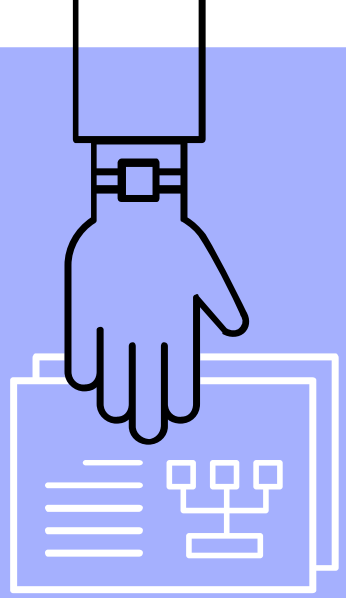
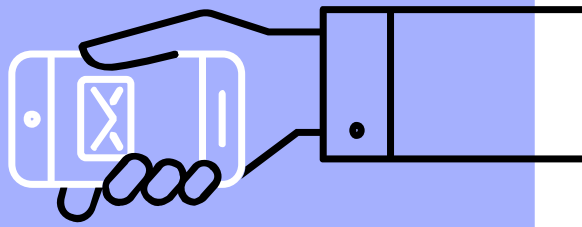
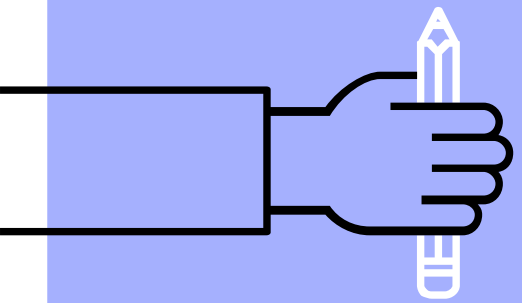


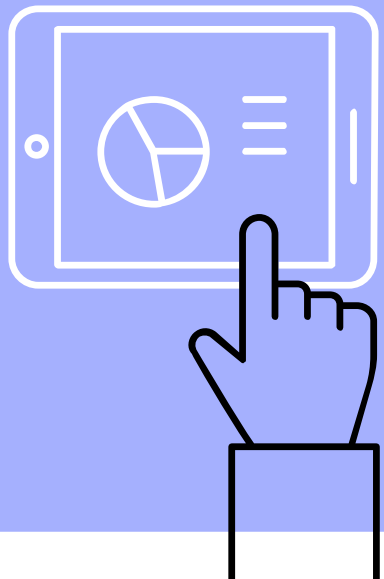


# JOB SEQUENCING

Sequencing on one processor

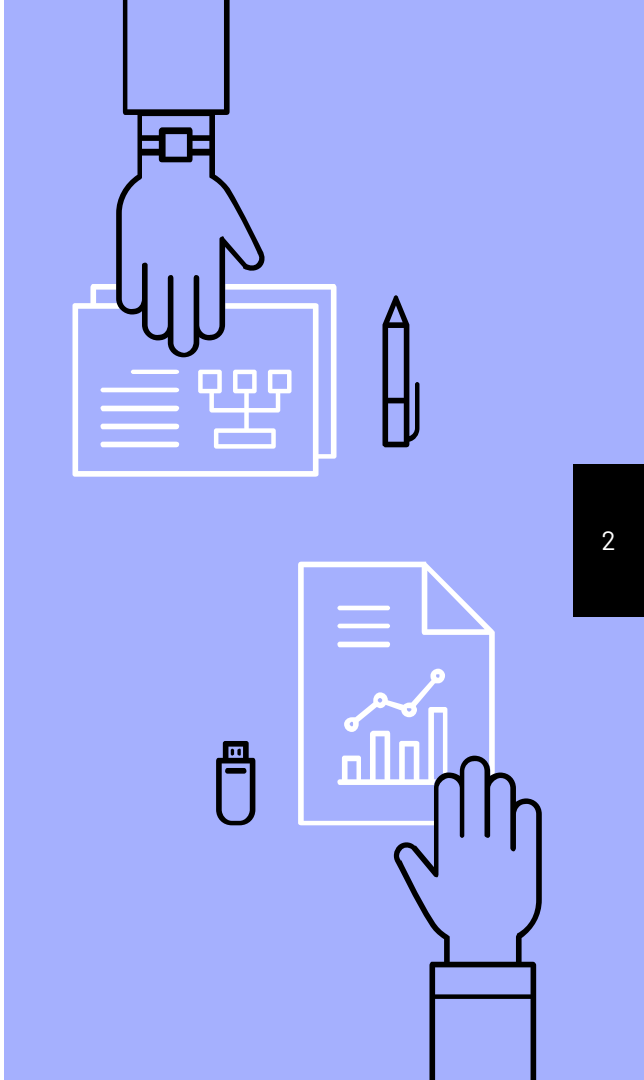
Sequencing and Scheduling

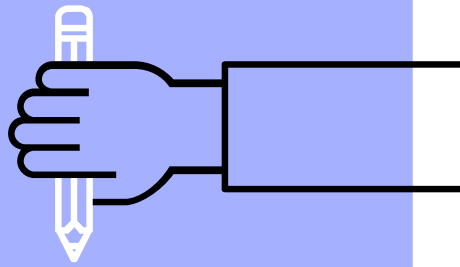




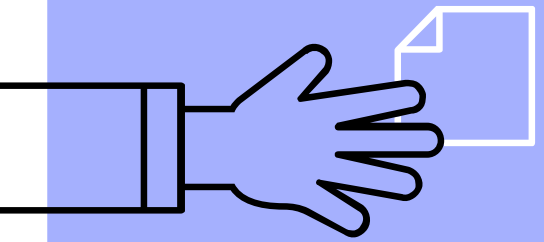
# OUTLINE

- Job Sequencing, what is it and how is it an NP Complete problem
- Demo





# JOB SEQUENCING

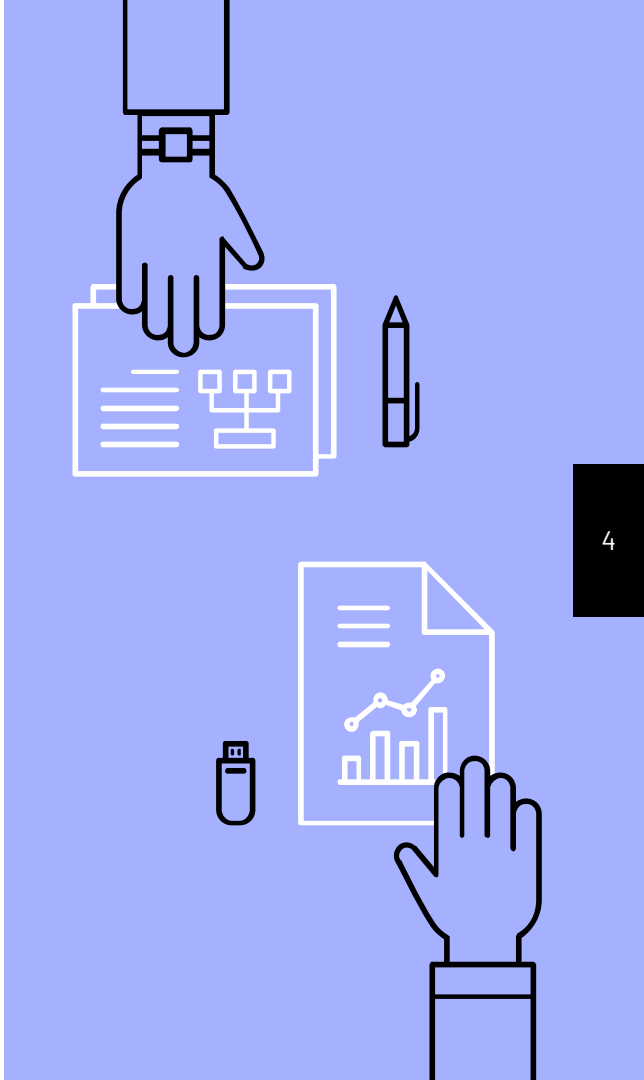


What is it and how is it NP Complete

# JOB SEQUENCING

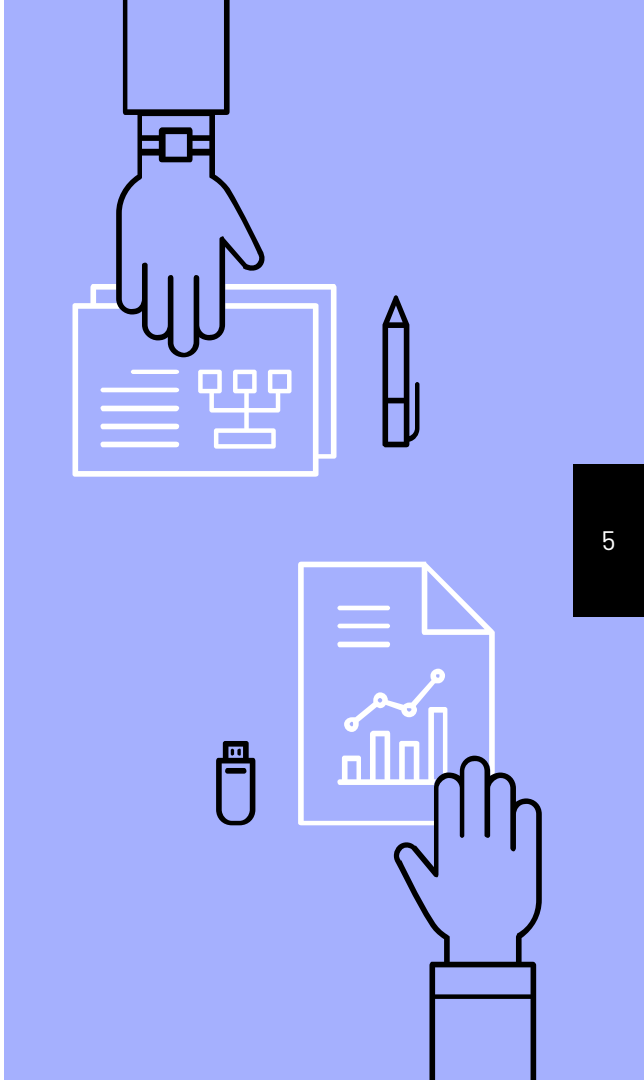
We have a set of n jobs to run on a processor (CPU) or machine

- Each job i has a deadline d*i* >=1 and profit p*i* >=0
- There is one processor or machine
- Each job takes 1 unit of time (simplification)



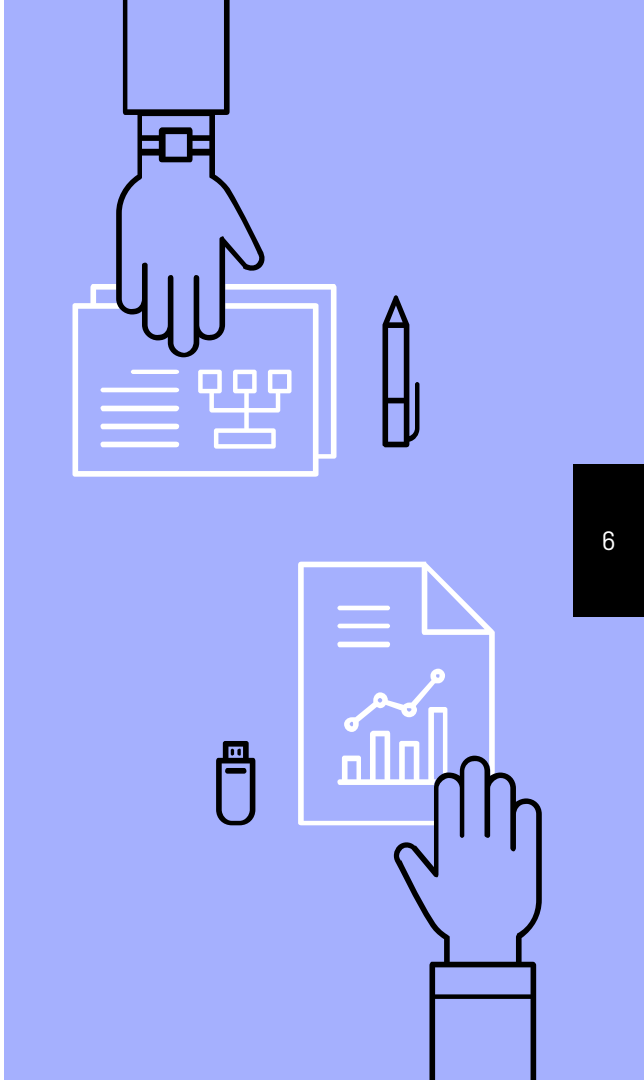
# JOB SEQUENCING

- We earn the profit if and only if the job is completed by its deadline
  - "Profit" can be the priority of the task in a real time system that discards tasks that cannot be completed by their deadline



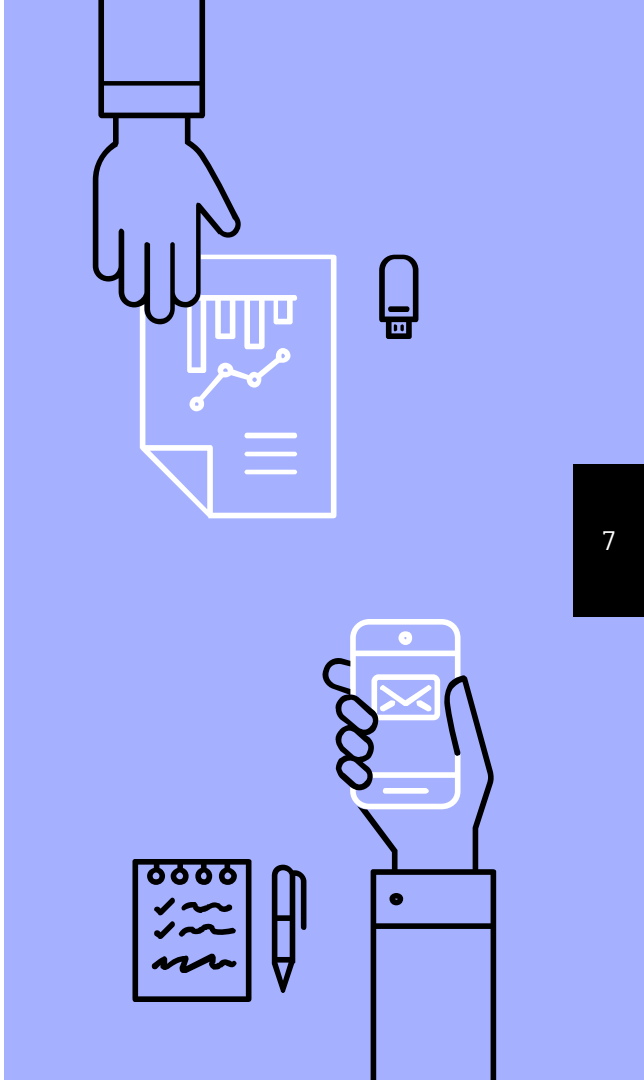
# GOAL

- We want to find the sequence of jobs that maximizes our profit



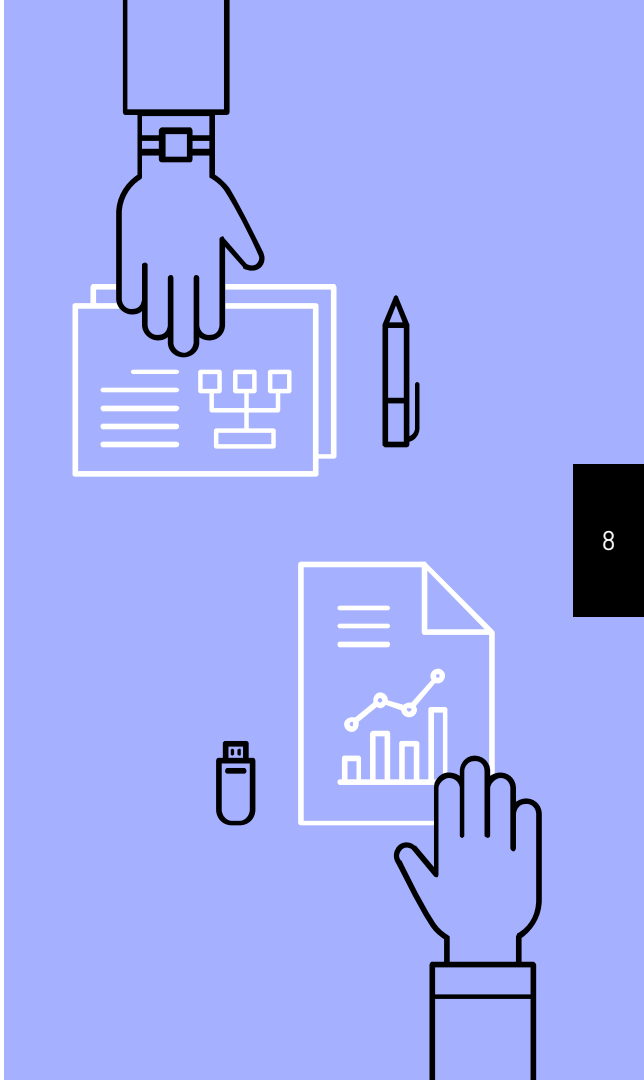
# ILLUSTRATION

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 4 | 20 |
| B | 1 | 10 |
| C | 1 | 40 |
| D | 1 | 30 |



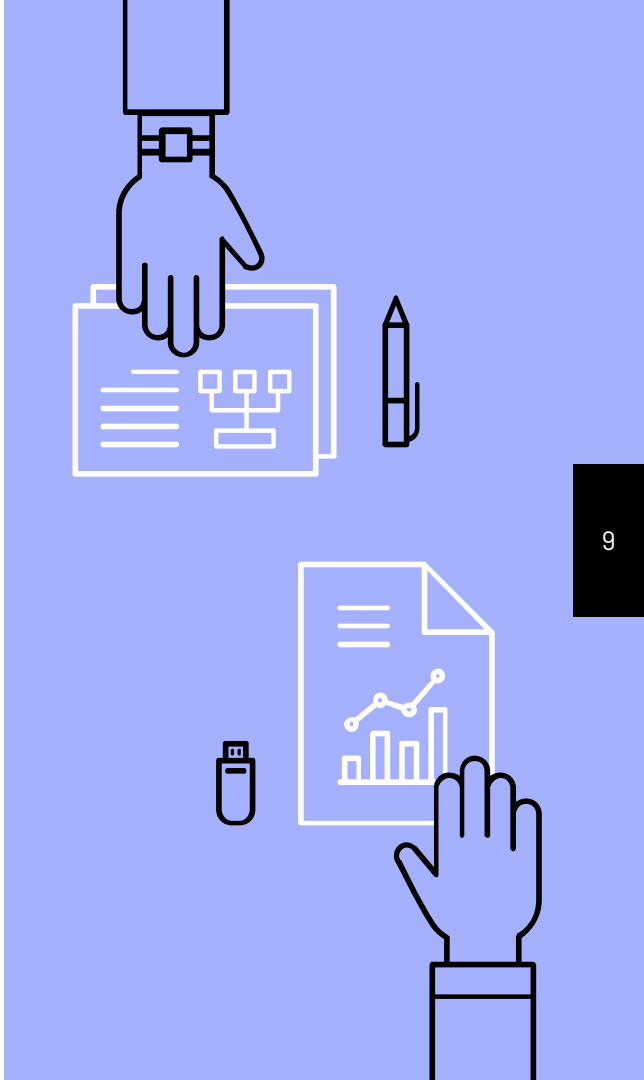
# NP COMPLETE

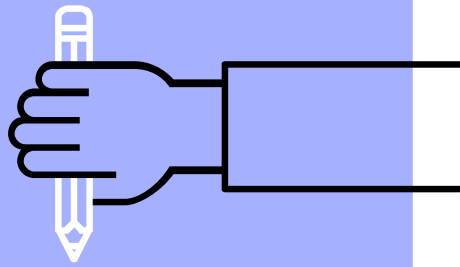
- Search problems with no known efficient algorithm to solve them.
  - Efficient = polynomial time algorithm
- solved in exponential time (at best) and uses non-deterministic method to solve the problem
- can solve it in polynomial time if we can break the normal rules of step-by-step computing



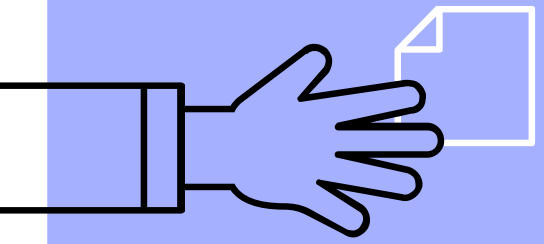
# NP COMPLETE

We show that the problem of finding an optimal schedule for a set of jobs is NP- complete even if all jobs require 1 time unit.





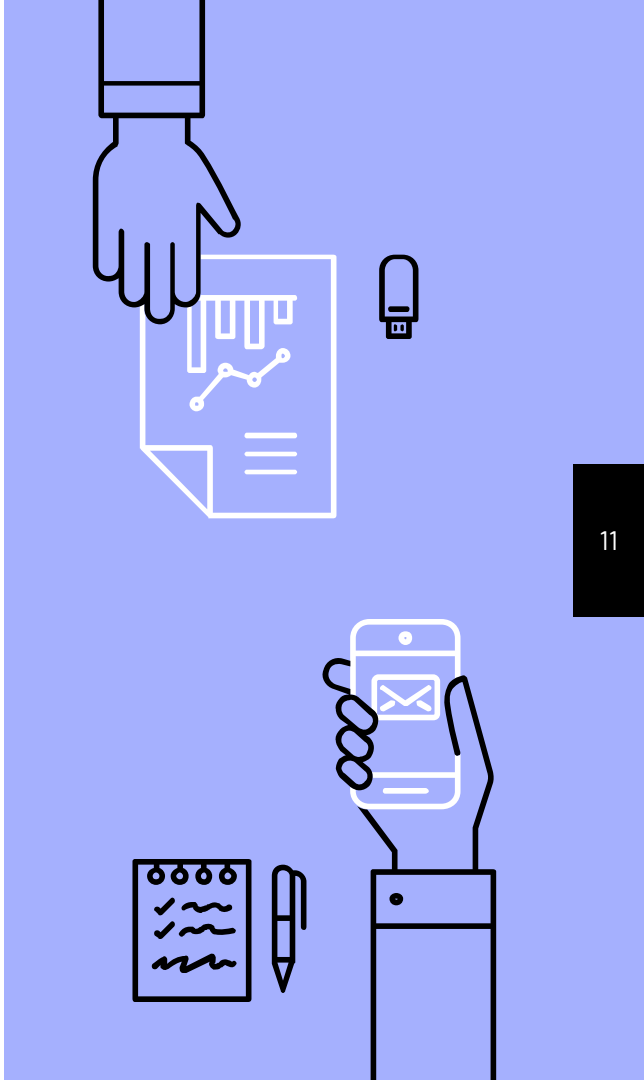
# DEMO



Illustrative and Program solutions via greedy algorithm

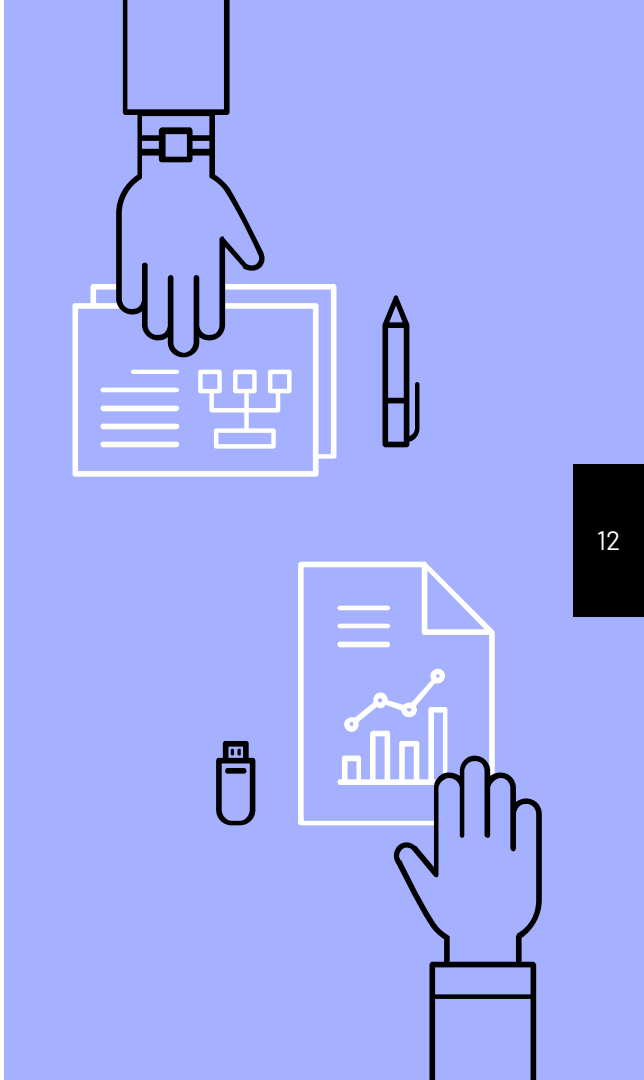
# ILLUSTRATION

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 4 | 20 |
| B | 1 | 10 |
| C | 1 | 40 |
| D | 1 | 30 |



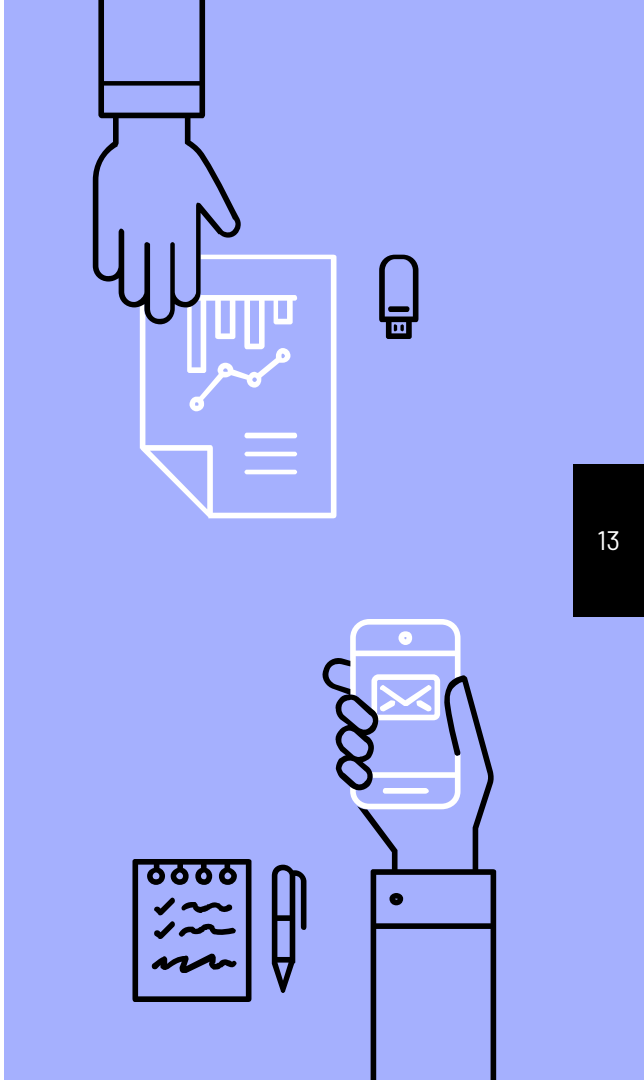
# GREEDY ALGORITHM

1. Sort all jobs in decreasing order of profit.
2. Initialize the result sequence as first job in sorted jobs.
3. Do following for remaining n-1 jobs
   - If the current job can fit in the current result sequence without missing the deadline, add current job to the result. Else ignore the current job.



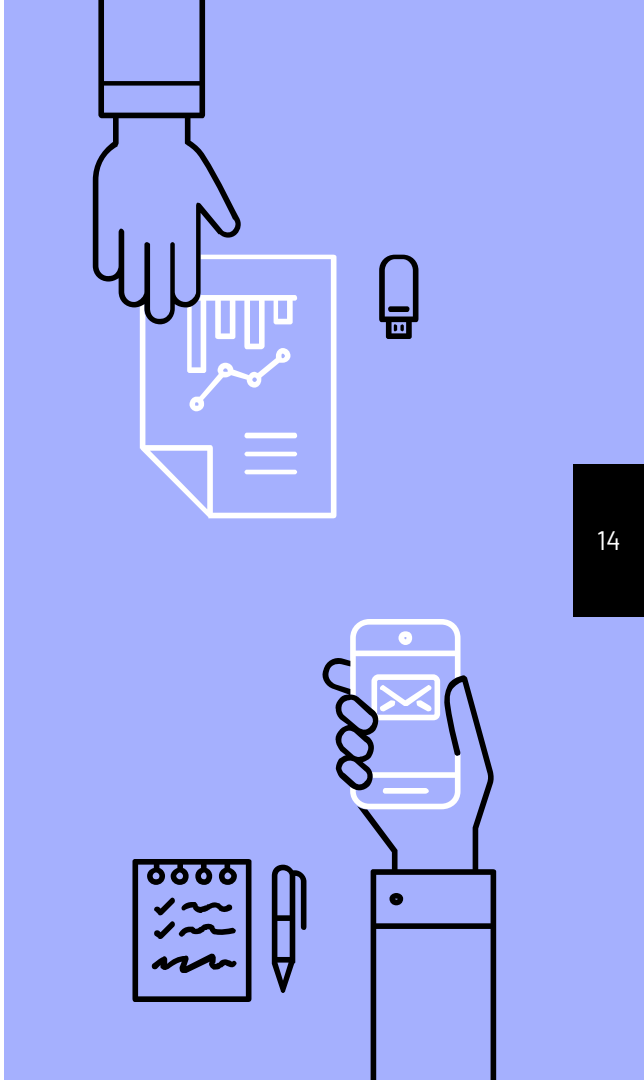
# GREEDY ALGORITHM

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 4 | 20 |
| B | 1 | 10 |
| C | 1 | 40 |
| D | 1 | 30 |



# Sort all jobs in decreasing order of profit.

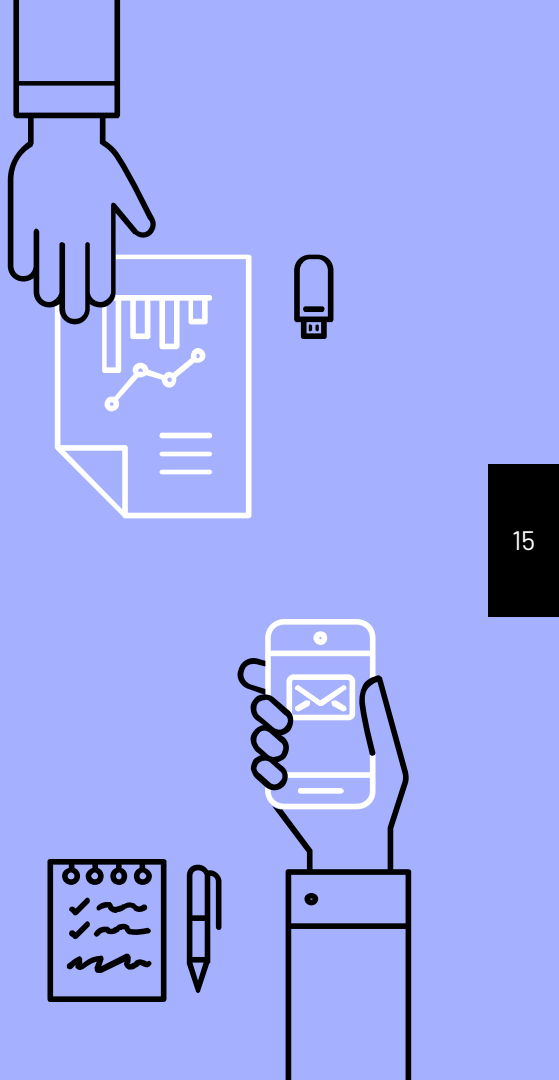
| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| C | 1 | 40 |
| D | 1 | 30 |
| A | 4 | 20 |
| B | 1 | 10 |



# Initialize the result sequence as first job in sorted jobs.

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| C | 1 | 40 |
| D | 1 | 30 |
| A | 4 | 20 |
| B | 1 | 10 |

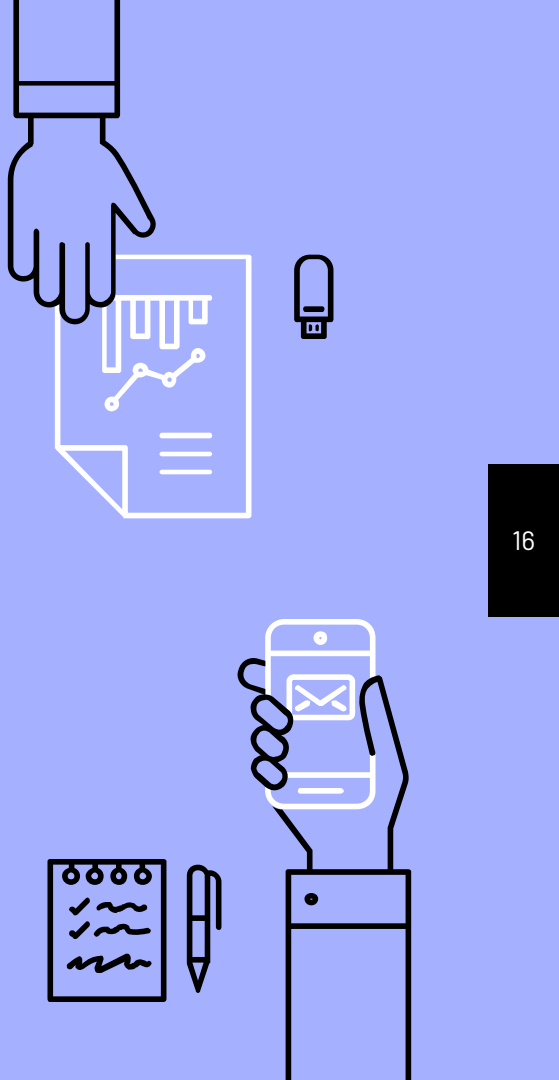
| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | | | |
| 1-2 | | | |
| 2-3 | | | |
| 3-4 | | | |



# Initialize the result sequence as first job in sorted jobs.

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| C | 1 | 40 |
| D | 1 | 30 |
| A | 4 | 20 |
| B | 1 | 10 |

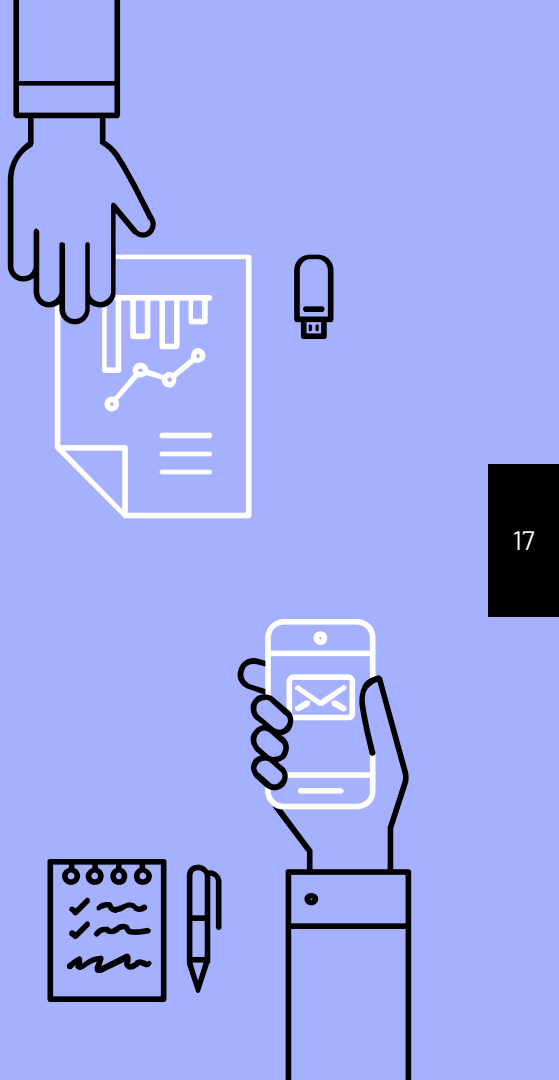
| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | C | 1 | 40 |
| 1-2 | | | |
| 2-3 | | | |
| 3-4 | | | |



# Initialize the result sequence as first job in sorted jobs.

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| C | 1 | 40 |
| D | 1 | 30 |
| A | 4 | 20 |
| B | 1 | 10 |

| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | C | 1 | 40 |
| 1-2 | A | 4 | 20 |
| 2-3 | | | |
| 3-4 | | | |

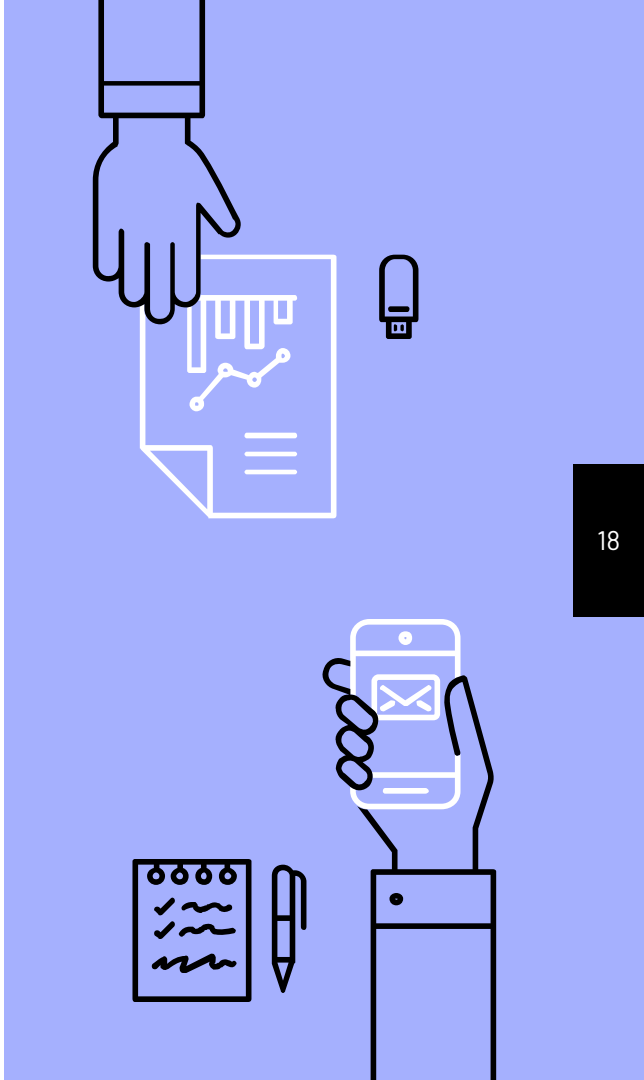


# Initialize the result sequence as first job in sorted jobs.

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| C | 1 | 40 |
| D | 1 | 30 |
| A | 4 | 20 |
| B | 1 | 10 |

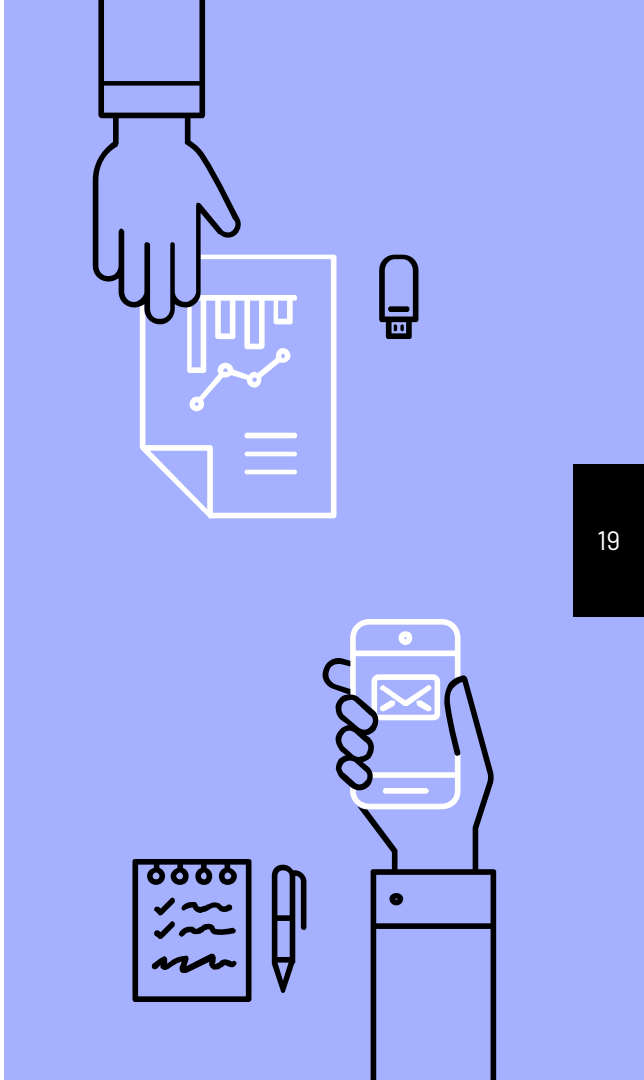
| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | C | 1 | 40 |
| 1-2 | A | 4 | 20 |

Thus, the maximum profit sequence of jobs is : C, A with a total profit of 60



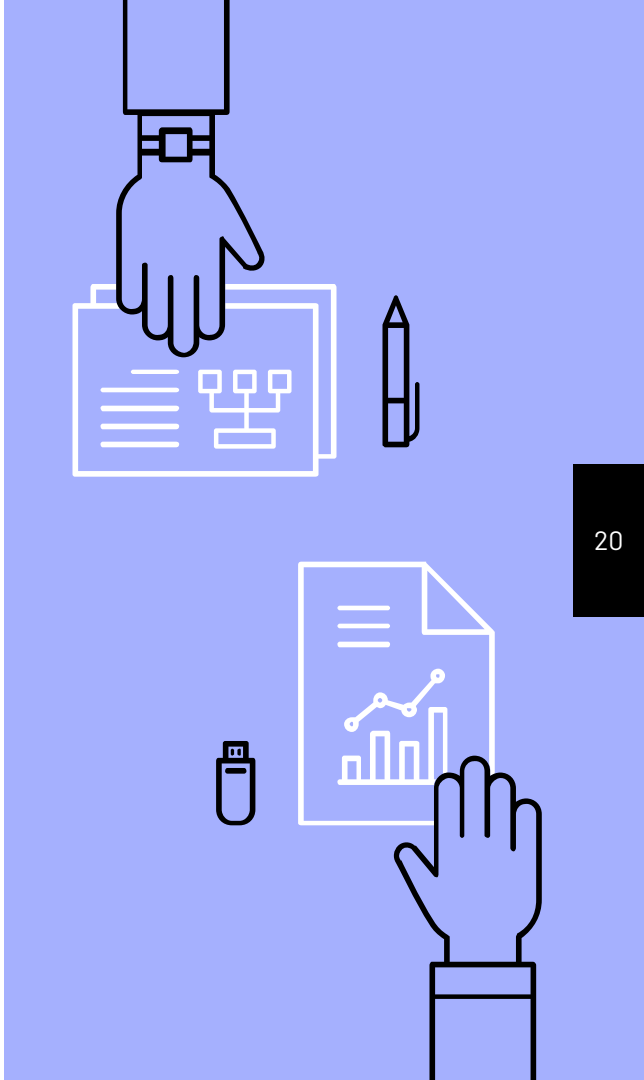
# ANOTHER EXAMPLE

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| B | 1 | 19 |
| C | 2 | 27 |
| D | 1 | 25 |
| E | 3 | 15 |



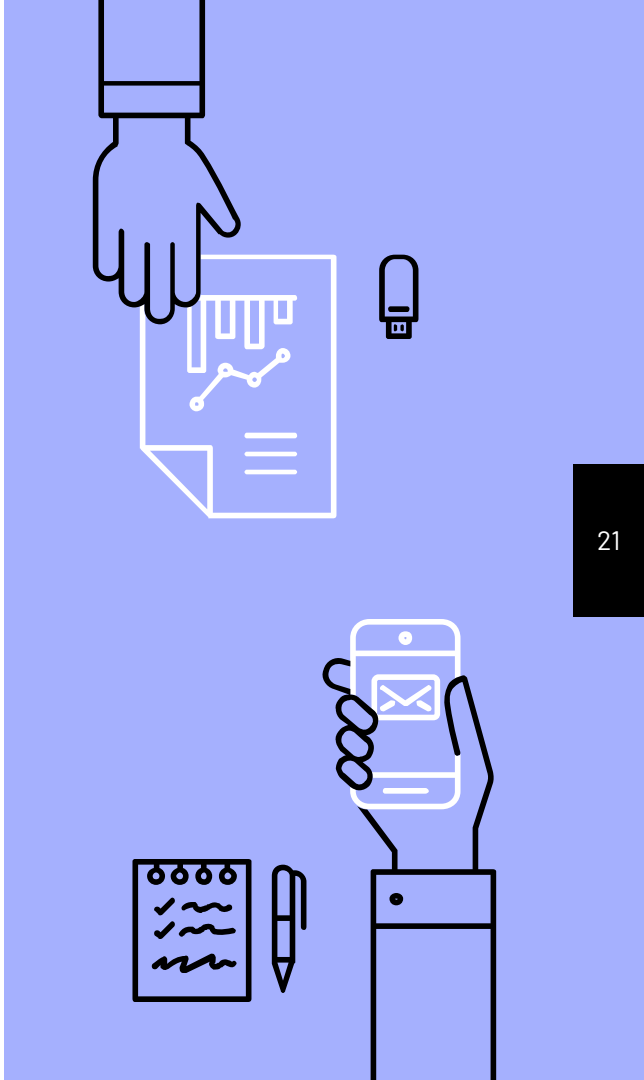
# GREEDY ALGORITHM

1. Sort all jobs in decreasing order of profit.
2. Place each job at latest time that meets its deadline
    - Nothing is gained by scheduling it earlier, and scheduling it earlier could prevent another more profitable job from being done
3. Solve



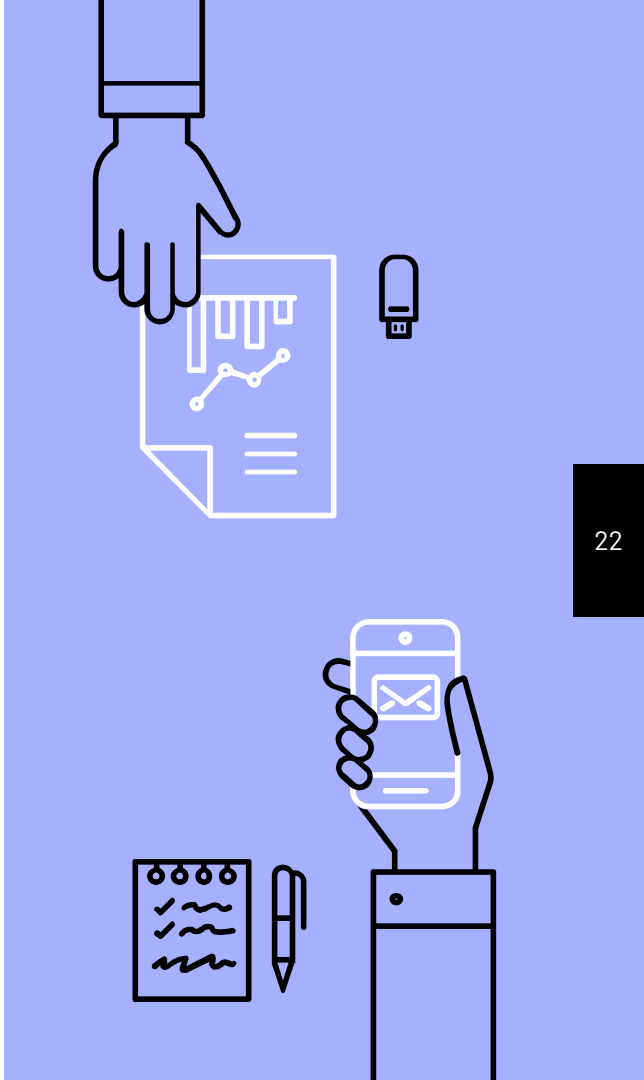
# Sort all jobs in decreasing order of profit.

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| B | 1 | 19 |
| C | 2 | 27 |
| D | 1 | 25 |
| E | 3 | 15 |



# Sort all jobs in decreasing order of profit.

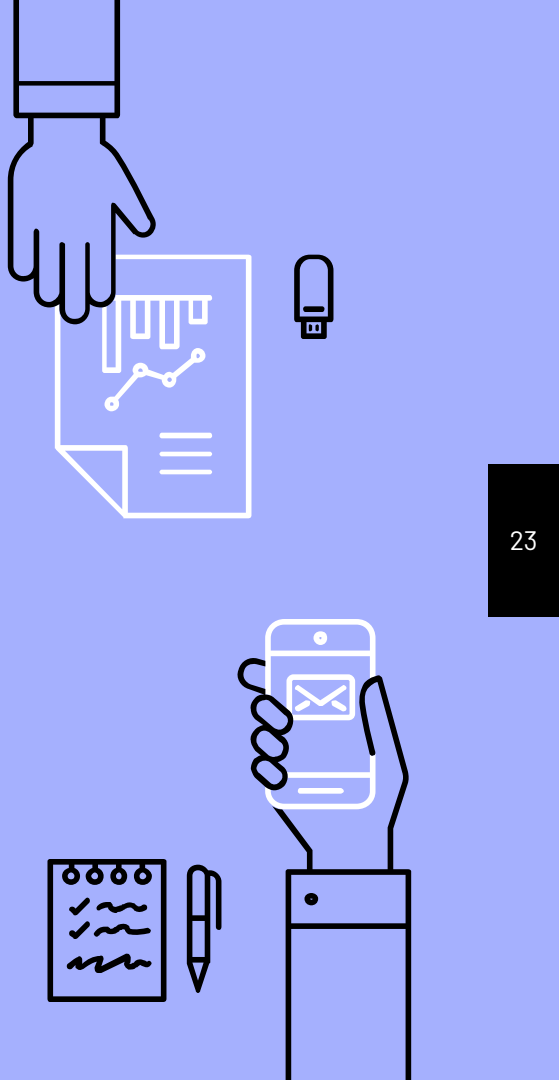
| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| C | 2 | 27 |
| D | 1 | 25 |
| B | 1 | 19 |
| E | 3 | 15 |



# Place each job at latest time that meets its deadline

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| C | 2 | 27 |
| D | 1 | 25 |
| B | 1 | 19 |
| E | 3 | 15 |

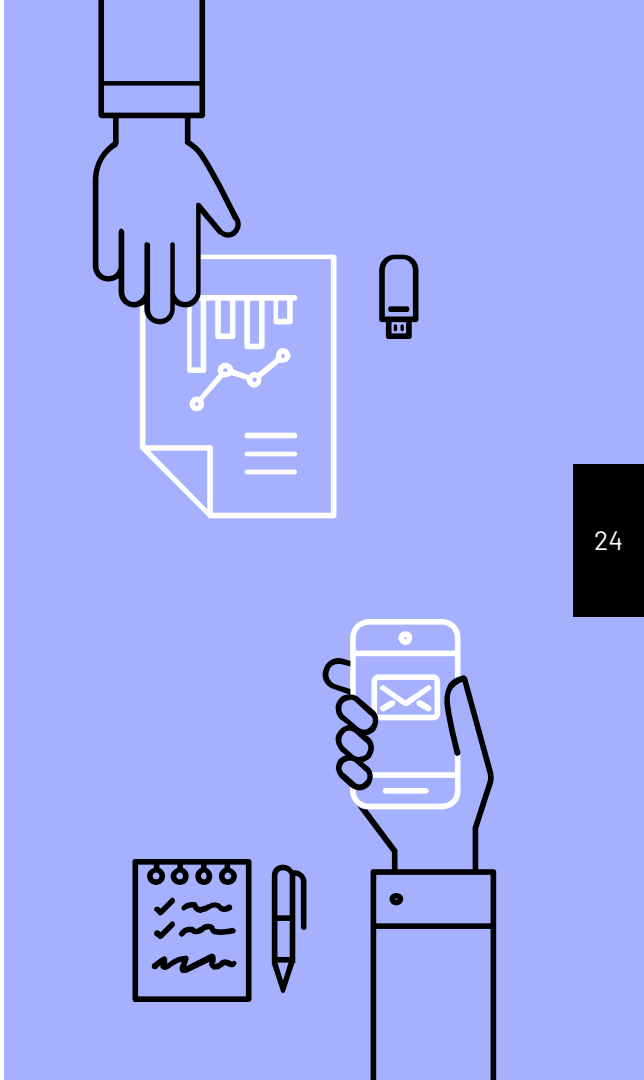
| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | | | |
| 1-2 | | | |
| 2-3 | | | |



# Place each job at latest time that meets its deadline

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| C | 2 | 27 |
| D | 1 | 25 |
| B | 1 | 19 |
| E | 3 | 15 |

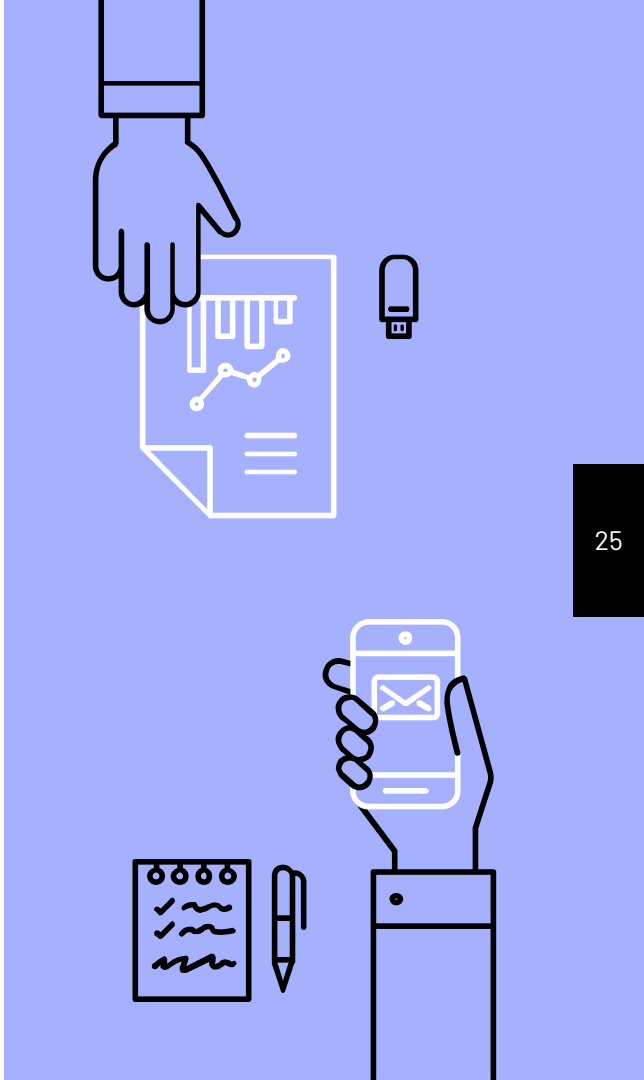
| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | | | |
| 1-2 | A | 2 | 100 |
| 2-3 | | | |



# Place each job at latest time that meets its deadline

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| C | 2 | 27 |
| D | 1 | 25 |
| B | 1 | 19 |
| E | 3 | 15 |

| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | C | 2 | 27 |
| 1-2 | A | 2 | 100 |
| 2-3 | | | |

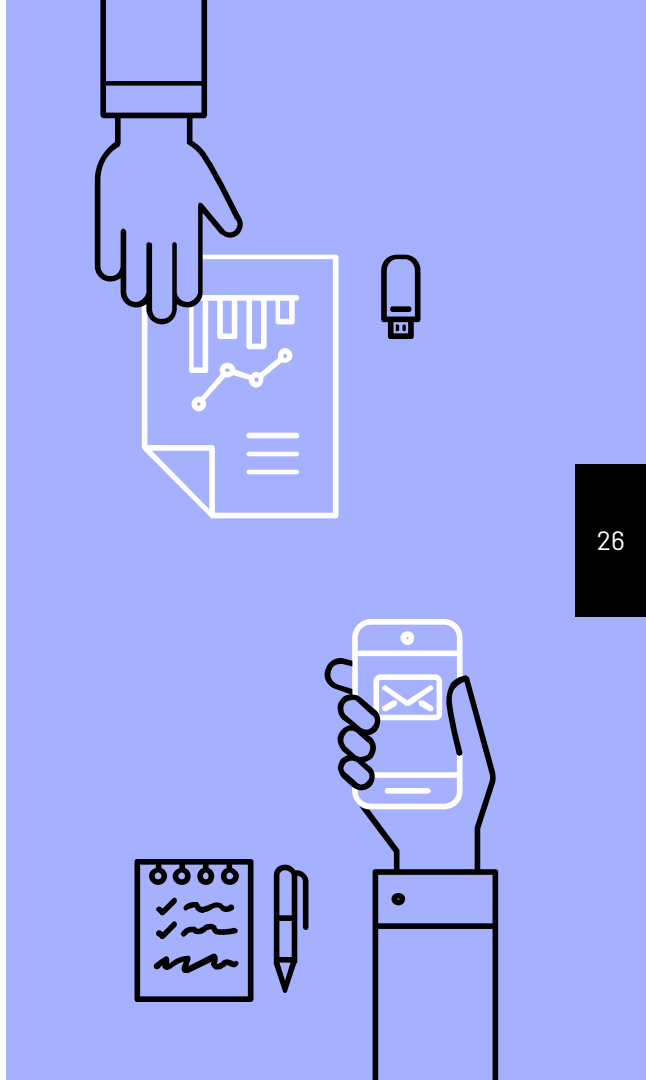


# Place each job at latest time that meets its deadline

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| C | 2 | 27 |
| D | 1 | 25 |
| B | 1 | 19 |
| E | 3 | 15 |

| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | C | 2 | 27 |
| 1-2 | A | 2 | 100 |
| 2-3 | E | 3 | 15 |

Thus, the maximum profit sequence of jobs is :C, A, E with a total of 142



# PROGRAM

Driver code

```
public static void main(String args[])
{
    ArrayList<Job> arr = new ArrayList<Job>();

    arr.add(new Job('a', 2, 100));
    arr.add(new Job('b', 1, 19));
    arr.add(new Job('c', 2, 27));
    arr.add(new Job('d', 1, 25));
    arr.add(new Job('e', 3, 15));

    // Function call
    System.out.println("Following is maximum "
                       + "profit sequence of jobs");

    Job job = new Job();

    // Calling function
    job.printJobScheduling(arr, 3);
}
```

# PROGRAM

```
arr.add(new Job('a', 2, 100));
arr.add(new Job('b', 1, 19));
arr.add(new Job('c', 2, 27));
arr.add(new Job('d', 1, 25));
arr.add(new Job('e', 3, 15));
```

```
import java.util.*;

class Job
{
    // Each job has a unique-id,
    // profit and deadline
    char id;
    int deadline, profit;

    // Constructors
    public Job() {}

    public Job(char id, int deadline, int profit)
    {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
```

# PROGRAM

```
arr.add(new Job('a', 2, 100));
arr.add(new Job('b', 1, 19));
arr.add(new Job('c', 2, 27));
arr.add(new Job('d', 1, 25));
arr.add(new Job('e', 3, 15));
```

```
// Function to schedule the jobs take 2
// arguments arraylist and no of jobs to schedule
void printJobScheduling(ArrayList<Job> arr, int t)
{
    // Length of array
    int n = arr.size();

    // Sort all jobs according to
    // decreasing order of profit
    Collections.sort(arr,
                     (a, b) -> b.profit - a.profit);

    // To keep track of free time slots
    boolean result[] = new boolean[t];

    // To store result (Sequence of jobs)
    char job[] = new char[t];
```

# PROGRAM

```
arr.add(new Job('a', 2, 100)); 0
arr.add(new Job('b', 1, 19));  3
arr.add(new Job('c', 2, 27));  1
arr.add(new Job('d', 1, 25));  2
arr.add(new Job('e', 3, 15));  4
```

```
// Calling function
job.printJobScheduling(arr, 3);
```

```
    // Iterate through all given jobs
    for (int i = 0; i < n; i++)
    {
        // Find a free slot for this job
        // (Note that we start from the
        // last possible slot)
        for (int j
             = Math.min(t - 1, arr.get(i).deadline - 1);
             j >= 0; j--) {

            // Free slot found
            if (result[j] == false)
            {
                result[j] = true;
                job[j] = arr.get(i).id;
                break;
            }
        }
    }

    // Print the sequence
    for (char jb : job)
    {
        System.out.print(jb + " ");
    }
    System.out.println();
}
```

# PROGRAM

Driver code

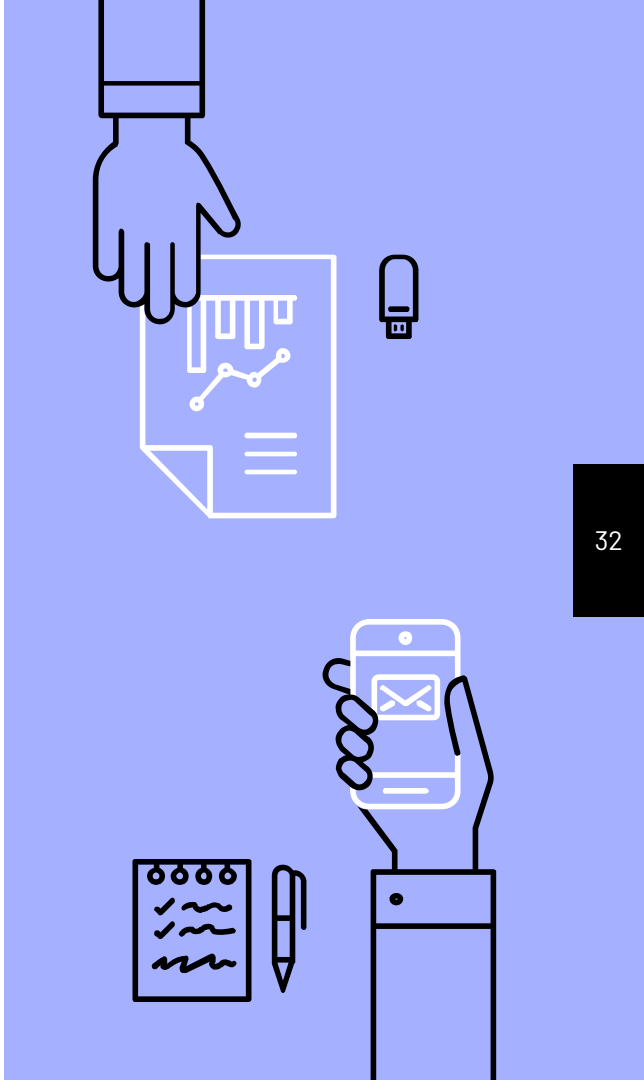
```
public static void main(String args[])
{
    ArrayList<Job> arr = new ArrayList<Job>();

    arr.add(new Job('a', 2, 100));
    arr.add(new Job('b', 1, 19));
    arr.add(new Job('c', 2, 27));
    arr.add(new Job('d', 1, 25));
    arr.add(new Job('e', 3, 15));

    // Function call
    System.out.println("Following is maximum "
                       + "profit sequence of jobs");

    Job job = new Job();

    // Calling function
    job.printJobScheduling(arr, 3);
}
```

# Place each job at latest time that meets its deadline

| JOB ID | DEADLINE | PROFIT |
|---|---|---|
| A | 2 | 100 |
| C | 2 | 27 |
| D | 1 | 25 |
| B | 1 | 19 |
| E | 3 | 15 |

| TIME | JOB ID | DEADLINE | PROFIT |
|---|---|---|---|
| 0-1 | C | 2 | 27 |
| 1-2 | A | 2 | 100 |
| 2-3 | E | 3 | 15 |

Thus, the maximum profit sequence of jobs is :C, A, E



# THANK YOU

End of Presentation

# REFERENCES


Links

https://www.geeksforgeeks.org/job-sequencing-problem/

https://ocw.mit.edu/courses/civil-and-environmental-engineering/1-204-computer-algorithms-in-systems-engineering-spring-2010/lecture-notes/MIT1_204S10_lec10.pdf

https://core.ac.uk/reader/82723490

https://www.youtube.com/watch?v=zPtI8q9gvX8&t=16s

https://www.mathsisfun.com/sets/np-complete.html

APA Citation

3.2 Job Sequencing with Deadlines - Greedy Method.(2018, February 07). Retrieved December 03, 2021, from https://www.youtube.com/watch?v=zPtI8q9gvX8&t=16s

Computer Algorithms in Systems Engineering.(2010). https://bit.ly/3Dprups

Job Sequencing Problem. (2014, November 18). GeeksforGeeks. https://www.geeksforgeeks.org/job-sequencing-problem/

Ullman, J.D. . NP-Complete Scheduling Problems.(1973, May 16). https://core.ac.uk/reader/82723490