COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

Algorithm

An algorithm is **a finite, ordered sequence** of well-defined instructions designed to transform input into output that satisfies **a given specification**.

A specification defines what the system or program must do — the desired behavior or goal.

An algorithm defines how that goal is achieved through a step-by-step logical process.

Specification → describes the problem.

Algorithm → describes the solution approach.

Program → implements the algorithm.

Algorithm

Specification —

Given (Input) — A list of numbers $L = [a_1, a_2, \ldots, a_n]$ and a number x

Output — Find the index (position) of x in the list if it is present, else output -1.

Algorithm

Specification —

Given (Input) — A list of numbers $L = [a_1, a_2, \ldots, a_n]$ and a number x

Output — Find the index (position) of x in the list if it is present, else output -1.

1. Start the process. We are given a list of integers and the target integers we need to find.
2. Begin the search from the very beginning of the list.
3. Compare the current element with the target number.
4. If they are the same,
    - We have found the number — return its position (index) and stop the process.
5. If they are not the same, move on to the next element in the list and repeat the comparison.
6. Continue this process (3-5) until:
    - Either we find the number, or we have checked all the elements in the list.
7. If the number is not found after checking every element, then return -1 , stop the process.

Algorithm

Specification —

Given (Input) — A list of numbers $L = [a_1, a_2, \ldots, a_n]$ and a number x

Output — Find the index (position) of x in the list if it is present, else output -1.

Algorithm

Specification —

Given (Input) — A list of numbers $L = [a_1, a_2, \ldots, a_n]$ and a number x

Output — Find the index (position) of x in the list if it is present, else output -1.

```
Algorithm: Find the Index of a Given Number

1. Start
2. Set an index variable i = 0
3. Repeat while i < n:
      a. If L[i] = x:
            i. Output i (the index of the number)
            ii. Stop the algorithm
      b. Else, increment i = i + 1
4. If the end of the list is reached and no match is found:
      Output -1
5. End
```

Algorithm

```
Algorithm: Find the Index of a Given Number

1. Start
2. Set an index variable i = 0
3. Repeat while i < n:
      a. If L[i] = x:
            i. Output i (the index of the number)
            ii. Stop the algorithm
      b. Else, increment i = i + 1
4. If the end of the list is reached and no match is found:
      Output -1
5. End
```

Algorithm

```
Algorithm: Find the Index of a Given Number

1. Start
2. Set an index variable i = 0
3. Repeat while i < n:
      a. If L[i] = x:
            i. Output i (the index of the number)
            ii. Stop the algorithm
      b. Else, increment i = i + 1
4. If the end of the list is reached and no match is found:
      Output -1
5. End
```

In best case — if the element is found at the beginning — 1 step
In the worst case — if the element is at the end or not present — n step
Extra space — nope, no extra space required.

Algorithm

```
Algorithm: Find the Index of a Given Number

1. Start
2. Set an index variable i = 0
3. Repeat while i < n:
      a. If L[i] = x:
            i. Output i (the index of the number)
            ii. Stop the algorithm
      b. Else, increment i = i + 1
4. If the end of the list is reached and no match is found:
      Output -1
5. End
```

In best case — if the element is found at the beginning — 1 step
In the worst case — if the element is at the end or not present — n step
Extra space — nope, no extra space required.

} Time complexity

Algorithm

```
Algorithm: Find the Index of a Given Number

1. Start
2. Set an index variable i = 0
3. Repeat while i < n:
      a. If L[i] = x:
            i. Output i (the index of the number)
            ii. Stop the algorithm
      b. Else, increment i = i + 1
4. If the end of the list is reached and no match is found:
      Output -1
5. End
```

In best case — if the element is found at the beginning — 1 step
In the worst case — if the element is at the end or not present — n step } Time complexity

Extra space — nope, no extra space required. } Space complexity

Algorithm

```
Algorithm: Find the Index of a Given Number

1. Start
2. Set an index variable i = 0
3. Repeat while i < n:
      a. If L[i] = x:
            i. Output i (the index of the number)
            ii. Stop the algorithm
      b. Else, increment i = i + 1
4. If the end of the list is reached and no match is found:
      Output -1
5. End
```

} Linear Search

In best case — if the element is found at the beginning — 1 step
In the worst case — if the element is at the end or not present — n step
} Time complexity

Extra space — nope, no extra space required.
} Space complexity

The Activity Selection Problem

You are given several activities with start and finish times. You can perform only one activity at a time. The goal is to select the maximum number of non-overlapping activities.

Activity	Start	Finish
A1	1	3
A2	4	6
A3	2	5
A4	6	7
A5	5	8
A6	8	9

The Activity Selection Problem

You are given several activities with start and finish times. You can perform only one activity at a time. The goal is to select the maximum number of non-overlapping activities.

Activity	Start	Finish
A1	1	3
A2	4	6
A3	2	5
A4	6	7
A5	5	8
A6	8	9

Each activity i has a start time s[i] and a finish time f[i].
Two activities i and j can be scheduled if their time intervals do not overlap, i.e. f[i] ≤ s[j] or f[j] ≤ s[i].

The Activity Selection Problem

Idea 1

The Activity Selection Problem

Idea 1

1. Generate **all possible subsets** of the n activities.
2. For each subset:
    - Check if no two activities overlap.
    - If compatible, count the number of activities in it.
3. Keep track of the subset with the largest count.

The Activity Selection Problem

Idea 1

1. Generate **all possible subsets** of the n activities.
2. For each subset:
   - Check if no two activities overlap.
   - If compatible, count the number of activities in it.
3. Keep track of the subset with the largest count.

Time ?

Works ?

— How many possible subsets — 2^n

— Total time = number of subsets × time per subset

The Activity Selection Problem

1. Generate **all possible subsets** of the n activities.
2. For each subset:
    - Check if no two activities overlap.
    - If compatible, count the number of activities in it.
3. Keep track of the subset with the largest count.

} Brute force approach

The Activity Selection Problem

1. Generate **all possible subsets** of the n activities.
2. For each subset:
    - Check if no two activities overlap.
    - If compatible, count the number of activities in it.
3. Keep track of the subset with the largest count.

} Brute force approach

Brute force approach (also called the exhaustive search) is a straightforward problem-solving method that tries all possible solutions and selects the one that satisfies the required condition or gives the best result

It systematically goes through **every possible option** until it finds the correct or optimal one.

The Activity Selection Problem

Idea 2

The Activity Selection Problem

Idea 2

1. Sort all activities by their finish time
2. Select the first activity (the one that finishes earliest)
3. For each subsequent activity:
    - If its start time is greater than or equal to the finish time of the last selected activity, select it
4. Continue until all activities are checked

The Activity Selection Problem

Idea 2

1. Sort all activities by their finish time
2. Select the first activity (the one that finishes earliest)
3. For each subsequent activity:
    - If its start time is greater than or equal to the finish time of the last selected activity, select it
4. Continue until all activities are checked

Works ?

Time ?
— Once sorted, you need to go through all activities for selection.
Total time = Sorting + Selection = Sorting + n.

The Activity Selection Problem

Activity	Start	Finish
A1	1	3
A2	4	6
A3	2	5
A4	6	7
A5	5	8
A6	8	9

Sorted as per finish time →

Activity	Start	Finish
A1	1	3
A3	2	5
A2	4	6
A4	6	7
A5	5	8
A6	8	9

After sorting by finish time: A1, A2, A4, A5, A6

Select A1 → (finish = 3)

A2 starts at $4 \geq 3$ → select A3 (finish = 6)

A4 starts at $6 \geq 6$ → select A4 (finish = 7)

A6 starts at $8 \geq 7$ → select A6 (finish = 9)

The Activity Selection Problem

the idea relies on the principle of making the locally optimal choice — always pick the next activity that finishes earliest among the remaining compatible ones.

The Activity Selection Problem

1. Sort all activities by their finish time
2. Select the first activity (the one that finishes earliest)
3. For each subsequent activity:
    - If its start time is greater than or equal to the finish time of the last selected activity, select it
4. Continue until all activities are checked

the idea relies on the principle of making the locally optimal choice — always pick the next activity that finishes earliest among the remaining compatible ones.

The Activity Selection Problem

1. Sort all activities by their finish time
2. Select the first activity (the one that finishes earliest)
3. For each subsequent activity:
    - If its start time is greater than or equal to the finish time of the last selected activity, select it
4. Continue until all activities are checked

the idea relies on the principle of making the locally optimal choice — always pick the next activity that finishes earliest among the remaining compatible ones.

always choosing the next piece that looks best at the moment — without worrying about the consequences for future choices.

The Activity Selection Problem

1. Sort all activities by their finish time
2. Select the first activity (the one that finishes earliest)
3. For each subsequent activity:
    - If its start time is greater than or equal to the finish time of the last selected activity, select it
4. Continue until all activities are checked

Greedy approach

the idea relies on the principle of making the locally optimal choice — always pick the next activity that finishes earliest among the remaining compatible ones.

always choosing the next piece that looks best at the moment — without worrying about the consequences for future choices.

There will be no classes on 3rd, 6th, and 7th November.

A make-up class will be held on 8th November from 3:00 p.m. to 5:30 p.m. in Room No. 408.

You are also welcome to attend Sayan's class for the coming week if you wish.