CSCI 570 - HW 3

1. (i) $T(n) = 4T(n/2) + (n^2)\log n$
a=4, b=2, k=2, p=1
$\log_b a = 2$
->case 3
Therefore: Ø((n^2)*(logn)^2)

(ii) $T(n) = 8T(n/6) + n\log n$
a=8, b=6, k=1, p=1
$\log_b a = \log_3 4 = 1.2$
->case 1
Therefore: Ø(n^($\log_6 8$)))

(iii) T(n) = sqrt(6006)T(n/2) + (n^sqrt(6006))
T(n) = 77.5T(n/2) + (n^77.5)
a=77.5, b=2, k=77.5, p=0
$\log_b a = \log_2 77.5 = 6.3$
->case 3
Therefore: Ø(n^sqrt(6006))

(iv) T(n) = 10T(n/2) + (2^n)
T(n) = 10T(n/2) + logn
a=10, b=2, k=0, p=1
$\log_b a = \log_2 10 = 3.3$
->case 1
Therefore: Ø(n^($\log_2 10$))

(v)T(n) = 2T(sqrt(n)) + logn
Master theorem is not applicable because the equation is not of the form T=aT(n/b)+f(n)
Substitute: m=logn
T(2^m) = 2T(2^(m/2))+m
Substitute: s=2^m
T(s) = 2T(s/2)+logs
a=2, b=2, k=0, p=1
$\log_b a = \log_2 2 = 1$
->case 1
Therefore: Ø(s^$\log_2 2$)
=>Ø(2^m)
=>Ø(2^logn)

(vi)T(n)=T(n/2)-n+10
Master Theorem cannot be applied because f(n) is negative and when increases the value tends to decrease and will be less than 0

(vii)T(n)=(2^n)T(n/2)+n
Master Theorem cannot be applied because a is not a constant

(viii)T(n)=2T(n/4)+(n^0.51)

a=2, b=4, k=0.51, p=0
$\log_b a = \log_4 2 = 0.5$
case 3
Therefore: Ø(n^0.51)

(ix)T(n)=0.5T(n/2)+1/n
a=0.5, b=2, k=-1, p=0
$\log_b a = \log_2 0.5 = -1$
case 2
Therefore: Ø((1/n)logn)

2. Given A array of n numbers
n>2, index i is said to be local minimum of the array A, if it satisfies 1<i<n, A[i-1]>=A[i] and A[i+1]>=A[i]
Proof using Induction
Base case: n=3
A[1:3]
The array consists of 3 numbers, A[1]>=A[2] and A[3]>=A[2]
Therefore A[2] is the local minimum
Case 2: A consists n number of elements and lm be the local minimum
So there exists a local minimum A[1:lm+1] (from the base case)
Consider there exists a local minimum at the array length lm, so there exists a local minimum for A[i:lm+(i-1)], that is consider i=2, A[2:lm+1] with the array length lm
This satisfies the induction hypothesis that there exists a local minimum for every lm sub array, hence by induction it is proved that there exists a local minimum for A[i:lm+(i-1)]

Algorithm:
Step 1: if n==3, then A[2] is local minimum
Step 2: if n> 3, then lm=n/2
Step 3: if A[lm-1]>=A[lm] and A[lm+1]>=A[lm] then A[lm] is local minimum
Step 4: if A[lm-1]<A[lm] then return A[1:lm] else return A[lm:n]

Complexity:
Recurrence Relation=> T(n) = T(n/2)+1
a=1, b=2, k=0, p=0
$\log_b a = \log_2 1 = 0$
case 2
Therefore: O(logn)

Proof of correctness:
By using induction. Assume that the algorithm is correct for n<=k and now consider the input k+1, then lm=[(k+1)/2], from the algorithm we know that it returns correct output for 3 number of elements, for n number of elements, and from the question the given algorithm can find correct result if lm<=k(by using induction hypothesis). This is valid for lm>=2 for all values of m>=3. Therefore by induction the algorithm gives the correct output for k+1 number of elements.
7. Let P be the original array and n be the number of elements of the array
Algorithm:
Step 1: construct the infinite array using the original array and the length of the infinite array is len(infinite_array)>=b

Step 2: count the number of 1's in infinite array up to len(infinite_array)==2n and store in variable count

Step 3: divide the algorithm into 2 parts, i.e. part 1 from index i=0 to i=a-1 and second part i=a to i= len(infinite_array)

Step 4: now use the second sub array and divide the second sub array into two parts from i=a to i%n==0 (first occurrence of this condition, let the index value here be g) and i%n==0 to len(infinite_array)

Step 5: now use a flag variable and increment flag if i%n==0, repeat this step until len(infinite_array)<=b (let the index value here be h), and when this condition fails divide the second sub array from step 4 again into two parts from index value g to index value h and second sub array from index value h to len(infinite_array)

Step 6: divide the second sub array from step 5 into two parts, i.e. from index value i=h to index value i=b and the other sub array from index value i=b+1 to len(infinite_array)

Step 7: using for loop count the number of 1's in the sub arrays i=a to i=g and another sub array i=h to i=b, store this value to final_answer

Step 8: now calculate the number of 1's between i=g to i=h by the formula number_1s=(flag/2)*count

Step 9: now add the count and final_answer to get the total number 1's between a and b

Recurrence Relation:

$T(n) = 5T(n/2)+n$

Master Theorem:

a=5, b=2, k=1, p=0

$\log_b a = \log_2 5 = 2.3$

->case 1

Therefore: $\text{Ø}(n^\wedge(\log_2 5))$

Time Complexity:

O(n)<O(b)

4.let cour[k,a] be the minimum sum of courses for which Erica will get exhausted for completing all the courses before her exam by completing at least k number of courses for every two consecutive days and ai is number of courses she can complete for the i^{th} day without getting exhausted and bi be the number of courses Erica must complete for the i^{th} day even if she gets exhausted.

Sub problems:

cour[k,a]=sum(max(0,b_i-a_i))

Base case: if for all b_i<=a_i

cour[k,a]=0

Case 2: if for any b_i>a_i

cour[k,a]=sum(max(0,b_i-a_i))

Pseudo code:

```
Cour(k,a):
        b = []
        exh=0
        b0=a0
        for i in range(1,len(a)):
                if i%2==0:
                        bi=ai
                else:
                        bi=k-ai
```

```
	for i in range(len(a)):
		exh=exh+max(0,bi-ai)
	retrun exh
```
Time Complexity: O(n)

5. let chessman_score[n,a] be the maximum points which can be scored by the chessman before moving out of the given array a[]
Sub problems:
chessman_score[n,a]=max(scores)
Base case: if n=1
chessman_score[n,a]=scores0
Case 2: if n>0
chessman_score[n,a]=max(scores)

Pseudo code:
```
chessman_score(){
	scores = []
	for i in range(n):
		scores[i] = a[i]
		 for j in range((i+a[i]),n,(a[j]+j)):
			scores[i]=scores[i]+a[j]
	return max(scores)
}
```

Time Complexity: O(n)

6. Algorithm:
Step 1: read the two strings a and b
Step 2: check whether the two strings a and b are same or not, if a and b are same then return “a and b are J similar”
Step 3: if a and b are not same then rearrange both a and b in alphabetical order
Step 4: check if len(a) is equal to len(b), if they are same then go to step 5, else return “a and b are not J similar”
Step 5: divide the array into two sub arrays and repeat steps 4 and 5 for each substring, if a substring is J similar then return “a and b are J similar” else return ““‘a and b are not J similar”, repeat these steps until there is no more possibility to divide the sub array

Recurrence Relation:
T(n)=2T(n/2)+n
Master Theorem:
a=2, b=2, k=1, p=0
$\log_b a = \log_2 2 = 1$
->case 2
Therefore: Ø(nlogn)

3. let min_time[i,j] be the minimum sum of travel time required for marco and polo to pass through all the cities, in the shortest time possible, i and j values represent the city index, i<j and $T_{i,j}$ is the time required to travel from one city to another
Sub problems:
Base case: n=1

min_time[i,j] = 0
Pseudo code:
```
min_time(n,T[][]):
    M_time=0
    P_time=0
    for i from 0 to n:
        calculate the time taken to find the shortest time taken for marco to travel from city 1 to city n and add the sum to M_time
        for j from 0 to n:
            find if there is any other possible path to city j and if that path is less than the time taken for marco then assign that time to polo P_time and decrement M_time
    total_time = P_time+M_time
    return total_time
```
Time Complexity: $O(n^2)$