ASSIGNMENT 2

CODE:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

// structure and value for Item
struct Item
{
        float weight;
        int value;
};

// tree
struct Node
{

        int level;
    int profit;
    int bound;
        float weight;
};

// Comparison function to sort Item
bool cmp(Item a, Item b)
{
        double r1 = (double)a.value / a.weight;
        double r2 = (double)b.value / b.weight;
```

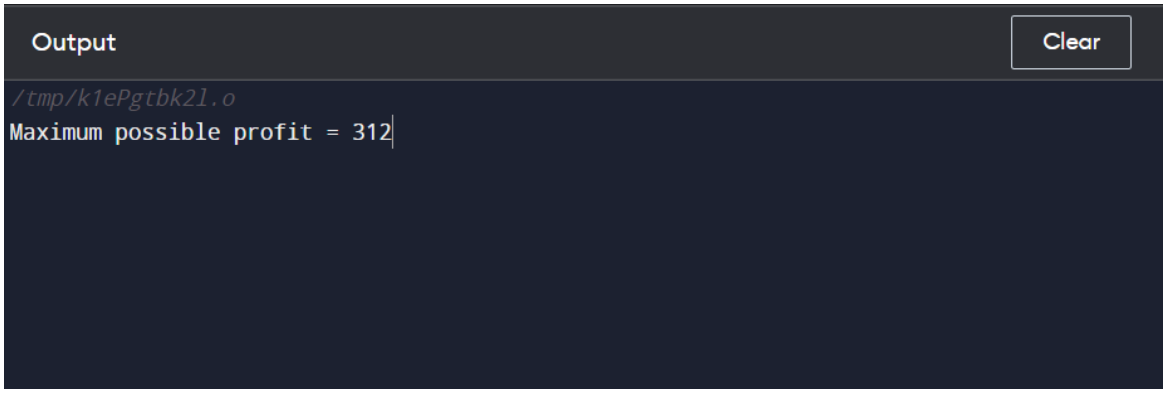
```
	return r1 > r2;
}

// function to find an upper bound on maximum profit
int bound(Node u, int n, int W, Item arr[])
{

	if (u.weight >= W)
		return 0;

	int profit_bound = u.profit;


	int j = u.level + 1;
	int totweight = u.weight;


	while ((j < n) && (totweight + arr[j].weight <= W))
	{
		totweight += arr[j].weight;
		profit_bound += arr[j].value;
		j++;
	}


	if (j < n)
		profit_bound += (W - totweight) * arr[j].value/arr[j].weight;

	return profit_bound;
```

```
}


// Returns maximum profit we can get with capacity W
int knapsack(int W, Item arr[], int n)
{

        sort(arr, arr + n, cmp);


        queue<Node> Q;
        Node u, v;


        u.level = -1;
        u.profit = u.weight = 0;
        Q.push(u);


        int maxProfit = 0;
        while (!Q.empty())
        {

                u = Q.front();
                Q.pop();


                if (u.level == -1)
                        v.level = 0;
```

```
if (u.level == n-1)
        continue;

v.level = u.level + 1;

v.weight = u.weight + arr[v.level].weight;
v.profit = u.profit + arr[v.level].value;

if (v.weight <= W && v.profit > maxProfit)
        maxProfit = v.profit;

v.bound = bound(v, n, W, arr);

if (v.bound > maxProfit)
        Q.push(v);

v.weight = u.weight;
v.profit = u.profit;
v.bound = bound(v, n, W, arr);
if (v.bound > maxProfit)
        Q.push(v);
```

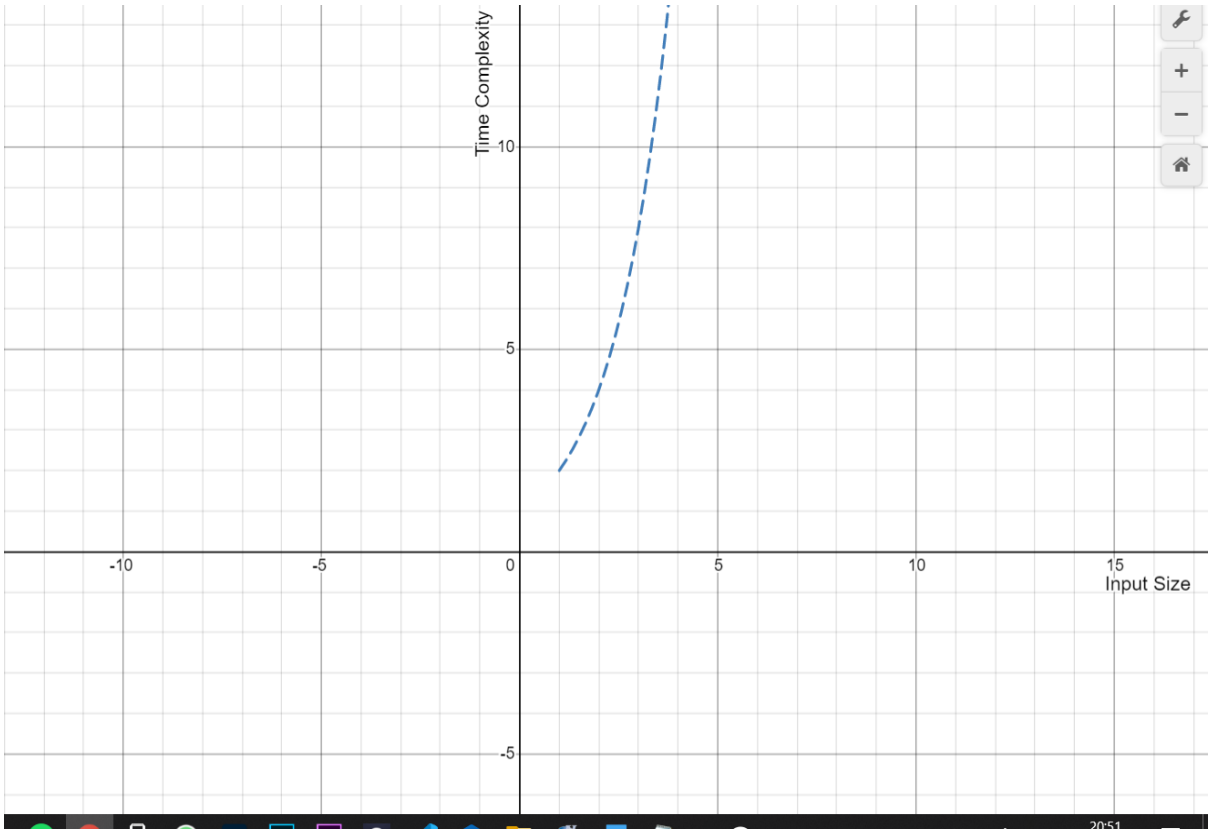
```
        }

        return maxProfit;
}

// testing above function
int main()
{
        int W = 18;
        Item arr[] = {{7, 65}, {3.14, 48}, {2.67, 100},
                                {7, 89}, {5, 75}};
        int n = sizeof(arr) / sizeof(arr[0]);

        cout << "Maximum possible profit = "
                << knapsack(W, arr, n);

        return 0;
}
```

OUTPUT:



TIME COMPLEXITY GRAPHS:

GENERAL: typically exponential in terms of time complexity.



Function wise complexity:

Function Name	Complexity
cmp	1
bound	5
knapsack	8
main	1