Project #2

COMP 491

April 6, 2016

David Sabzanov

Borhan Alizadeh

This project consists of testing three different lists of job tasks. Each list varies in size and only contain the length of each job, the deadline for each job, and the profit for each job. The goal of this project is to obtain the highest profit by scheduling each list of job tasks with five different methods. The first method is the brute force method. The brute force method goes through each and every combination of jobs and determines which set of scheduled jobs obtains the highest profit. Because this is a brute force method that looks through each and every combination of jobs, it will have a runtime of "n factorial." The second method is the first of three greedy methods. This method sorts each list by earliest deadline first and then comes up with a schedule and profit based off each newly sorted list. The third method is a greedy method that sorts each list to have the shortest job first and then comes up with a schedule and profit. The fourth method is the last greedy method. This method sorts each list by highest priority first and then creates the schedule with the highest priority. The final method is a custom method that we create to obtain the goal of this project. This method needs to have a runtime less than or equal to "n cubed." Since this is a custom method, we created an algorithm that schedules the list of jobs by first setting a start time for each job. The start time is determined by subtracting the length of each job from the deadline of each job. Then we use this "start" time to order the lists of jobs in increasing order so that the "earliest" start time is in the beginning.

Our first list (test A) includes four job tasks. When running the brute force method, we received a schedule that includes a profit of 36 and only includes jobs #3, #2 and #1 in the schedule. The earliest deadline first method resulted in a profit of only 19 and had scheduled jobs #0 and #1. The method that sorted by shortest job first had resulted in a profit of only 23 and jobs #2 and #1 were scheduled. The method where the highest priority comes first had resulted with a profit of 36 and had scheduled jobs #2, #3 and #1. Our custom method had scheduled jobs 0 and 1 and had received a profit of 19. From this data we can see that the greedy method where this list is sorted with the highest priority first had a profit of exactly the same as the brute force method and had used the same jobs but in a slightly different order. This shows that the third greedy method had found the optimal solution and best of all it found it using a much faster algorithm.

The next list to test out (test B) consists of ten jobs. Looking at the results from running the earliest deadline first algorithm, we can see that the profit received was 43 for the scheduled jobs: #5, #2, #0 #1, #7, and #6. The shortest job first algorithm resulted in a profit of 29 from jobs: #2, #0, #6, #1, and #7. The highest profit first algorithm resulted in a schedule of #3, #7, and #6. Its profit came out to be 37. The custom algorithm had outputted a schedule of #8, #2, #0, #7, and #6 and a profit of 27. While the highest profit first algorithm had impressively received a profit of 37 with only 3 jobs scheduled, it is still not an optimal solution which can be seen from the earliest deadline algorithm that resulted in a profit of 43 from 6 jobs.

The last list that was tested (test C) consisted of 25 jobs. Since the number of jobs is too large, we are not able to run the brute force method as it is very time consuming and computer resource heavy. But we were able to test this case with the rest of the algorithms. Results from the earliest deadline algorithm show that with a schedule of #19, #5, #0, #2, #21, #20, #10, #24, #11, #9, #13, and #12, the profit resulted in a total of 83. The shortest job first algorithm resulted in a profit of 66 from a schedule of jobs: #2, #0, #21, #11, #9, #1, #8, #12, #10, #24, #13, and #23. The highest profit first algorithm had came up with a schedule of #3, #2, #17, #10, #13, #11, and #12 with a grand profit of 83. Lastly our custom algorithm has came up with a schedule of #22, #1, #8, #6, #10, #11, #9, #13, and #12 with a profit of 58. From the data of this test case, we can see that the highest profit first had less jobs scheduled but the same profit as the earliest deadline first schedule. This shows that in this case, the earliest deadline first was able to get more jobs scheduled for the same profit as the highest profit first algorithm which essentially says that the earliest deadline first algorithm is more optimal.

We can see that from all tests we have done, results show that for lists of jobs as large as ten or greater result in a better schedule when using the earliest deadline first algorithm which can be seen in test cases B and C where earliest deadline first receives profits 43 and 83 respectively while the highest priority first receives 37 and 83 respectively. For sizes as small as four, the highest profit algorithm is more optimal, resulting in a higher profit and more jobs scheduled. Out of all three greedy methods, the shortest job first algorithm did not provide an optimal solution in all test cases. But

they were all able to run efficiently. While our custom algorithm was quick in achieving a result, unfortunately based off of the results 19, 27, and 58, it does not produce an optimal result in any of the test cases since its profit was the lowest in all three cases. While the brute force algorithm can retrieve the most optimal solution, it does not achieve this faster than any of the other algorithms. Therefore it may not be viable in real world scenarios were large amounts of data need to be looked through to find an optimal solution. Unfortunately, for Test B, the brute force algorithm would run without displaying any result as if it were stuck in an infinite loop. But since it was able to output a result for Test A, then the issue might be that the amount of jobs in Test B might be too large for the brute force algorithm to handle. Overall, it seems that the most efficient and optimal algorithm to solve this scheduling problem would be the earliest deadline first algorithm for job sets of about ten or more.

run:
TestCase A
Jobs to be scheduled
Job format is (length, deadline, profit, start, finish)
#0:(7,7,10,-1,-1)
#1:(4,16,9,-1,-1)
#2:(2,8,14,-1,-1)
#3:(5,10,13,-1,-1)

Optimal Solution Using Brute Force O(n!)

Schedule Profit = 36
#3:(5,10,13,0,5)
#2:(2,8,14,5,7)
#1:(4,16,9,7,11)
#0:(7,7,10,11,18)
#0:(7,7,10,11,18)

EDF with unprofitable jobs last
Schedule Profit = 19
#0:(7,7,10,0,7)
#1:(4,16,9,7,11)
#2:(2,8,14,11,13)
#3:(5,10,13,13,18)

SJF with unprofitable jobs last
Schedule Profit = 23
#2:(2,8,14,0,2)
#1:(4,16,9,2,6)
#3:(5,10,13,6,11)
#0:(7,7,10,11,18)

HPF with unprofitable jobs last
Schedule Profit = 36
#2:(2,8,14,0,2)
#3:(5,10,13,2,7)
#1:(4,16,9,7,11)
#0:(7,7,10,11,18)

Your own creative solution
Schedule Profit = 19
#0:(7,7,10,0,7)
#1:(4,16,9,7,11)
#3:(5,10,13,11,16)
#2:(2,8,14,16,18)
BUILD SUCCESSFUL (total time: 0 seconds)

run:
TestCase B
Jobs to be scheduled
Job format is (length, deadline, profit, start, finish)
#0:(2,10,2,-1,-1)
#1:(3,12,5,-1,-1)
#2:(1,9,13,-1,-1)
#3:(10,22,28,-1,-1)
#4:(7,10,9,-1,-1)
#5:(4,4,14,-1,-1)
#6:(2,18,2,-1,-1)
#7:(5,15,7,-1,-1)
#8:(7,5,3,-1,-1)
#9:(7,9,10,-1,-1)

EDF with unprofitable jobs last
Schedule Profit = 43
#5:(4,4,14,0,4)
#2:(1,9,13,4,5)
#0:(2,10,2,5,7)
#1:(3,12,5,7,10)
#7:(5,15,7,10,15)
#6:(2,18,2,15,17)
#8:(7,5,3,17,24)
#9:(7,9,10,24,31)
#4:(7,10,9,31,38)
#3:(10,22,28,38,48)

SJF with unprofitable jobs last
Schedule Profit = 29
#2:(1,9,13,0,1)
#0:(2,10,2,1,3)
#6:(2,18,2,3,5)
#1:(3,12,5,5,8)
#7:(5,15,7,8,13)
#5:(4,4,14,13,17)
#8:(7,5,3,17,24)
#9:(7,9,10,24,31)
#4:(7,10,9,31,38)
#3:(10,22,28,38,48)

HPF with unprofitable jobs last
Schedule Profit = 37
#3:(10,22,28,0,10)
#7:(5,15,7,10,15)
#6:(2,18,2,15,17)

#5:(4,4,14,17,21)
#2:(1,9,13,21,22)
#9:(7,9,10,22,29)
#4:(7,10,9,29,36)
#1:(3,12,5,36,39)
#8:(7,5,3,39,46)
#0:(2,10,2,46,48)

Your own creative solution
Schedule Profit = 27
#8:(7,5,3,0,7)
#2:(1,9,13,7,8)
#0:(2,10,2,8,10)
#7:(5,15,7,10,15)
#6:(2,18,2,15,17)
#5:(4,4,14,17,21)
#9:(7,9,10,21,28)
#4:(7,10,9,28,35)
#1:(3,12,5,35,38)
#3:(10,22,28,38,48)
BUILD SUCCESSFUL (total time: 0 seconds)

run:
TestCase C
Jobs to be scheduled
Job format is (length, deadline, profit, start, finish)
#0:(2,10,2,-1,-1)
#1:(3,12,5,-1,-1)
#2:(1,15,13,-1,-1)
#3:(10,8,28,-1,-1)
#4:(7,10,8,-1,-1)
#5:(4,9,7,-1,-1)
#6:(6,22,6,-1,-1)
#7:(9,12,5,-1,-1)
#8:(3,15,3,-1,-1)
#9:(2,35,4,-1,-1)
#10:(5,29,9,-1,-1)
#11:(2,32,7,-1,-1)
#12:(5,45,6,-1,-1)
#13:(7,41,9,-1,-1)
#14:(7,13,14,-1,-1)
#15:(6,16,2,-1,-1)
#16:(3,10,7,-1,-1)
#17:(7,20,11,-1,-1)
#18:(8,10,3,-1,-1)
#19:(4,4,10,-1,-1)
#20:(5,18,8,-1,-1)
#21:(2,15,5,-1,-1)
#22:(9,5,9,-1,-1)
#23:(10,9,10,-1,-1)
#24:(5,30,3,-1,-1)

EDF with unprofitable jobs last
Schedule Profit = 83
#19:(4,4,10,0,4)
#5:(4,9,7,4,8)
#0:(2,10,2,8,10)
#2:(1,15,13,10,11)
#21:(2,15,5,11,13)
#20:(5,18,8,13,18)
#10:(5,29,9,18,23)
#24:(5,30,3,23,28)
#11:(2,32,7,28,30)
#9:(2,35,4,30,32)
#13:(7,41,9,32,39)
#12:(5,45,6,39,44)
#22:(9,5,9,44,53)

#3:(10,8,28,53,63)
#23:(10,9,10,63,73)
#16:(3,10,7,73,76)
#18:(8,10,3,76,84)
#4:(7,10,8,84,91)
#7:(9,12,5,91,100)
#1:(3,12,5,100,103)
#14:(7,13,14,103,110)
#8:(3,15,3,110,113)
#15:(6,16,2,113,119)
#17:(7,20,11,119,126)
#6:(6,22,6,126,132)

SJF with unprofitable jobs last
Schedule Profit = 66
#2:(1,15,13,0,1)
#0:(2,10,2,1,3)
#21:(2,15,5,3,5)
#11:(2,32,7,5,7)
#9:(2,35,4,7,9)
#1:(3,12,5,9,12)
#8:(3,15,3,12,15)
#12:(5,45,6,15,20)
#10:(5,29,9,20,25)
#24:(5,30,3,25,30)
#13:(7,41,9,30,37)
#16:(3,10,7,37,40)
#19:(4,4,10,40,44)
#5:(4,9,7,44,48)
#20:(5,18,8,48,53)
#15:(6,16,2,53,59)
#6:(6,22,6,59,65)
#4:(7,10,8,65,72)
#14:(7,13,14,72,79)
#17:(7,20,11,79,86)
#18:(8,10,3,86,94)
#7:(9,12,5,94,103)
#22:(9,5,9,103,112)
#3:(10,8,28,112,122)
#23:(10,9,10,122,132)

HPF with unprofitable jobs last
Schedule Profit = 83
#3:(10,8,28,0,10)
#2:(1,15,13,10,11)
#17:(7,20,11,11,18)

#10:(5,29,9,18,23)
#13:(7,41,9,23,30)
#11:(2,32,7,30,32)
#12:(5,45,6,32,37)
#14:(7,13,14,37,44)
#23:(10,9,10,44,54)
#19:(4,4,10,54,58)
#22:(9,5,9,58,67)
#4:(7,10,8,67,74)
#20:(5,18,8,74,79)
#5:(4,9,7,79,83)
#16:(3,10,7,83,86)
#6:(6,22,6,86,92)
#7:(9,12,5,92,101)
#1:(3,12,5,101,104)
#21:(2,15,5,104,106)
#9:(2,35,4,106,108)
#18:(8,10,3,108,116)
#24:(5,30,3,116,121)
#8:(3,15,3,121,124)
#15:(6,16,2,124,130)
#0:(2,10,2,130,132)

Your own creative solution
Schedule Profit = 58
#22:(9,5,9,0,9)
#1:(3,12,5,9,12)
#8:(3,15,3,12,15)
#6:(6,22,6,15,21)
#10:(5,29,9,21,26)
#11:(2,32,7,26,28)
#9:(2,35,4,28,30)
#13:(7,41,9,30,37)
#12:(5,45,6,37,42)
#3:(10,8,28,42,52)
#23:(10,9,10,52,62)
#19:(4,4,10,62,66)
#18:(8,10,3,66,74)
#4:(7,10,8,74,81)
#7:(9,12,5,81,90)
#5:(4,9,7,90,94)
#14:(7,13,14,94,101)
#16:(3,10,7,101,104)
#0:(2,10,2,104,106)
#15:(6,16,2,106,112)
#21:(2,15,5,112,114)

#17:(7,20,11,114,121)
#20:(5,18,8,121,126)
#2:(1,15,13,126,127)
#24:(5,30,3,127,132)
BUILD SUCCESSFUL (total time: 0 seconds)

```
/*
Project #2
COMP 496
David Sabzanov
Borhan Alizadeh
April 6, 2017
*/

package jobscheduler;


import java.util.ArrayList;

public class JobScheduler
{
	private int nJobs;
	private Job[]  jobs;

	int length = 0;
	String[] bf;
	private int pcount = 0;



	//adds the jobs into jobs array
	public JobScheduler( int[] joblength, int[] deadline, int[] profit)
	{
		//Set nJobs
		//Fill jobs array. The kth job entered has JobNo = k;
		nJobs = joblength.length;
		jobs = new Job[nJobs];
		for (int i = 0; i < joblength.length; i++) {
			Job jb = new Job(i, joblength[i], deadline[i], profit[i]);
			jobs[i] = jb;
		}
	    printJobs();
	}


       //helper method to call permutation
	  public void permutation(String str) {
		permutation("", str);
	  }

	  //gives us all the combination of mutiple digits
	  private void permutation(String prefix, String str) {
```

```
		int n = str.length();

		if (n == 0){
			bf[pcount] = prefix;
			pcount++;
		}
		else {
			for (int i = 0; i < n; i++)
				permutation(prefix + str.charAt(i), str.substring(0, i) +
str.substring(i+1, n));
		}
	}

	//gives us the factorial result of a digit
	public static long factorial(long number) {
		if (number <= 1) // test for base case
			return 1; // base cases: 0! = 1 and 1! = 1
		else
			// recursion step
			return number * factorial(number - 1);
	}



	//Brute force. Try all n! orderings. Return the schedule with the most profit
	public Schedule bruteForceSolution()
	{
		Schedule Sc = new Schedule();

		int maxProfit = 0;
		Job[] finalJobs = new Job[jobs.length];

		length = (int) factorial(nJobs);
		bf = new String[length];

		String perm = "";
		//automatically makes the permutation number
		for (int m = 0; m < nJobs; m++) {
			perm = perm + m;
		}

		int firstPermutationDigit = 0;
		permutation(perm);
		int i;
```

```
                System.out.println("----------------");


                for(int kk=0; kk < bf.length;kk++){
                        //we get each permutation here
                        maxProfit = 0;
                        int[] temp = new int[jobs.length];

                        for(i=0; i < (int) Math.log10(Integer.parseInt(bf[0])) + 1;i++){
                                firstPermutationDigit = Integer.parseInt((bf[kk]).substring(i, i
+1));
                                //we get each digits of each permutation one by one


                                for(int ii=0;ii<jobs.length;ii++){
                                        temp[i] = firstPermutationDigit;
                                }
                                Job tempJob;

                                //We use this for-loop to move the jobs according to sorted
list
                                for(int j=0; j < temp.length; j++){
                                        for(int k=j; k < jobs.length; k++){
                                                if(jobs[k].jobNumber == temp[j]){
                                                        tempJob = jobs[j];
                                                        jobs[j] = jobs[k];
                                                        jobs[k] = tempJob;
                                                        break;
                                                }
                                        }
                                }
                        }



                        jobs[0].start = 0;
                        jobs[0].finish = jobs[0].length;
                        maxProfit = jobs[0].profit;

                        int pointer = 1;
                        int count = 1;

                        while(pointer < jobs.length){

                                //this if statement is for when a job is acceptable and we can
add its benefit
```

```
                        if((jobs[count-1].finish+jobs[count].length) <=
jobs[count].deadline){

                                jobs[count].start = jobs[count-1].finish;
                                jobs[count].finish = jobs[count].start +
jobs[count].length;

                                maxProfit = maxProfit + jobs[count].profit;
                                Sc.add(jobs[count]);
                                count++;
                                pointer++;
                        }
                        else {
                                Job tempJ;
                                tempJ = jobs[count];
                                for(int k=count+1; k < jobs.length; k++){
                                        jobs[k-1] = jobs[k];
                                }
                                jobs[jobs.length-1] = tempJ;
                                pointer++;
                        }
                }


                if(maxProfit >= Sc.getProfit()){
                        Sc.schedule.clear();
                        Sc.setBFProfit(maxProfit);


                        for(int m=0; m < finalJobs.length; m++){
                                finalJobs[m] = jobs[m];
                                Sc.add(finalJobs[m]);
                        }

                }


                for (int q = count; q < temp.length; q++) {
                        Sc.add(jobs[q]);
                }

                while(count < jobs.length){

                        jobs[count].start = jobs[count-1].finish;
                        jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                        count++;
```

```
                }
        }

        return Sc;

}


public Schedule makeScheduleEDF()
//earliest deadline first schedule. Schedule items contributing 0 to total profit last
{

        Schedule Sc = new Schedule();

        int[] temp = new int[jobs.length];
        for(int i=0;i<jobs.length;i++){
                temp[i] = jobs[i].deadline;
        }
        insertionsort(temp);

        Job tempJob;
        for(int j=0; j < temp.length; j++){
                for(int k=j; k < jobs.length; k++){
                        if(jobs[k].deadline == temp[j]){
                                tempJob = jobs[j];
                                jobs[j] = jobs[k];
                                jobs[k] = tempJob;
                                break;
                        }
                }
        }

        jobs[0].start = 0;
        jobs[0].finish = jobs[0].length;
        Sc.add(jobs[0]);
        Sc.setProfit(jobs[0].profit);

        int pointer = 1;
        int count = 1;
        while(pointer < jobs.length){
                if((jobs[count-1].finish+jobs[count].length) <= jobs[count].deadline){

                        jobs[count].start = jobs[count-1].finish;
                        jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                        Sc.setProfit(jobs[count].profit);
```

```
                        //                      System.out.println("Profit is:
"+Sc.getProfit());
                        Sc.add(jobs[count]);
                        count++;
                        pointer++;
                    }
                    else {
                        Job tempJ;
                        tempJ = jobs[count];
                        for(int k=count+1; k < temp.length; k++){
                            jobs[k-1] = jobs[k];
                        }
                        jobs[temp.length-1] = tempJ;
                        pointer++;
                    }
                }
                for (int q = count; q < temp.length; q++) {
                    Sc.add(jobs[q]);
                }

                while(count < jobs.length){

                    jobs[count].start = jobs[count-1].finish;
                    jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                    count++;
                }

                return Sc;
            }


            public Schedule makeScheduleSJF()
            //shortest job first schedule. Schedule items contributing 0 to total profit last
            {
                Schedule Sc = new Schedule();

                int[] temp = new int[jobs.length];
                for(int i=0;i<jobs.length;i++){
                    temp[i] = jobs[i].length;
                }
                insertionsort(temp);

                Job tempJob;
                for(int j=0; j < temp.length; j++){
                    for(int k=j; k < jobs.length; k++){
                        if(jobs[k].length == temp[j]){
```

```
                        tempJob = jobs[j];
                        jobs[j] = jobs[k];
                        jobs[k] = tempJob;
                        break;
                }
        }
}


jobs[0].start = 0;
jobs[0].finish = jobs[0].length;
Sc.add(jobs[0]);
Sc.setProfit(jobs[0].profit);

int pointer = 1;
int count = 1;
while(pointer < jobs.length){
        if((jobs[count-1].finish+jobs[count].length) <= jobs[count].deadline){

                jobs[count].start = jobs[count-1].finish;
                jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                Sc.setProfit(jobs[count].profit);
                Sc.add(jobs[count]);
                count++;
                pointer++;
        }
        else {
                Job tempJ;
                tempJ = jobs[count];
                for(int k=count+1; k < temp.length; k++){
                        jobs[k-1] = jobs[k];
                }
                jobs[temp.length-1] = tempJ;
                pointer++;
        }
}
for (int q = count; q < temp.length; q++) {
        Sc.add(jobs[q]);
}


while(count < jobs.length){

        jobs[count].start = jobs[count-1].finish;
        jobs[count].finish = jobs[count-1].finish + jobs[count].length;
        count++;
```

```
        }

        return Sc;
}

public Schedule makeScheduleHPF()
//highest profit first schedule. Schedule items contributing 0 to total profit last
{
        Schedule Sc = new Schedule();

        int[] temp = new int[jobs.length];
        for(int i=0;i<jobs.length;i++){
                temp[i] = jobs[i].profit;
        }

        insertionsort(temp);

        Job tempJob;
        for(int j=0; j < temp.length; j++){
                for(int k=j; k < jobs.length; k++){
                        if(jobs[k].profit == temp[j]){
                                tempJob = jobs[j];
                                jobs[j] = jobs[k];
                                jobs[k] = tempJob;
                                break;
                        }
                }
        }

        for(int i = 0; i < jobs.length / 2; i++)
        {
                Job tempVal = jobs[i];
                jobs[i] = jobs[jobs.length - i - 1];
                jobs[jobs.length - i - 1] = tempVal;
        }


        jobs[0].start = 0;
        jobs[0].finish = jobs[0].length;
        Sc.setProfit(jobs[0].profit);
        Sc.add(jobs[0]);
        int count = 1;
        int pointer = 1;

        while(pointer< jobs.length){
```

```
                if((jobs[count-1].finish+jobs[count].length) <= jobs[count].deadline){


                        jobs[count].start = jobs[count-1].finish;
                        jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                        Sc.setProfit(jobs[count].profit);
                        Sc.add(jobs[count]);
                        count++;
                        pointer++;

                }
                else {

                        Job tempJ;
                        tempJ = jobs[count];
                        for(int k=count+1; k < temp.length; k++){
                                jobs[k-1] = jobs[k];
                        }
                        jobs[temp.length-1] = tempJ;
                        pointer++;
                }
        }


        for (int q = count; q < temp.length; q++) {
                Sc.add(jobs[q]);
        }


        while(count < jobs.length){

                jobs[count].start = jobs[count-1].finish;
                jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                count++;
        }

        return Sc;
}


public Schedule newApproxSchedule() //Your own creation. Must be <= O(n3)
{

        Schedule Sc = new Schedule();
        int[] temp = new int[jobs.length];
```

```
//deadline - length for each job
for(int j=0; j < jobs.length; j++){
	int m = (jobs[j].deadline) - (jobs[j].length);
	jobs[j].start = m;
}

//save the temp digit into start time temporary
for(int i=0;i<jobs.length;i++){
	temp[i] = jobs[i].start;
}

insertionsort(temp);

for(int aa=0; aa < temp.length; aa++){
}

Job tempJob;
for(int j=0; j < temp.length; j++){
	for(int k=j; k < jobs.length; k++){
		if(jobs[k].start == temp[j]){
			tempJob = jobs[j];
			jobs[j] = jobs[k];
			jobs[k] = tempJob;
			break;
		}
	}
}


for(int k=0; k < jobs.length; k++){
	jobs[k].start = -1;
}

for(int i=0;i<jobs.length;i++){
	if(i==0){
		jobs[0].start = 0;
		jobs[i].finish = jobs[i].length;

	}
	else{
		jobs[i].start = jobs[i-1].finish;
		jobs[i].finish = (jobs[i-1].finish+jobs[i].length);
	}
}
```

```
        jobs[0].start = 0;
        jobs[0].finish = jobs[0].length;
        Sc.add(jobs[0]);
        Sc.setProfit(jobs[0].profit);

        int pointer = 1;
        int count = 1;
        while(pointer < jobs.length){
                if((jobs[count-1].finish+jobs[count].length) <= jobs[count].deadline){


                        jobs[count].start = jobs[count-1].finish;
                        jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                        Sc.setProfit(jobs[count].profit);
                        Sc.add(jobs[count]);
                        count++;
                        pointer++;
                }
                else {
                        Job tempJ;
                        tempJ = jobs[count];
                        for(int k=count+1; k < temp.length; k++){
                                jobs[k-1] = jobs[k];
                        }
                        jobs[temp.length-1] = tempJ;
                        pointer++;
                }
        }
        for (int q = count; q < temp.length; q++) {
                Sc.add(jobs[q]);
        }

        while(count < jobs.length){

                jobs[count].start = jobs[count-1].finish;
                jobs[count].finish = jobs[count-1].finish + jobs[count].length;
                count++;
        }

        return Sc;
}

public static void insertionsort( int[] a)
{
        int t = 0;
```

```
                int n;
                for(int i = 1 ; i < a.length; i++)
                {
                        t = a[i];
                        n = i;
                        while((n >= 1) && (t < a[n - 1]))
                        {
                                a[n] = a[n - 1];
                                n--;
                        }
                        a[n] = t;
                }
        }

        public void printJobs()  //prints the array jobs
        {
                for (int i = 0; i < nJobs; i++) {
                        System.out.println(jobs[i].toString());
                }
        }



}//end of JobScheduler class




//--------------------------------------------------
class Job
{
        int jobNumber;
        int length;
        int deadline;
        int profit;
        int start;
        int finish;


        public Job( int jn , int len, int d, int p)
        {
                jobNumber = jn; length = len; deadline = d;
                profit = p;  start = -1;  finish = -1;
        }
```

```
	public String toString()
	{
		return "#" + jobNumber + ":(" + length + ","
				+ deadline + "," + profit +
				"," + start + "," + finish + ")";
	}


}//end of Job class



// --------------------------------------------------
class Schedule
{
	ArrayList<Job> schedule;
	int profit;

	public Schedule()
	{
		profit = 0;
		schedule = new ArrayList<Job>();
	}

	public void add(Job job)
	{
		schedule.add(job);
	}


	public int getProfit()
	{
		return profit;
	}

	public void setProfit(int p)
	{
		profit = profit + p;
	}

	public void setBFProfit(int p)
	{
		profit = p;
	}

	public String toString()
```

```
	{
		String s = "Schedule Profit = " + profit;
		for(int k = 0 ; k < schedule.size(); k++)
		{
			s = s + "\n"  + schedule.get(k);

		}

		return s;
	}


}// end of Schedule class
```

```
/*
Project #2
COMP 496
David Sabzanov
Borhan Alizadeh
April 6, 2017
*/

package jobscheduler;


//Comp 496ALG  Project 2  TestCase A with solutions
import java.util.*;

public class SchedulerDriver2
{

   public static void main (String[] args)
   {
       //TEST CASE A
//       int[] length = { 7,4,2,5};
//       int[] deadline = {7 ,16 ,8, 10};
//       int[] profit = { 10, 9, 14, 13};

       //TEST CASE B
//       int[] length = { 2,3,1,10,7,4,2,5,7,7};
//       int[] deadline = { 10,12, 9 ,22,  10, 4, 18, 15, 5, 9};
//       int[] profit = { 2,5,13,28,9,14, 2, 7, 3, 10};



       //TEST CASE C
       int[] length = { 2,3,1,10,7,  4,6,9,3,2,  5,2,5,7,7,  6,3,7,8,4,  5,2,9,10,5};
       int[] deadline = { 10,12,15,8,10,  9,22,12,15,35,  29,32,45,41,13,
                                  16,10,20,10,4,  18,15,5,9, 30};
       int[] profit = { 2,5,13,28,8, 7,6,5,3,4,  9,7,6,9,14,  2,7,11,3,10,
                              8,5,9,10,3 };




     System.out.println("TestCase C");
```

```
        System.out.println("Jobs to be scheduled");
        System.out.println("Job format is " +
        "(length, deadline, profit, start, finish)" );
       JobScheduler js = new JobScheduler( length,deadline, profit);



      //-----------------------------------------
//      System.out.println("\nOptimal Solution Using Brute Force O(n!)");
//      Schedule bestSchedule = js.bruteForceSolution();
//      System.out.println( bestSchedule);


        //---------------------------------------
        System.out.println("\nEDF with unprofitable jobs last ");
        Schedule EDFPSchedule = js.makeScheduleEDF();
       System.out.println(EDFPSchedule);

        //-------------------------------------
        System.out.println("\nSJF with unprofitable jobs last");
        Schedule SJFPSchedule = js.makeScheduleSJF();
        System.out.println(SJFPSchedule);

        //-----------------------------------------
        System.out.println("\nHPF with unprofitable jobs last");
        Schedule HPFSchedule = js.makeScheduleHPF();
        System.out.println(HPFSchedule);


        // ----------------------------
        System.out.println("\nYour own creative solution");
        Schedule NASSchedule = js.newApproxSchedule();
        System.out.println(NASSchedule);

    }
}
```