Experiment-3

Student Name: Anshuman Singh **UID: 20BCS2665**
Branch: CSE **Section/Group: 902/A**
Semester: 6th **Date of Performance: 23-02-2023**
Subject Name: Competitive Coding II **Subject Code: 20CSP-351**

Aim: **To demonstrate the concept of Heap model.**

Problem1: Last Stone Weight

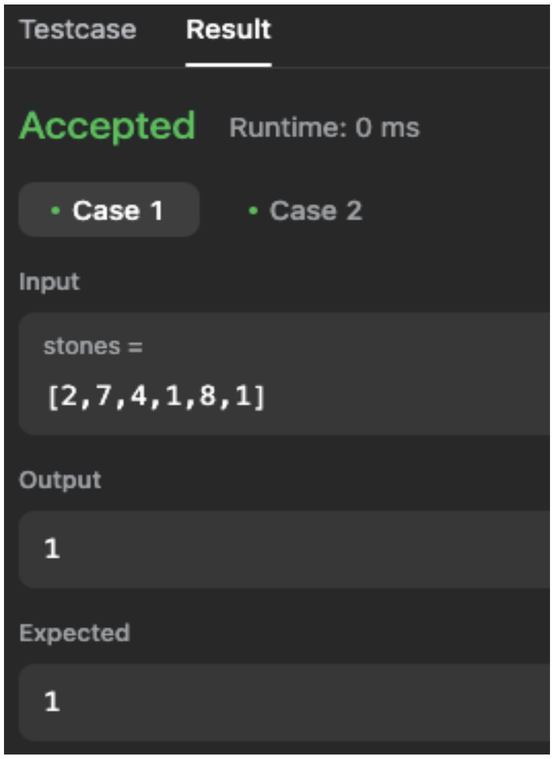
You are given an array of integers stones where stones[i] is the weight of the ith stone.
We are playing a game with the stones. On each turn, we choose the heaviest two stones and smash them together. Suppose the heaviest two stones have weights x and y with x <= y. The result of this smash is:
If x == y, both stones are destroyed, and
If x != y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x.
At the end of the game, there is at most one stone left.
Return the weight of the last remaining stone. If there are no stones left, return 0.
Example 1:
Input: stones = [2,7,4,1,8,1]
Output: 1
Explanation:
We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,
we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,
we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of the last stone.
Example 2:
Input: stones = [1]
Output: 1

Constraints:
1 <= stones.length <= 30
1 <= stones[i] <= 1000

Code:-

```
class Solution {
public int lastStoneWeight(int[] stones) {
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Comparator.reverseOrder());
for (int stone : stones) {
maxHeap.add(stone);
}
while (maxHeap.size() != 1) {
int y = maxHeap.remove();
int x = maxHeap.remove();
if (x == y) maxHeap.add(0);
if (x != y) maxHeap.add(y - x);
}
return maxHeap.peek();
}
}
```

Output:-



Accepted Runtime: 0 ms

Input

```
stones =
[2,7,4,1,8,1]
```

Output

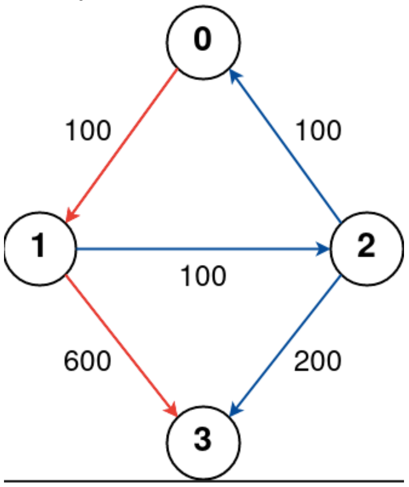
```
1
```

Expected

```
1
```

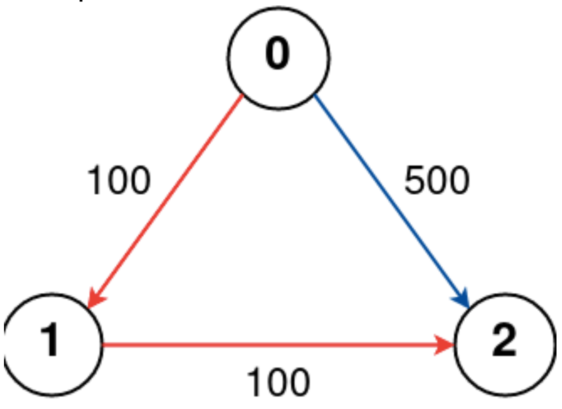
Probem2: Cheapest Flights Within K Stops

There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromi, toi, pricei] indicates that there is a flight from city fromi to city toi with cost pricei.
You are also given three integers src, dst, and k, return the cheapest price from src to dst with at most k stops. If there is no such route, return -1.
Example 1:



Input: n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1
Output: 700
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700.
Note that the path through cities [0,1,2,3] is cheaper but is invalid because it uses 2 stops.

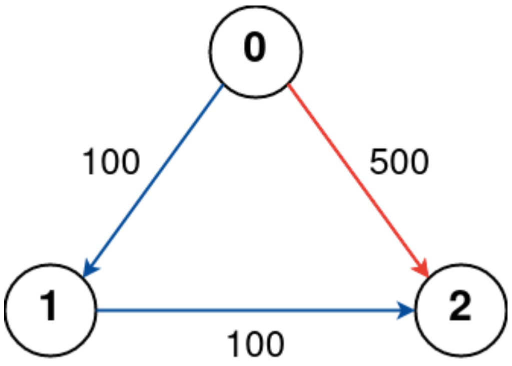
Example 2:



Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1
Output: 200

Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 = 200.

Example 3:



Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 0
Output: 500
Explanation:
The graph is shown above.
The optimal path with no stops from city 0 to 2 is marked in red and has cost 500.

Constraints:

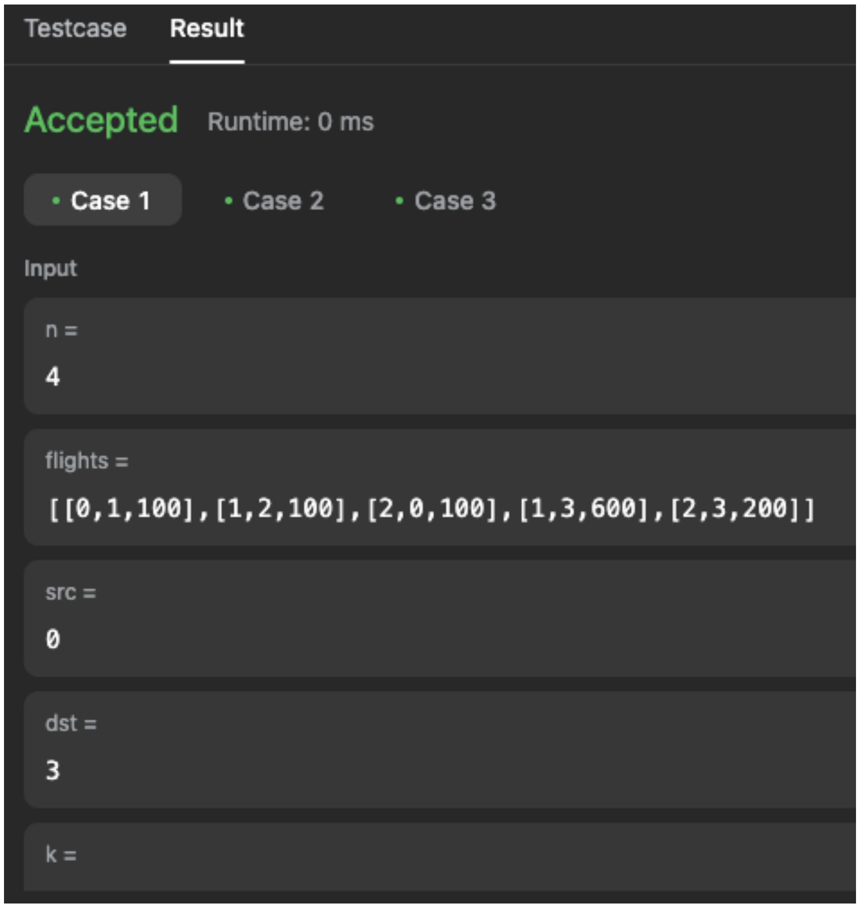
1 <= n <= 100
0 <= flights.length <= (n * (n - 1) / 2)
flights[i].length == 3
0 <= fromi, toi < n
fromi != toi
1 <= pricei <= 104
There will not be any multiple flights between two cities.
0 <= src, dst, k < n
src != dst

Code:-

```
import java.util.*;
class Solution {
```

```
public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {
int prev[]=new int[n];
Arrays.fill(prev,Integer.MAX_VALUE);
prev[src]=0;
for(int i=0;i<=k;i++){
int cur[]=new int[n];
for(int j=0;j<n;j++) cur[j]=prev[j];
for(int e[]:flights){
int u=e[0],v=e[1],wt=e[2];
if(prev[u]!=Integer.MAX_VALUE&&prev[u]+wt<cur[v]){
cur[v]=prev[u]+wt;
}
}
for(int j=0;j<n;j++) prev[j]=cur[j];
}
return prev[dst]==Integer.MAX_VALUE?-1:prev[dst];
}
}
```

Output:-



Accepted Runtime: 0 ms

Input

```
n =
4
flights =
[[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]]
src =
0
dst =
3
k =
```