PROBLEM SET 4

Zhiqiang Xie 77892769

Problem 1

Python

```python
def findKth(A,B,k):
    ''' Find the k'th element of the two sorted array by Binary Search
    '''
    if len(A)>len(B):
        A,B=B,A
    if not A:
        return B[k]
    if k==len(A)+len(B)-1:
        return max(A[-1],B[-1])

    i=len(A)//2
    j=k-i
    if A[i]>B[j]:
        return findKth(A[:i],B[j:],i)
    elif A[i] < B[j]:
        return findKth(A[i:],B[:j],j)
    else:
        return A[i]

def findMedian(A, B):
    ''' Convert the find median problem into k'th problem
    '''
    l=len(A)+len(B)
    return findKth(A,B,l//2) if l%2==1 else (findKth(A,B,l//2-1)+findKth(A,B,l//2))/2
```

The time complexity $T(n) \in O(log(m+n))$

Problem 2

Python

```python
def minimal(f, left=a, right=b, precision):
    ''' Where a, b should be specified to be the bounds of the interval.
        Here precision is the error we can tolerate
    '''
    while abs(left - right) <= precision:
        left_new = (2*left + right) / 3
        right_new = (left + 2*right) / 3

        if f(left_new) < f(right_new):
            right = right_new
        else:
            left = left_new

    return f((left + right) / 2)
```

The time complexity $T(n) \in O(log(b-a))$

Problem 3

Since the v_i is the unit value of the materials and we can pack materials in fractional part, this problem will be a quite simple greedy one.

- Firstly, sort the materials by their unit values in descending order. This step costs $O(n \log n)$ time.
- Then pack the materials in the order we sorted until we fullfil our package. Since the x_i can be fractional, we can just fullfil our package. This step costs $O(n)$ time.
- The time complexity $T(n) = O(n) + O(n \log n) \in O(n \log n)$

Proof:

- Assume here is an optimal solution to pick materials. We firstly sort the materials packed by unit values in descending order.
- For every unit of materials we pack, we compare it to the corresponding choice in optimal solution. It's guaranteed that the value of our choice is not less than optimal solution, because here is no materials left with higher unit value.
- Therefore, our greedy solution is optimal.

Problem 4

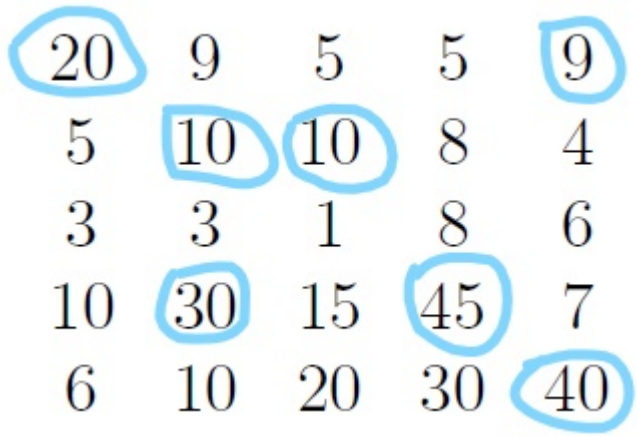
Greedy Algorithm:

- Sort the set of n points $x_1, x_2, \ldots, x_n$ to get a new set $Y = y_1, y_2, \ldots, y_n$ such that $y_1 \leq y_2 \leq \ldots \leq y_n$.
- Then we scan the set started from y_1. Everytime while encountering y_i ($i \in 1, \ldots, n$), we put the closed interval $[y_i, y_i + 1]$ in our optimal solution set S, and remove all the points in Y covered by $[y_i, y_i + 1]$.
- Repeat the above procedure, finally output S while Y becomes empty.

Proof:

- Suppose that there exists an optimal solution S', such that y_1 is covered by $[x', x' + 1] \in S'$, where $x' < y_1$, Since y_1 is the leftmost element of the sorted set, there is no other point lying in $[x', y_1)$.
- If we replace $[x', x' + 1]$ in S, by $[y_1, y_1 + 1]$, we will get another optimal solution.
- Repeating solving the remaining subproblem (repeating comparing following interval), we'll find our greedy strategy lead to optimal.

Problem 5

1. 

2. Let $a = max(A)$ and the index is (i, j), there is no doubt that:
   - $a \geq max\{A_{i-1,j}, A_{i+1,j}, A_{i,j-1}, A_{i,j+1}\}$, therefore a is certainly one of the peak in the matrix.

   Here is a divide and conquer method to find a peak:
   - Treat the middle column as an $m \times 1$ matrix, and find the entry with **max value** and index (i, j) in this submatrix. This step costs $O(m)$ time.
   - Consider the two values $A_{i,j-1}$ and $A_{i,j+1}$. If neighter value is greater than $A_{i,j}$, then return $A_{i,j}$ as a peak. Otherwise, recurse on half of the matrix containing a greater value. This step costs $O(1)$ time and it'll be excuted $O(\log n)$ times at most.
   - The time complexity $T(n) = O(m) * O(1) * O(\log n) \in O(m \log n)$
3. The time complexity depends on the implementation of finding peak in the $m \times 1$ matrix:
   - It can be done in $\log m$ time if we apply the divide and conquer method too.

- $T(n) = O(\log m) * O(1) * O(\log n) \in O(\log m \log n)$

This algorithm is incorrect:

- $A = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 2 & 0 \\ 4 & 0 & 0 \\ 5 & 6 & 0 \end{bmatrix}$ Firstly, we'll choose the $A_{2,2} = 2$ to be the peak in this column.
- Then the submatrix is going to be $\begin{bmatrix} 0 \\ 3 \\ 4 \\ 5 \end{bmatrix}$, thus the entry $A_{4,1}$ is the choice but it's wrong since $A_{4,1} = 5 < 6 = A_{4,2}$.