PROBLEM SET 2**

Zhiqiang Xie 77892769

Problem 1

Raw	First	Next	Last
COW	SEA	TAB	BAR
DOG	TEA	BAR	BIG
SEA	MOB	EAR	BOX
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX	DIG	DIG	EAR
TAB	BIG	BIG	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX	FOX	RUG	TEA

Problem 2

The number in **bold** text is comparing to those in `monospace`, and they'll all be moved.

4	9	2	3	5	7	8	1	6
4	9	2	3	5	7	8	1	6
`4`	**9**	2	3	5	7	8	1	6
`4`	`9`	**2**	3	5	7	8	1	6
2	`4`	`9`	**3**	5	7	8	1	6
2	3	4	`9`	**5**	7	8	1	6
2	3	4	5	`9`	**7**	8	1	6
2	3	4	5	7	`9`	**8**	1	6
`2`	`3`	`4`	`5`	`7`	`8`	`9`	**1**	6
1	2	3	4	5	`7`	`8`	`9`	**6**
1	2	3	4	5	6	7	8	9

Problem 3

Python

```python
def sortStack(stack1):
    """
    :type stack1: List[int]
    :rtype: List[int]
    """
    stack2 = []
    while not stack1.empty():
        tem = stack1.pop()
        while (not stack2.empty()) and tem > stack2.top():
            stack1.push(stack2.pop())
        stack2.push(tem)
    return stack2
```

Problem 4

Python

```python
def findKthSmallest(array, k):
    """
    Revised quick sort algorithm to find the k'th smallest number of an array
    in expected O(n) time
    :type array: List[int]
    :type k: int
    :rtype: int
    For concise, 1 <= k <= len(array) and no duplicate are supposed
    """
    pivot = array[0]
    tem_index = 1
    for i in range(1, len(array)):
        if array[i] < pivot:
            array[i], array[tem_index] = array[tem_index], array[i]
            tem_index += 1
        array[0], array[tem_index-1] = array[tem_index-1], array[0]
    if k == tem_index:
        return pivot
    elif k > tem_index:
        return findKthSmallest(array[tem_index:], k - tem_index)
    else:
        return findKthSmallest(array[:tem_index], k)
```

The expected time complexity $T(n) = \Theta(n) + T(n/2) \in O(n)$

Problem 5

Python

```python
def searchCorTarget(array):
    """
    Revised binary search to find the i, A[i] == i
    :type array: List[int]
    :rtype: int or None if target is not found
    Count the index from 0 instead of 1
    """
    left, right = 0, len(array) - 1
    while left <= right:
        mid = (left+right) // 2
        if array[mid] == mid:
            return mid
        elif array[mid] < mid:
            left = mid + 1
        else:
            right = mid - 1
    return None
```

The time complexity $T(n) = \Theta(1) + T(n/2) \in \Theta(log(n))$

Problem 6

1. As for base case $n = 2$, we can easily show that CURLY-SORT swaps the two elements if they are not sorted.

   Now we assume CURLY-SORT correctly sorts an array with length of k and $1 \le k < n$. Particularly, we set $k = n/3$. Therefore,

   - We firstly correctly sort $A[i : j - k]$ which contains approximately the preceding $2n/3$ elements.
   - Then we correctly sort $A[i + k : j]$ which contains approximately the last $2n/3$ elements.
   - Since $k \le (j - i + 1)/3$, we have $(j - k) - (i + k) + 1 = j - i - 2k + 1 >= k = j - (j - k)$

     Therefore, we guarantee that all elements in $A[j - k + 1 : j]$ are larger than anyone in $A[i : j - k]$
   - Finally, we correctly sort $A[i : j - k]$ and we have a sorted $A[i : j]$ with length of n.

   Proved by the principle of induction.

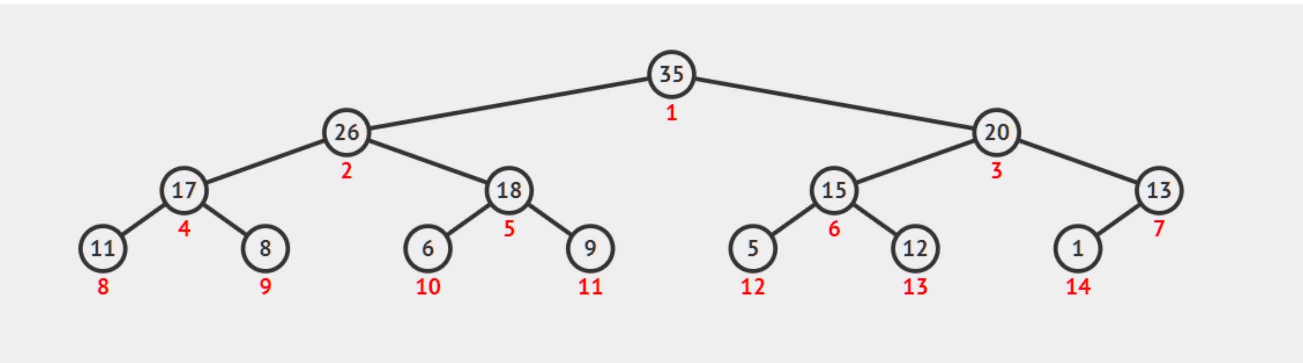
2. $T(n) = 3T(2n/3) + \Theta(1) = \Theta(n^{log_{3/2}3})$ By Master Theorem.

3. As for worst case, $log_{3/2}3 \approx 2.71 > 2$, which means this algorithm is the worst one.

Algorithm	Time Complexity(worst case)
Insertion-sort	$\Theta(n^2)$
Merge-sort	$\Theta(nlog(n))$
Quick-sort	$\Theta(n^2)$
CURLY-SORT	$\Theta(n^{2.71})$

Problem 7

The max binary heap is like this:



Problem 8

Python

```python
class MedianHeap(object):
    """
    A pseudo MedianHeap consist of a maxheap and a minheap.
    The details of maxheap and minheap are omitted for conciseness.
    """

    def __init__(self):
        self.maxheap = []
        self.minheap = []

    def push(self, n):
        """
        The method is designed for streaming data.
        Optimize it for other kinds of interfaces.
        """
        if self.maxheap.size > self.minheap.size:
            if self.maxheap[0] > n:
                self.maxheap[0], n = n, self.maxheap[0]
                self.maxheap.heapify(0) # 0 specifies the node to be heapified
            self.minheap.push(n)
        else:
            if self.minheap.size != 0 and self.minheap[0] < n:
                self.minheap[0], n = n, self.minheap[0]
                self.minheap.heapify(0)
            self.maxheap.push(n)

    def top(self):
        return self.maxheap[0] if maxheap.size != 0 else None

    def pop(self):
        median = self.maxheap.pop()
        if self.maxheap.size < self.minheap.size:
            self.maxheap.push(self.minheap.pop())
        return median
```

The time complexity for wrost case $T(n) = \Theta(1) + \sum_{i=1}^{n} 2\log i + \log n \in \Theta(n \log n)$

Problem 9

1. A Young tableau example (please ignore the bold head, since Markdown doesn't support it):

5	11	14	24
45	47	54	∞
98	∞	∞	∞
∞	∞	∞	∞

2.
    - Since the $Y[1,1]$ is the minimum element in the table, $Y[1,1] = \infty$ indicates there is no element in this table.
    - Since the $Y[m,n]$ is the maximum element in the table, $Y[m,n] < \infty$ indicates there is no vacancy in this table.

Python

```python
class YoungTableaux(object):
    """
    The elements in Young tableaux are stored in the numpy style.
    float("Inf") will be used to represent the inf
    This class counts starting from 0 instead 1
    """

    def __init__(self, m, n, elements=[]):
        """
        Not the optimal initialization for input elements
        """
        self.__tableaux = [[float("Inf")] * (n + 1) for i in range(m + 1)]
        self.__height, self.__width = m, n
        self.__valid = 0
        for i in range(len(elements)):
            self.insert(elements[i], i // n, i % n)

    def get_table(self):
        return [i[:-1] for i in self.__tableaux[:-1]]

    def extract_min(self):
        """
        No checking for the empty case of table
        Return float("Inf") if empty
        """
        table = self.__tableaux  # a local table which is refered to the self.__tableaux
        minimum, table[0][0] = table[0][0], float("Inf")
        i = j = 0
        while i < self.__height and j < self.__width:
            if table[i][j] <= min(table[i + 1][j], table[i][j + 1]):
                break
            if table[i + 1][j] > table[i][j + 1]:
                table[i][j], table[i][j + 1] \
                    = table[i][j + 1], table[i][j]
                j += 1
            else:
                table[i][j], table[i + 1][j] \
                    = table[i + 1][j], table[i][j]
                i += 1
        self.__valid -= 1
        return minimum
```

The time complexity for the worst case of the extracting minimum method
$T(m, n) = \Theta(1) + O(m) + O(n) \in O(m + n)$

```python
    def insert(self, element, i=-2, j=-2):
        """
        No checking for the full case of table
        Overwriting the maximum entry if full
        """
        table = self.__tableaux  # a local table which is refered to the self.__tableaux
        i, j = i % (self.__height + 1), j % (self.__width + 1)
        table[i][j] = element
        while i >= 1 and j >= 1:
            if table[i][j] >= max(table[i - 1][j], table[i][j - 1]):
                break
            if table[i - 1][j] > table[i][j - 1]:
                table[i][j], table[i - 1][j] \
                    = table[i - 1][j], table[i][j]
                i -= 1
            else:
                table[i][j], table[i][j - 1] \
                    = table[i][j - 1], table[i][j]
                j -= 1
        else:
            while i >= 1:
                if table[i][j] >= table[i - 1][j]:
                    break
                table[i][j], table[i - 1][j] \
                    = table[i - 1][j], table[i][j]
                i -= 1
            while j >= 1:
                if table[i][j] >= table[i][j - 1]:
                    break
                table[i][j], table[i][j - 1] \
                    = table[i][j - 1], table[i][j]
                j -= 1
        self.__valid += 1
        return True
```

The time complexity for the worst case of the inserting method
$T(m,n) = \Theta(1) + O(m) + O(n) \in O(m+n)$

```python
    def sort(self):
        """
        All elements will be removed after sorting
        """
        sorted_array = [0] * self.__valid
        for i in range(self.__valid):
            sorted_array[i] = self.extract_min()
        return sorted_array
```

The time complexity for the worst case of the sorting method $T(n) = n^2 * O(n+n) \in O(n^3)$

```python
    def search(self, target):
        i, j = self.__height - 1, 0
        while i >= 0 and j < self.__width:
            if self.__tableaux[i][j] == target:
                return True
            elif self.__tableaux[i][j] > target:
                i -= 1
            else:
                j += 1
        return False
```

The time complexity for the worst case of the searching method
$T(m,n) = \Theta(1) + O(m) + O(n) \in O(m+n)$

```python
if __name__ == "__main__":
    """
    Testing code if you like
    """
    import random
    original = [random.randint(1, 1000) for i in range(100)]
    target = original[0]
    young = YoungTableaux(10, 10, original)
    print("YoungTableaux:",young.get_table())
    print("target got:", young.search(target))
    print("no -1 in the table", not young.search(-1))
    sorted_array = sorted(original)
    print("extract the minimum successfully?",sorted_array[0] == young.extract_min())
    young.insert(sorted_array[0])
    print("Sort successfully?",young.sort() == sorted(original))
```