Given the preference lists for five interns and five hospitals, suppose the Gale-Shapley algorithm produces the following matching 'M'.

M = { (I_1, H_1), (I_2, H_3), (I_3, H_2), (I_4, H_5), (I_5, H_4) }

Someone claims that (I_1, H_3) is an instability in M. Without looking at the preference lists, how will you **prove** them wrong?

You had to show that the output produced by Gale-Shapley algorithm is stable. The proof was done in class.

Consider the maximum subarray sum problem given to you in a home activity.

The description of the maximum subarray sum problem is as follows:

You are given an array `P[0 ... n-1]` of `n` integers. These integers can be both positive and negative. Let `sum(i,j)` denote the sum of integers `P[i … j]`, where `i≤j`.

(i.e. $sum(i,j) = \sum_{k=i}^{j} P[k]$)

You have to find indices max_`i` and max_`j` such that:

`maxSum = sum(max_i,max_j) = max(sum(i,j) | 0≤i≤j<n)`

You may assume that the 'n' integers are a combination of both positive and negative integers (i.e., they will not be all negative nor will they be all positive integers).

Your friend has written the following code to find `maxSum` in `O(n)` time, however, the statements for determining `max_i` and `max_j` are missing. **You have to fill in the missing statements in the code.** You can assume that initializing of 'n' and array `P[]` is already done. You may declare additional variables, if needed.

Write your answer **clearly** inside the following box.

```
int currentSum = 0, maxSum = 0;
int index, max_i = 0, max_j = 0;
int curr_i = 0;

for (index=0; index<n; index++) {
      currentSum += P[index];



      if (currentSum < 0) {
        currentSum = 0;
       curr_i = index + 1;

      }
      if (maxSum < currentSum ) {
        maxSum = currentSum;
        max_j = index;
        max_i = curr_i;


      }
}
```

Suppose you have a single processor and there are a set of ‘n’ jobs that need to run on it. The jobs can only run one at a time. Each job has a weight associated with it that indicates its priority. Let the jobs be J_1, J_2, … J_n, which require time t_1, t_2 .. t_n respectively to run. Their weights are w_1, w_2, … w_n. Suppose job J_a runs first, followed by job J_b, then completion time for J_a is $C_a = t_a$ and completion time for J_b is $C_b = t_a+t_b$. We want to find how to schedule the ‘n’ jobs such that their weighted sum of completion times is minimized. Suppose the optimal schedule of jobs is given by S_1, S_2, … S_n then you have to **minimize** $\sum_{i=1}^{i=n} w_{si} . C_{si}$

Write your algorithm in clear descriptive steps as in Q1 above (do NOT write C code).

The optimal schedule is obtained by sorting the jobs in ascending order of the ratios t_i/w_i

Running time of your algorithm in Big-Oh notation __`O(nlog(n))`_________

Consider the assignment question on boomerangs.

What is its recurrence relation ________**4T(n/2) + c**___________________________________

What is the time complexity of this recurrence in Big-Oh notation _________**O(n^2)**_______

Suppose you have coded 'n' programs in the last year and you want to take backup of your hard work on a USB drive. Let's call the programs P_1, P_2, … P_n. Program P_i requires k_i bytes of space. The capacity of the USB drive is C bytes, where

$$C < \sum_{i=1}^{n} k_i$$

Notice C is less than the total space required, so you can't save all the 'n' programs.

Can you design an efficient and optimal greedy algorithm to **maximize the space utilization** of the drive?

If YES, then **describe** your algorithm. If NO, then explain why not.
YES / NO ___**NO**____________

This is similar to the knapsack problem.