Assignment 2

Question 1. [10 MARKS]

In the following let $A = \{a_1, ..., a_n\}$ be a set of n items where item a_i has value v_i and capacity c_i. Suppose you have a bag with maximum capacity C. We will develop a recurrence relation to find the maximum value obtained by a subset of A whose capacity does not exceed C. *Note: this is the classic **Knapsack Problem**. Many solutions on-line use linear programming or dynamic programming, but this is **not** what we are asking you to do here; you do not need anything other than the material you learned in class to answer this question.*

Part (1) [2 MARKS]

Let $T(i, c)$ be the maximum value obtained after considering the first i items with total capacity c. Come up with an equation for $T(0, c)$ and $T(1, c)$ i.e. what is the maximum value obtained if we considered none of the items? Only consider the first item?

SOLUTION. $T(0, c) = 0$ since, regardless of how much space is remaining you get no value if you consider none of the items. $T(1, c)$ is the following:

$$T(1, c) = \begin{cases} v_1 & \text{if } c_1 \leq c \\ 0 & \text{otherwise} \end{cases}$$

Part (2) [3 MARKS]

Suppose we know $T(i, c)$, the maximum value obtained after considering the first i items with total capacity c. Find an equation for $T(i + 1, c)$ in-terms of the values $T(i, 0), T(i, 1), \ldots, T(i, c)$. Use this equation along with part one to come up with a general recurrence relation for $T(k, c)$. How would we find the maximum value obtained by a subset of A whose capacity does not exceed C?

SOLUTION. $T(i, c)$ is the following:

$$T(i, c) = \begin{cases} \max\{v_i + T(i-1, c-c_i), T(i-1, c)\} & \text{if } c_i \leq c \\ T(i-1, c) & \text{otherwise} \end{cases}$$

The general recurrence relation for $T(k, c)$ is:

$$T(k, c) = \begin{cases} 0 & \text{if } k = 0 \\ \max\{v_k + T(k-1, c-c_k), T(k-1, c)\} & \text{if } c_k \leq c \\ T(k-1, c) & \text{otherwise} \end{cases}$$

To find the maximum value obtained by a subset of A whose capacity does not exceed C, we evaluate $T(n, C)$.

Part (3) [5 MARKS]

Suppose we have two bags with capacities C_1 and C_2 respectively. Let the total profit be the sum of the values of all items in both bags. Find the maximum total profit subject to the capacity constraints as we did for the one bag case above.

SOLUTION. A valid recursion is as follows. Let $T(i, c, c')$ be the total profit obtained after considering the first i items with capacity c remaining in bag 1 and capacity c' remaining in bag 2. As before $T(0, c, c') = 0$. At intermediate step k, let $u_0 = T(k-1, c, c')$,

$$u_1 = \begin{cases} v_k + T(k-1, c-c_k, c') & \text{if } c_k \leq c \\ 0 & \text{otherwise} \end{cases}$$

and

$$u_2 = \begin{cases} v_k + T(k-1, c, c'-c_k) & \text{if } c_k \leq c' \\ 0 & \text{otherwise} \end{cases}$$

Then define $T(k, c, c')$ as

$$T(k, c, c') = \begin{cases} 0 & \text{if } k = 0 \\ \max\{u_0, u_1, u_2\} & \text{otherwise} \end{cases}$$

Observe that the default value of $T(k, c, c')$ is u_0 when $k > 0$ is u_0 so it is valid to set u_1 and u_2 to zero if the item does not fit. As before to find the maximum total profit we evaluate $T(n, C_1, C_2)$.

Question 2. [11 MARKS]

In the following, consider an $m \times n$ grid as shown in Figure 1. The bottom left point and top right points are $p_{1,1}$ and $p_{m,n}$ respectively. All other points are indexed in the standard way.

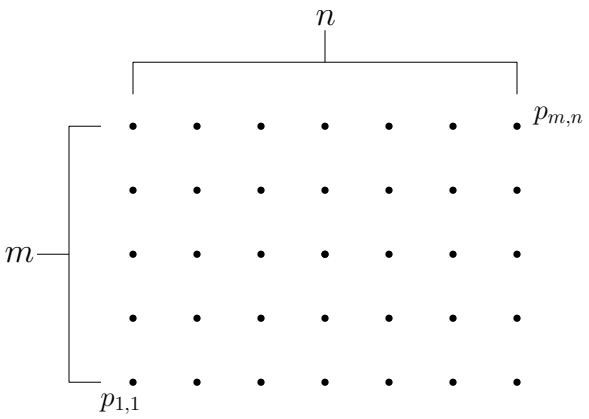

Figure 1: The grid of points we will consider for this problem.

Part (1) [6 MARKS]

Find a recurrence relation for the number of axis aligned rectangles in an $m \times n$ grid. See Figure 2 for an example where $m = 3$ and $n = 3$. Explain your process.

SOLUTION. Let $R(m, n)$ be the number of rectangle in an $m \times n$ grid. The recurrence relation of R is as follows:

$$R(m, n) = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ R(m, n-1) + \frac{m(m-1)(n-1)}{2} & \text{otherwise} \end{cases}$$

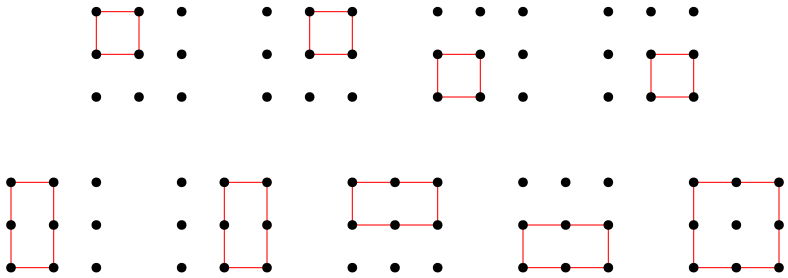

Figure 2: Example with $m = n = 3$. There are nine possible axis aligned rectangles possible.

We reason about this recurrence relation as follows: hold one of the dimensions of the grid constant. Since m and n are interchangeable, without loss of generality (w.l.o.g), say m is constant. This means that the height of the grid is fixed. We divide the $m \times n$ grid into the first $n-1$ columns and the n^{th} column. See Figure 3.

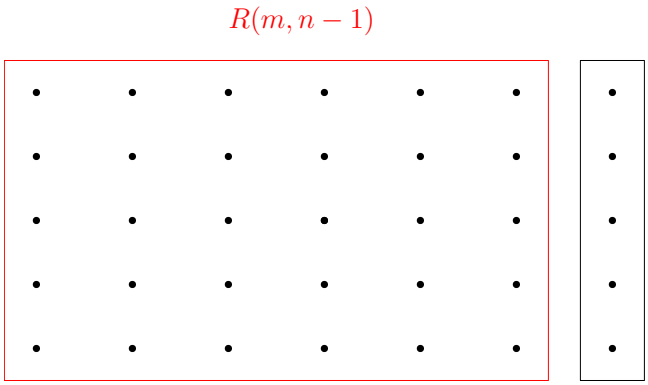

Figure 3: Division of an $m \times n$ grid into the first $n-1$ columns and the last column.

The first $n-1$ columns contain $R(m, n-1)$ axis aligned rectangles. It remains to count the number of rectangles with end-points in the n^{th} column. Observe a rectangle can be specified by picking two points which are not in the same row or column. With this in mind, remark that any $p_{i,j}$ with $2 \leq i \leq m$ and $1 \leq j \leq n-1$ forms an axis aligned rectangle with $p_{1,n}$. See Figure 4. Similarly any $p_{i,j}$ with $1 \leq i \leq m$, $i \neq 2$, and $1 \leq j \leq n-1$ forms an axis aligned rectangle with $p_{2,n}$. Generally all points $p_{i,j}$ with $1 \leq i \leq m$, $i \neq k$, and $1 \leq j \leq n-1$ forms an axis aligned rectangle with $p_{k,n}$ for $k = 1, 2, ..., m$.

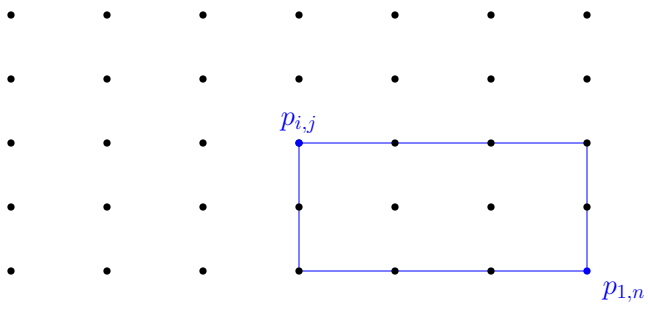

Figure 4: Forming rectangles with end-points in the n^{th} column.

There are $m(m-1)(n-1)$ rectangles counted in this way. Note however that each rectangle has been counted twice because there are two pairs of points which specify the same rectangle. Thus the total number of distinct axis aligned rectangles with end-points in the n^{th} column is $\frac{m(m-1)(n-1)}{2}$.

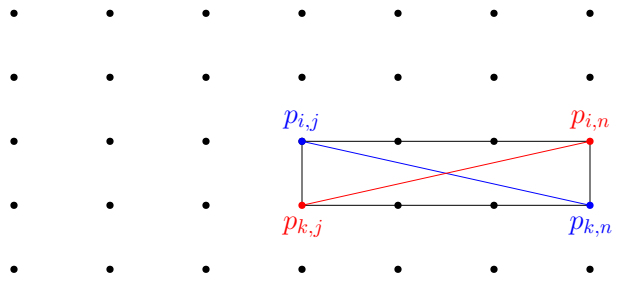

Figure 5: Each rectangle is counted twice.

Part (2) [1 MARK]

Hypothesize a closed form for your recurrence relation. *Hint: check to see if your closed form is correct in the special case when $m = n$ using the on-line encyclopedia of integer sequences.*

SOLUTION. The closed form of the recursion is

$$R(m,n) = \frac{m(m-1)n(n-1)}{4}$$

for all numbers $m, n \in \mathbb{N}$. When $m = n$ we obtain sequence [A000357] on the OEIS.

Part (3) [4 MARKS]

Prove your closed form using induction.

Proof. Let the predicate be:

$$P\left((m,n)\right) \coloneqq R(m,n) = \frac{m(m-1)n(n-1)}{4}.$$

We want to show that $P((m,n))$ holds for all $m, n \in \mathbb{N}$. Do this by fixing m and defining a new predicate

$$Q(n) \coloneqq R(m,n) = \frac{m(m-1)n(n-1)}{4}.$$

We show that $Q(n)$ holds by induction. In the base case $n = 0$. By the recursive definition of $R(m,0) = 0$. A grid with no columns cannot form any rectangles. Thus the base case holds.

Let $k \in \mathbb{N}$ and $k \geq 1$. Suppose for the inductive step $Q(k)$ holds. We want to show that $Q(k+1)$ holds.

$$\begin{aligned}
R(m,k+1) &= R(m,k) + \frac{m(m-1)k}{2} && \text{Recursive Def.} \\
&= \frac{m(m-1)k(k-1)}{4} + \frac{m(m-1)k}{2} && \text{I.H.} \\
&= \frac{m(m-1)k}{4}\left(k-1+2\right) \\
&= \frac{m(m-1)k(k+1)}{4}
\end{aligned}$$

Thus by the principle of mathematical induction, $Q(n)$ holds for all $n \in \mathbb{N}$. Since m and n are interchangeable, the same proof works when we fix n. Thus $P((m, n))$ holds for all pairs $m, n \in \mathbb{N}$. □

Question 3. [10 MARKS]

An array is said to be ***rotated*** by one space if all its elements are moved ahead circularly, that is every element is moved to the next position and the last element is moved to the first position. For instance the array $[1, 2, 3, 4, 5]$ rotated by one space gives $[5, 1, 2, 3, 4]$. Rotating an array by r spaces simply means rotating it by one space r times. So the array $[1, 2, 3, 4, 5]$ rotated 2 spaces will give $[4, 5, 1, 2, 3]$ and rotated 5 spaces will give $[1, 2, 3, 4, 5]$ back again.

The input is an array A of n ***distinct*** elements which is sorted in ascending order and then rotated a certain number of spaces. (The number of spaces the array is rotated may be zero as well.)
You have to devise an algorithm **minimum**, which finds the minimum element of the array.

Part (1) [2 MARKS]

Devise an $\mathcal{O}(n)$ brute-force algorithm for **minimum** in Python notation. Give an informal explanation of why the time complexity is $\mathcal{O}(n)$.

SOLUTION.

```
def minimum(A):
   min = A[0]
   for i in range (1, len(A)):
      if min < A[i]:
         min = A[i]
   return min
```

The algorithm iterates through the array maintaining the minimum element so far. It compares each element with `min`, each comparison takes constant time, and the array has n elements. The initial assignments and return are constant time operations. Thus the total time complexity of the algorithm is $\mathcal{O}(n)$.

Part (2) [3 MARKS]

Devise a divide-and-conquer algorithm for **minimum** which performs better than the brute-force version. (Be careful of the edge and base cases.)

SOLUTION.

```
def minimum(A, begin, end):
    if A[begin] <= A[end]:
        return A[begin]
    mid = (begin+end)//2
    if A[mid] < A[begin]:
        return minimum(A, begin, mid)
    else:
        return minimum(A, mid+1, end)
```

The first `if` condition handles the case of having a single element array, as well as the case in which the array is not rotated at all.

Part (3) [1 MARK]

Give recurrence relation $T(n)$ for the worst-case time complexity for the divide-and-conquer algorithm.

SOLUTION. Both comparisons and computing `mid` takes constant time. In the worst case the recursive call will be on the half with size $\lceil n/2 \rceil$. Assume that all the constant-time operations (comparisons, computing `mid`, recursive call, and return) take a total of c_1 operations. For the base case of the single element array, assume that the comparison and return statements take a total of c_0 time.

Then the time complexity can be given by,

$$T(n) = \begin{cases} c_0 & \text{for } n = 1, \\ c_1 + T(\lceil n/2 \rceil) & \text{for } n > 1 \end{cases}$$

Part (4) [4 MARKS]

Use repeated substitution to get the closed form for $T(n)$. Prove using induction that the closed form indeed satisfies the recurrence relation.

SOLUTION.

$$\begin{aligned} T(n) &= c_1 + T(\lceil n/2 \rceil) \\ &= c_1 + c_1 + T(\lceil n/4 \rceil) \\ &= c_1 + c_1 + c_1 + T(\lceil n/8 \rceil) \\ &\vdots \\ &= kc_1 + T(n/2^k) \end{aligned}$$

For having $\lceil n/2^k \rceil = 1$, we must have $n \leq 2^k$ or $k \geq \log_2 n$. Thus for $k = \lceil \log_2 n \rceil$ we have,

$$\begin{aligned} T(n) &= c_1 \lceil \log_2 n \rceil + T(1) \\ T(n) &= c_0 + c_1 \lceil \log_2 n \rceil \end{aligned}$$

Proof. Let $T_c(n) = c_0 + c_1 \lceil \log_2 n \rceil$ denote the closed form of the time complexity

For $n = 1$, $T_c(1) = c_0 + \lceil \log_2 n \rceil = c_0 = T(1)$ (**Induction Basis**).

For some $n > 1$, assume for all $1 \leq i < n$, $T(i) = T_c(i)$ (**Induction Hypothesis**).

Since $n > 1$, $1 \leq \lceil n/2 \rceil < n$. So by the induction hypothesis, $T(\lceil n/2 \rceil) = T_c(\lceil n/2 \rceil)$. Thus,

$$\begin{aligned} T(n) &= c_1 + T(\lceil n/2 \rceil) \\ &= c_1 + c_0 + c_1 \lceil \log_2 \lceil n/2 \rceil \rceil \\ &= c_1 + c_0 + c_1 (\lceil \log_2 n \rceil - 1) \\ &= c_0 + c_1 \lceil \log_2 n \rceil \\ &= T_c(n) \end{aligned}$$

Thus, by induction for all $n \geq 1$, we have $T(n) = T_c(n)$. □

Question 4. [9 MARKS]

Apply the Master Theorem to derive an asymptotic upper bound for each of the following recurrences.

Part (1) [3 MARKS]

$$T(n) = \begin{cases} 4 & n = 1 \\ 4 \cdot T(\lfloor n/2 \rfloor) + n^2 + 8n & n > 1 \end{cases}$$

SOLUTION. Note that $n^2 + 8n = \Theta(n^2)$. Hence, to apply the Master Theorem, we take $a = 4$, $b = 2$ and $k = 2$. Then $\log_b a = \log_2 4 = 2 = k$ so $T(n) = O(n^k \cdot \log n) = O(n^2 \cdot \log n)$.

Part (2) [3 MARKS]

$$T(n) = \begin{cases} 1 & n = 1 \\ 2 \cdot T(\lfloor n/4 \rfloor) + 1 & n > 1 \end{cases}$$

SOLUTION. Take $a = 2$, $b = 4$ and $k = 0$. Then $\log_b a = \log_2 8 = \frac{1}{2} > k$ so $T(n) = O(n^{\log_b a}) = O(\sqrt{n})$.

Part (3) [3 MARKS]

$$T(n) = \begin{cases} 10 & n = 1 \\ 9 \cdot T(\lfloor n/3 \rfloor) + n^3 \cdot \log n & n > 1 \end{cases}$$

SOLUTION. Unfortunately, $n^3 \cdot \log n$ is not of the form $\Theta(n^k)$ for any $k \geq 0$ so we cannot apply the Master Theorem directly. However, since $n^3 \log n \leq n^4$, we can apply the Master Theorem to the following recursion instead.

$$T'(n) = \begin{cases} 10 & n = 1 \\ 9 \cdot T'(\lfloor n/3 \rfloor) + n^4 & n > 1 \end{cases}$$

To do so, we take $a = 9$, $n = 3$ and $k = 4$. Then $\log_b a = \log_3 9 = 2 < k$ so $T'(n) = O(n^k) = O(n^4)$. Now, since $T(n) \leq T'(n)$ for all n, we have that $T'(n) = O(n^4)$ implies $T(n) = O(n^4)$.