Experiment-7A

Student Name: Deepanshu UID: 22BCS16443

Branch:BE-CSE Section/Group: NTPP 602-A

Semester:6TH Date of Performance:16/03/25

Subject Name: AP Lab-2 Subject Code: 22CSH-352

1. TITLE:

Maximum Units on a truck.

2. AIM:

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxes], numberOfUnitsPerBox].

3. Algorithm

- Sort boxTypes in descending order based on numberOfUnitsPerBoxiCompute the depth of the left subtree.
- o Iterate through boxTypes, adding as many boxes as possible to the truck until it is full.
- o **Keep track** of the total units loaded and return the sum).

Implemetation/Code

```
class Solution {
public:
int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
int ans = 0;
ranges::sort(boxTypes, ranges::greater{},
[](const vector<int>& boxType) { return boxType[1]; });

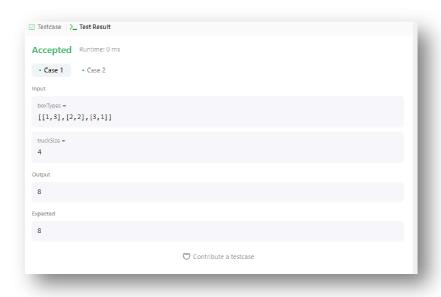
for (const vector<int>& boxType : boxTypes) {
   const int boxes = boxType[0];
   const int units = boxType[1];
   if (boxes >= truckSize)
   return ans + truckSize * units;
   ans += boxes * units;
   truckSize -= boxes;
}
```

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

```
return ans;
}
;;
```

4. Output:



Time Complexity: O(NlogN)

Space Complexity: O(n)

Learning Outcomes:-

- O Sorting and selecting the most valuable boxes first maximizes efficiency.
- O Sorting first ensures an optimal selection process in O(Nlog N)O(N \log N)O(N \log N)O(N \log N) ime.

Experiment - 7B

Student Name: Deepanshu UID: 22BCS16443

Branch:BE-CSE Section/Group: NTPP- 602(A)

Semester:6TH Date of Performance:16/03/25

Subject Name: AP Lab-2 Subject Code: 22CSH-352

1. TITLE:

Maximum Score From Removing Substrings.

2. AIM:

You are given a string s and two integers x and y. You can perform two types of operations any number of times.

3. Algorithm

- Iterate through the string and greedily remove high-scoring substrings first (e.g., "ab" before "ba").
- Keep track of the total score while modifying the string dynamically.
- Continue the process until no more valid substrings remain, then return the total score.

4. Implementation/Code:

```
class Solution {
public:
int maximumGain(string s, int x, int y) {
return x > y? gain(s, "ab", x, "ba", y): gain(s, "ba", y, "ab", x);
private:
int gain(const string& s, const string& sub1, int point1, const string& sub2,
int point2) {
int points = 0;
vector<char> stack1;
vector<char> stack2;
for (const char c : s)
if (!stack1.empty() && stack1.back() == sub1[0] && c == sub1[1]) {
stack1.pop back();
points += point1;
} else {
stack1.push_back(c);
```

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

```
for (const char c : stack1)
if (!stack2.empty() && stack2.back() == sub2[0] && c == sub2[1]) {
    stack2.pop_back();
    points += point2;
} else {
    stack2.push_back(c);
}
return points;
};
```

5. Output:

- **6.** Time Complexity : O(N)
- 7. Space Complexity : O(N)

8. Learning Outcomes:-

- O Prioritizing high-scoring removals first leads to an optimal solution.
- o Efficiently modifying a string while tracking scores is key to solving substring removal problems.

