD = 0;
    int slot[MAX];
    int totalProfit = 0;

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

    for (i = 0; i < n; i++)
    {
        printf("Enter Job id, Profit and Deadline: ");
        scanf("%s %d %d", jobs[i].id,
                          &jobs[i].profit,
                          &jobs[i].deadline);

        if (jobs[i].deadline > maxD)
```

```
            maxD = jobs[i].deadline;
    }

    sortJobs(jobs, n);
    for (i = 0; i <= maxD; i++)
        slot[i] = -1;

    for (i = 0; i < n; i++)
    {
        for (j = jobs[i].deadline; j > 0; j--)
        {
            if (slot[j] == -1)
            {
                slot[j] = i;
                totalProfit += jobs[i].profit;
                break;
            }
        }
    }

    printf("\nJob Sequence:\n");
    for (i = 1; i <= maxD; i++)
    {
        if (slot[i] != -1)
            printf("%s ", jobs[slot[i]].id);
    }

    printf("\nTotal Profit= %d\n", totalProfit);

    return 0;
}
```

OUTPUT:

```
PS C:\Sem-4\DAA\week-7> ./a
Enter number of jobs: 14
Enter Job id, Profit and Deadline: P1 22 3
Enter Job id, Profit and Deadline: P2 19 3
Enter Job id, Profit and Deadline: P3 29 8
Enter Job id, Profit and Deadline: P4 28 6
Enter Job id, Profit and Deadline: P5 30 7
Enter Job id, Profit and Deadline: P6 21 5
Enter Job id, Profit and Deadline: P7 27 10
Enter Job id, Profit and Deadline: P8 25 4
Enter Job id, Profit and Deadline: P9 24 6
Enter Job id, Profit and Deadline: P10 26 12
Enter Job id, Profit and Deadline: P11 14 13
Enter Job id, Profit and Deadline: P12 27 2
Enter Job id, Profit and Deadline: P13 19 14
Enter Job id, Profit and Deadline: P14 11 1

Job Sequence:
P6 P12 P1 P8 P9 P4 P5 P3 P7 P10 P11 P13
Total Profit= 292
```

Time Complexity:

The Job Sequencing algorithm first sorts the jobs in decreasing order of profit and then assigns each job to the latest available slot before its deadline. The sorting step takes $O(n^2)$time when simple sorting methods are used. During scheduling, each job may require checking multiple slots up to its deadline, which in the worst case also takes $O(n^2)$time. Hence, the overall time complexity of the algorithm is $O(n^2)$.

Space Complexity:

The algorithm uses additional space to store the job list and the slot array used for scheduling. These require space proportional to the number of jobs, resulting in a space complexity of $O(n)$. The algorithm is non-recursive, so no extra stack space is required.