LONGEST INCREASING SUBSEQUENCES

Presented by Team AIN

Course Title: Data Structure and Algorithm II Laboratory
Course Code: CSE 2218
Section: C

Course Faculty: Charles Aunkan Gomes
Lecturer, Department of Computer Science and Engineering
United International University

Group Information

Serial	Full Name	University ID
01	Azizul Islam Nayem	011201262
02	Md.Aman Ullah Faisal	011192136
03	Sadman Rafid	011181029
04	Nayeem Muhammad Al Farabi Sikder	011201269
05	Md Asad Jaman	011201239

Topics that will be covered here

1. Problem Description.
2. Algorithm Explain.
3. Simulation.
4. Implementation.
5. Time Complexity Analysis.

Definition

The longest increasing subsequence refers to the longest sequence of numbers in an array where each number is greater than the one before it. It doesn't have to be consecutive; it's about selecting elements from the array in a way that they're in increasing order.

Usages of LIS

Algorithm Practice: Common problem in algorithm courses and programming contests.

Data Compression: Identifies patterns for efficient data compression.

Financial Analysis: Detects trends in financial data for predictive analysis.

Genomics: Analyzes DNA sequences to uncover genetic patterns.

Network Routing: Optimizes data flow through nodes for better network performance.

Game Theory: Identifies winning strategies in strategy games.

Speech & Language: Extracts patterns in linguistic data for speech and language tasks.

Optimization Problems: Solves various optimization challenges involving ordered sequences.

Problem Description

Given array: {11, 23, 10, 32, 20, 49, 42, 61}

Subsequence: {11} or {11,10,32} or {23,20,61} and so on.

Increasing Subsequence: {11,23,32} or {10,20,49} or {32,49,61} and so on.

Notice that: {11} or {23} or {10} or {32} or {20} or {49} or {42} or {61} is also increasing subsequence.

But, Here, **Longest Increasing Subsequence(LIS)** is {11,23,32,49,61} or {10,20,32,42,61} and Length is 5.

Solving Technique

To crack the LIS algorithm, our weapon of choice is the **dynamic programming** approach.

Advantages:

Optimal Substructure: DP breaks the problem into smaller subproblems with clear relationships.

Overlapping Subproblems: DP avoids redundant calculations by storing and reusing results.

Memoization: DP optimizes by saving intermediate results in a table.

Bottom-Up Approach: Solves small sub-problems first and builds up, improving efficiency.

Efficiency: DP improves time complexity compared to recursive approaches.

Ease of Understanding: DP with memoization is intuitive and easier to implement.

Flexibility: Adaptable to handle variations and related problems.

Algorithm

- Initialize a DP array dp of the same size as the input array, filled with 1. This array will store the lengths of increasing subsequences ending at each index.
- Start iterating through the input array from the second element (index 1) to the last element (index n-1).
- For each element at index i, compare it with all the elements before it (from index 0 to i-1).
- If the element at index i is greater than the element at index j (where j varies from 0 to i-1), it means we can extend the LIS by including the current element (arr[i]).
- Update the dp[i] with the maximum value between its current value and the length of the LIS ending at index j + 1. This step ensures that we consider the longest increasing subsequence ending at index j and append the current element (arr[i]) to it if it can form a longer subsequence.
- After the loop ends, the dp array will contain the lengths of the LIS ending at each index.
- Find the maximum value in the dp array, which represents the length of the longest increasing subsequence in the given array.
- Return the maximum value as the result, which is the length of the LIS.

Simulation

For the value of i= 1 and j=0;

```
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator	j	i						
Array	11	23	10	32	20	49	42	61
LIS	1	1	1	1	1	1	1	1

Initially all 1

For the value of i= 1, j=0;

23>11? **Yes. So, LIS = {11,23}**

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator	j	i						
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	1	1	1	1	1

For the value of i= 2, j=0;

10>11? **No. So, LIS = {11,23}**

iterator	j	~~i~~	i					
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	1	1	1	1	1

```
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

For the value of i= 2, j=1;

10>23? **No. So, LIS = {11,23}**

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator	~~j~~	j	i					
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	1	1	1	1	1

For the value of i= 3, j=0;

32>11? **Yes. But, LIS = {11,23,?}**

```
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator	j		~~i~~	i				
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	1	1	1	1	1

For the value of i= 3, j=1;

32>23? **Yes. But, LIS = {11,23,?}**

```
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator	~~j~~	j		i				
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	~~1~~ **2**	1	1	1	1

For the value of i= 3, j=2;

32>10? **No. So, LIS = {11,23,32}**

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator		~~j~~	j	i				
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	~~2~~ 3	1	1	1	1

For the value of i= 4, j=0;

```
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

20>11? **Yes. But, LIS = {11,23,32,?}**

iterator	**j**			~~**i**~~	**i**			
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	1	1	1	1

For the value of i= 4, j=1;

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

20>23? **No. So, LIS = {11,23,32}**

iterator	~~j~~	**j**			**i**			
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	~~1~~ **2**	1	1	1

For the value of i= 4, j=2;

20>10? **Yes. But, LIS = {11,23,32}**

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator		~~j~~	j		i			
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	1	1	1

For the value of i= 4, j=3;

20>32? **No. So, LIS = {11,23,32}**

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

iterator			~~j~~	j	i			
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	1	1	1

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

For the value of i= 5, j=0;

49>11? **Yes. But, LIS = {11,23,32,?}**

iterator	**j**				~~**i**~~	**i**		
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	1	1	1

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

For the value of i= 5, j=1;

49>23? **Yes. But, LIS = {11,23,32,?}**

iterator	~~j~~	**j**				**i**		
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	~~1~~ **2**	1	1

For the value of i= 5, j=2;

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

49>10? **Yes. But, LIS = {11,23,32,?}**

iterator		~~j~~	**j**			**i**		
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	~~2~~ **3**	1	1

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

For the value of i= 5, j=3;

49>32? **Yes. But, LIS = {11,23,32,?}**

iterator			~~j~~	**j**		**i**		
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	3	1	1

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

For the value of i= 5, j=4;

49>20? **No. So, LIS = {11,23,32,49}**

iterator				~~j~~	j	i		
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	~~3~~ **4**	1	1

Same steps goes,
for of i= 6, when j= 0-5; and,
for of i= 7, when j= 0-5;
That's why skipping those steps.

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

```cpp
int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}
```

For the value of i= 7, j=6;

Finally,

61>42? **Yes. So, LIS = {11,23,32,49,61}**

iterator							j	i
Array	11	23	10	32	20	49	42	61
LIS	1	2	1	3	2	4	~~1 2 3~~ **4**	~~1 2 3 4~~ **5**

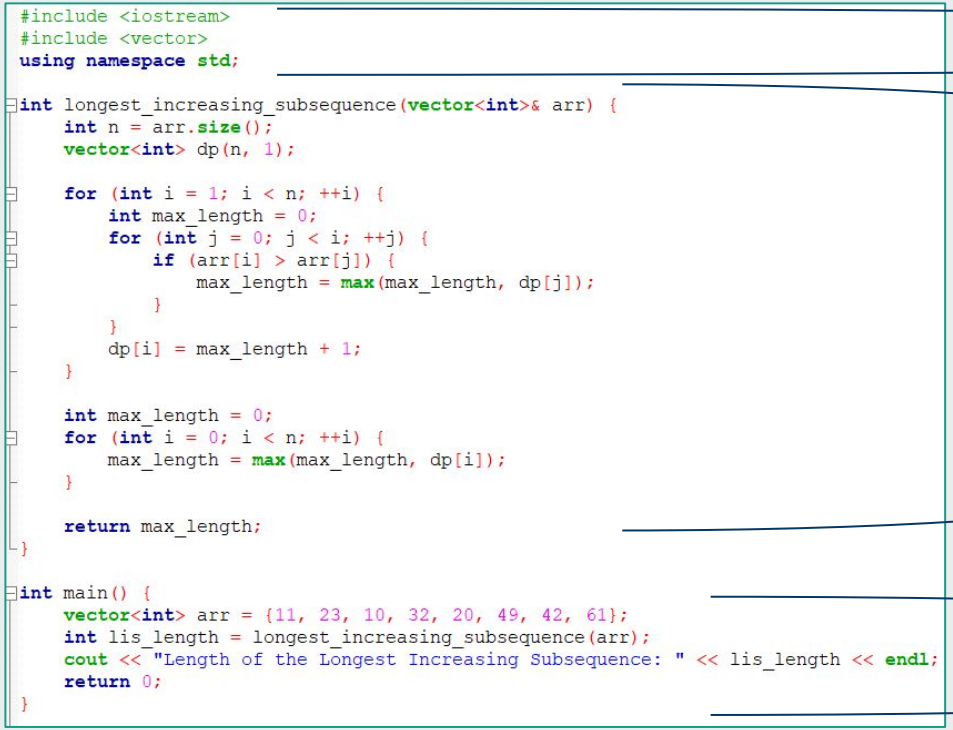
Return maximum length

Implementation

```cpp
#include <iostream>
#include <vector>
using namespace std;

int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}

int main() {
    vector<int> arr = {11, 23, 10, 32, 20, 49, 42, 61};
    int lis_length = longest_increasing_subsequence(arr);
    cout << "Length of the Longest Increasing Subsequence: " << lis_length << endl;
    return 0;
}
```

Time Complexity Analysis

Total: O(n^2)



```cpp
#include <iostream>
#include <vector>
using namespace std;

int longest_increasing_subsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, 1);

    for (int i = 1; i < n; ++i) {
        int max_length = 0;
        for (int j = 0; j < i; ++j) {
            if (arr[i] > arr[j]) {
                max_length = max(max_length, dp[j]);
            }
        }
        dp[i] = max_length + 1;
    }

    int max_length = 0;
    for (int i = 0; i < n; ++i) {
        max_length = max(max_length, dp[i]);
    }

    return max_length;
}

int main() {
    vector<int> arr = {11, 23, 10, 32, 20, 49, 42, 61};
    int lis_length = longest_increasing_subsequence(arr);
    cout << "Length of the Longest Increasing Subsequence: " << lis_length << endl;
    return 0;
}
```

O(1)

O(n^2)

O(n^2)

To Improve the overall time complexity we have another algorithm called **patience sorting**(optimized dp approach)**.**
O(n^2)------>O(nlogn).

Patience Sorting Concepts

This technique basically uses binary search and essentially breaks down the problem of finding the LIS into a visual and intuitive process of building piles. The length of the LIS is equivalent to the number of piles formed, and the actual LIS can be reconstructed by tracing back through the piles.

N.B. While Patience Sort is a helpful conceptual tool for understanding the LIS problem, it might not be the most efficient algorithm for finding the LIS, especially for **larger input** sequences. More efficient algorithms like dynamic programming (as shown in our initial code) are commonly used to solve the Longest Increasing Subsequence problem in practice.

Algorithm

- Start with an empty array to represent piles. The piles will store elements forming increasing subsequences.
- Iterate through the elements of the input array.
- For each element, find the pile where the element can be placed using binary search on the piles array.
- If the element should be placed at the end of the piles, create a new pile with that element.
- If the element should be placed in between existing piles, update the existing pile with the element.
- Continue this process for all elements in the input array.
- The number of piles formed represents the length of the LIS.
- The elements in the piles, in the order they appear, form the Longest Increasing Subsequence.

Simulation

Element (arr[i])	Current Piles	Explanation
11	[11]	Create a new pile since piles is empty.
23	[11,23]	Create a new pile since 23 > 11.
10	[10,23]	Replace 11 with 10 as 10 < 11 and 10 < 23.
32	[10,23,32]	Create a new pile since 32 > 23.
20	[10,20,32]	Replace 23 with 20 as 20 < 23 and 20 < 32.
49	[10,20,32,49]	Create a new pile since 49 > 32.
42	[10,20,32,42]	Replace 49 with 42 as 42 < 49.
61	[10,20,32,42,61]	Create a new pile since 61 > 42.

Implementation

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int longest_increasing_subsequence(vector<int>& arr) {
    vector<int> piles;
    for (int num : arr) {
        auto it = lower_bound(piles.begin(), piles.end(), num);
        if (it == piles.end()) {
            piles.push_back(num);
        } else {
            *it = num;
        }
    }
    return piles.size();
}

int main() {
    vector<int> arr = {11, 23, 10, 32, 20, 49, 42, 61};
    int lis_length = longest_increasing_subsequence(arr);
    cout << "Length of the Longest Increasing Subsequence: " << lis_length << endl;
    return 0;
}
```

Time Complexity Analysis

Total: O(nlogn)

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int longest_increasing_subsequence(vector<int>& arr) {
    vector<int> piles;
    for (int num : arr) {
        auto it = lower_bound(piles.begin(), piles.end(), num);
        if (it == piles.end()) {
            piles.push_back(num);
        } else {
            *it = num;
        }
    }
    return piles.size();
}

int main() {
    vector<int> arr = {11, 23, 10, 32, 20, 49, 42, 61};
    int lis_length = longest_increasing_subsequence(arr);
    cout << "Length of the Longest Increasing Subsequence: " << lis_length << endl;
    return 0;
}
```

O(1)

O(n*logn)

O(n*logn)

THANK YOU

Contributions

University ID	Full Name	Topics
011201262	Azizul Islam Nayem	DP-> Algorithm, Simulation, Codes,Time Complexity.
011192136	Md.Aman Ullah Faisal	DP-> Definition, Usage of LIS, Problem Description.
011181029	Sadman Rafid	Patience Sorting-> Concepts, Algorithm.
011201269	Nayeem Muhammad Al Farabi Sikder	Patience Sorting-> Codes, Time Complexity.
011201239	Md Asad Jaman	DP-> Solving Technique, Simulation(Patience sorting).