ADS
Homework2

Aashish Paudel
MAtID:30001861

February 22, 2019

Problem 2,1

(a)

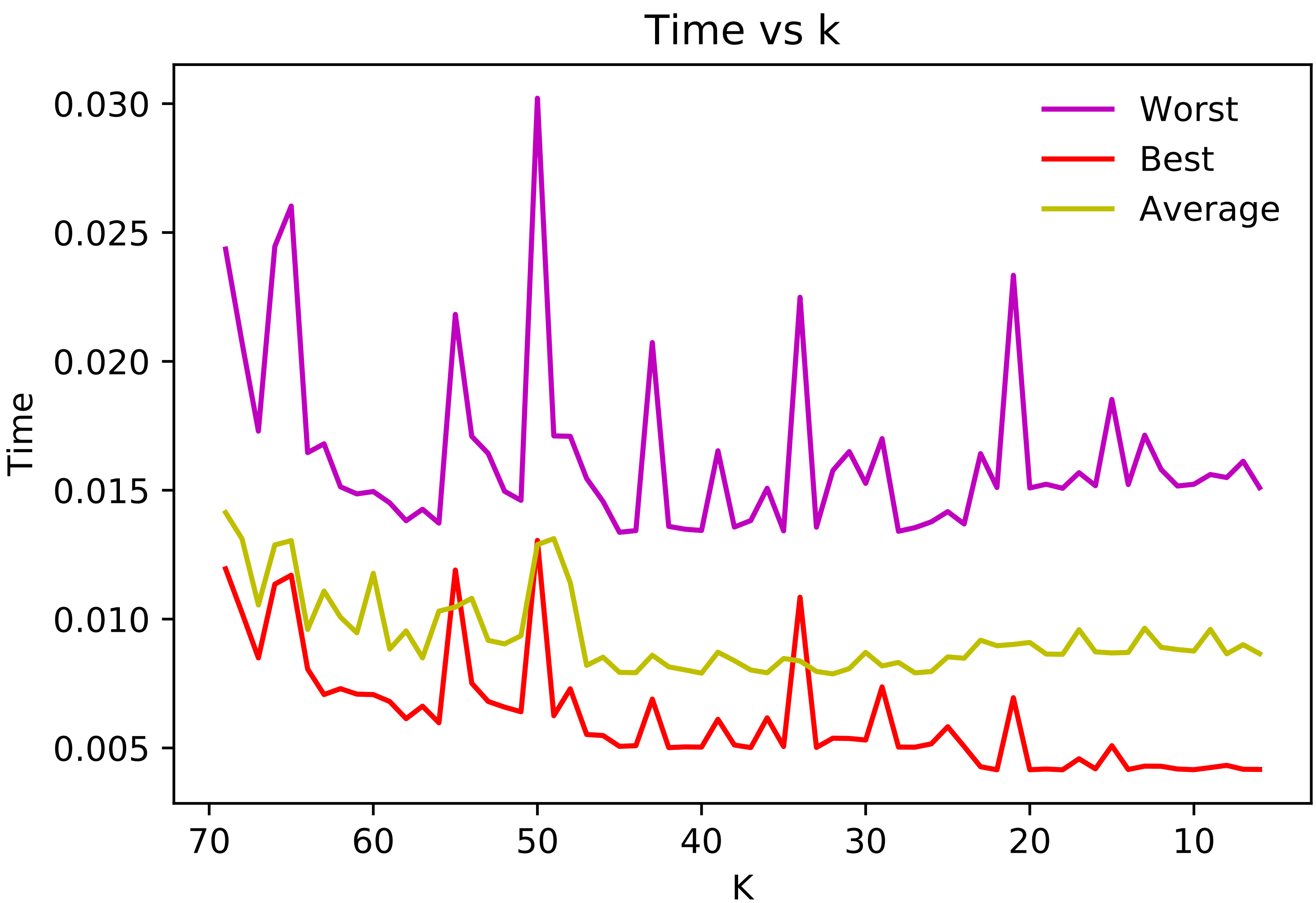
Implementation of algorithm in python is given below:

```
def insertionSort(list_1):
    for index in range(1,len(list_1)):

        currentvalue = list_1[index]
        position = index

        while (position>0 and list_1[position-1]>currentvalue):
            list_1[position]=list_1[position-1]
            position = position-1

        list_1[position]=currentvalue

    return list_1

def merge_sort(arr, K_TO_BREAK = 1):
    if len(arr) > K_TO_BREAK:
        mid = len(arr)//2 #Finding the mid of the array
        L = arr[:mid] # Dividing the array elements
        R = arr[mid:] # into 2 halves

        merge_sort(L, K_TO_BREAK) # Sorting the first half
        merge_sort(R, K_TO_BREAK) # Sorting the second half

        if len(L)==K_TO_BREAK:
            L = insertionSort(L)
        if len(R) == K_TO_BREAK:
```

```python
        R = insertionSort(R)

    i = j = k = 0


    # Copy data to temp arrays L[] and R[]
    while i < len(L) and j < len(R):
        if L[i] < R[j]:
            arr[k] = L[i]
            i+=1
        else:
            arr[k] = R[j]
            j+=1
        k+=1

    # Checking if any element was left
    while i < len(L):
        arr[k] = L[i]
        i+=1
        k+=1

    while j < len(R):
        arr[k] = R[j]
        j+=1
        k+=1

    # Using last index value as sys.maxsize created some problems;
    # Hence, different approach is taken to clear the remaining elements
    # of sorted arrays
```

(b)

The graph is plotted taking n = 1500 and for various values of k. We can see in the graph that with decreasing value of k, the time taken by our combined sorting algorithm decreases.


Time vs k
Worst
Best
Average
0.030
0.025
0.020
0.015
0.010
0.005
Time
70
60
50
40
30
20
10
K

(c)

For, all three cases the time complexity of insertion sort:
$O(n) = n^2$
Here, And, insertion sort is applied to arrays of length k for $\frac{n}{k}$ times. Hence,
$O(k) = (\frac{n}{k}).k^2$
Also, for all three cases the time complexity of merge sort:
$O(n) = n \log n)$
Here,
The height (h) = $\log (n/k)$
Hence,
$O(k) = n \log \frac{n}{k}$

Now, the combined time complexity:
$O(k) = n \log (n/k) + (\frac{n}{k}).k^2)$
$O(k) = n \log n - n \log k + n.k$

Here,

$$O'(k) = \frac{-n}{k} + n \tag{1}$$
$$O''(k) = \frac{n}{k^2} \tag{2}$$

Putting, k = 1 in eqn (1), $O'(k) = 0$
Putting k = 1 in eqn(2), $O''(k) > 0 \; \forall n > 0$(which is always the case)
Hence, O(k) has minimum at k = 1
Also, $\forall k >= 1, O'(k) > 0$. Hence, O(k) is increasing from k.
i.e, more we increase k, our time taken to solve algorithm will increase.

(d)

We can see on the graph that the time decreases with decreasing value of k. On (c) of this problem we can see the same with a little mathematics. Hence, we can safely say that we should take the minimum possible value of k in practice *i.e* 1.

Problem 2.2

(a)

Given,
$T(n) = 36T(n/6) + 2n$
Compare: $T(n) = aT(n/b) + f(n)$
a = 36 b = 6
$\therefore n^{log_6 36} = n^2$

We know, $2n = O(n^{2-\epsilon}), \epsilon =$ very small number
Hence,
$T(n) = O(n^2) \&\& T(n) = \omega(n^2)$

(b)

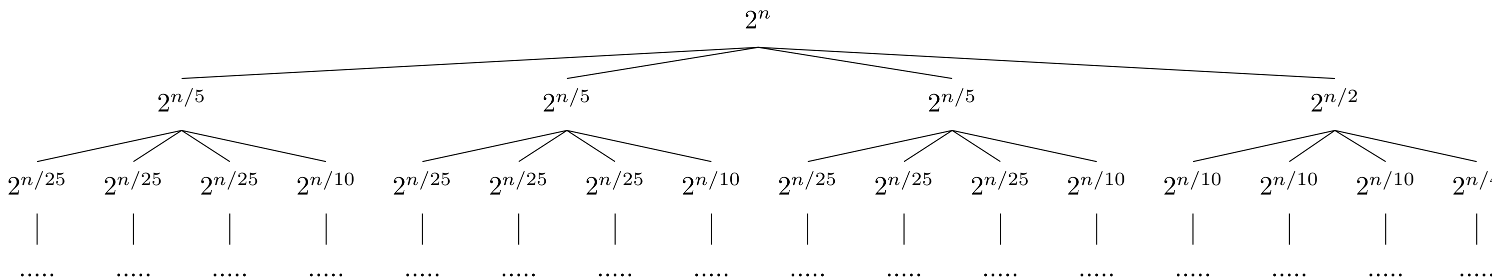
Given,
$T(n) = 5T(n/3) + 17n^{1.2}$
Compare: $T(n) = aT(n/b) + f(n)$
a = 5 b = 3
$\therefore n^{log_3 5} = n^{1.465}$
We know, $17n^{1.2} = O(n^{1.465-\epsilon}), \epsilon =$ very small number
Hence,
$T(n) = O(n^{(\log_3 5)})$ && $T(n) = \omega(n^{(\log_3 5)})$

(c)

Given,
$T(n) = 12T(n/2) + n^2 \lg n$
Compare: $T(n) = aT(n/b) + f(n)$
a = 12 b = 2
$\therefore n^{log_2 12} = n^{3.585}$
We know that any function with a order of 3.585 has upper bound to the function of order 2+δ, where $\delta < 0.5$ *i.e.* $n^2 lg(n) = O(n^{3.585-\epsilon}), \epsilon =$ very small number
Hence,
$T(n) = O(n^{(\log_2 12)})$ && $T(n) = \omega(n^{(\log_2 12)})$

(d)

Given,
$T(n) = 3T(\frac{n}{5}) + T(\frac{n}{2})$
Making a recursion tree,

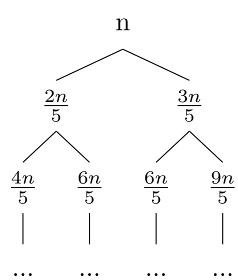

2^n
$2^{n/5}$ $2^{n/5}$ $2^{n/5}$ $2^{n/2}$
$2^{n/25}$ $2^{n/25}$ $2^{n/25}$ $2^{n/10}$ $2^{n/25}$ $2^{n/25}$ $2^{n/25}$ $2^{n/10}$ $2^{n/25}$ $2^{n/25}$ $2^{n/25}$ $2^{n/10}$ $2^{n/10}$ $2^{n/10}$ $2^{n/10}$ $2^{n/4}$
....

We can see that every final leaf will be in the form $a_i.2^{\frac{n}{k_i}}$, where $k_i \in \mathbb{Z}, a_i \in \mathbb{Z}$
$i.e\ 2^n(a_1.2^{\frac{1}{k_1}} + a_2.2^{\frac{1}{k_2}} + a_3.2^{\frac{1}{k_3}} + ...)$
As power value of k_i becomes higher $2^{\frac{1}{k_i}}$ goes to 0.
Thus the sum of given series will converge to some constant (say k)
Thus,
$T(n) = 2^n.k$ or, $T(n) = \theta 2^n$

Hence,
$T(n) = O(2^n)$ && $T(n) = \omega(2^n)$

(e)

Given,
$T(n) = T(\frac{2n}{5}) + T(\frac{3n}{5}) + \theta(n)$
Making a recursion tree,



Here,
Sum at each horizontal end = n
Vertical height (leftmost) = $\log_{3/5} n$
Horizontal height (rightmost) = $\log_{2/5} n$
We say, height (in general) = $\log n$
Thus,
$T(n) = O(n \log n)$ && $T(n) = \omega(n \log n)$