Homework 2 : Algorithms and Data Structures

Digdarshan Kunwar

February 2019

Problem 2.2

Use the substitution method, the recursion tree, or the master method to derive upper and lower bounds for $T(n)$ in each of the following recurrences. Make the bounds as tight as possible. Assume that $T(n)$ is constant for $n \leq 2$.

(a) $T(n) = 36T(n/6) + 2n$

Here we have,
$a = 36$, $b = 6$

$$n^{\log_6 36} = n^2$$
$$f(n) = 2n$$

Case 1:$f(n) = O(n^{2-E})$ for $E = 1$

$$\boxed{T(n) = \Theta\left(n^2\right)}$$

(b) $T(n) = 5T(n/3) + 17n^{1.2}$

Here we have,
$a = 5$,$b = 3$

$$n^{\log_3 5} = n^{1.46}$$
$$f(n) = 17n^{1.2}$$

Case 1:$f(n) = O(n^{1.46-E})$ for $E = 0.265$

$$\boxed{T(n) = \Theta\left(n^{\log_3 5}\right)}$$

(c) $T(n) = 12T(n/2) + n^2 lg(n)$

Here we have,
$a = 12$,$b = 2$

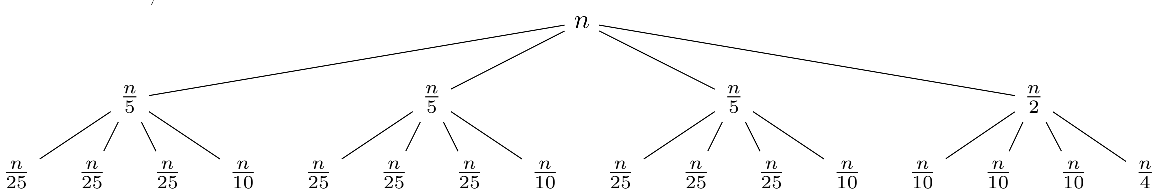
$$n^{\log_2 12} = n^{3.6}$$
$$f(n) = n^2 lg(n)$$
We know that,
$$n^2 lg(n) < n^{3.6}$$

Case 1:$f(n) = O(n^{3.6-E})$ for $E which\ exists$

$$\boxed{T(n) = \Theta\left(n^{\log_2 12}\right) = \Theta\left(n^{3.6}\right)}$$

(d) $T(n) = 3T(n/5) + T(n/2) + 2^n$

Here we have,

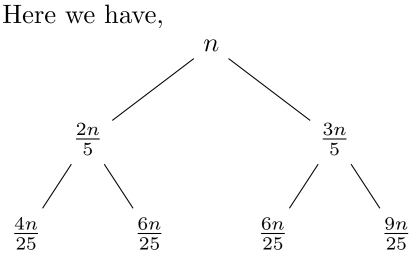


Compared to the total cost of the other levels as of $2^{\frac{n}{5}}, 2^{\frac{n}{2}}, 2^{\frac{n}{25}}...$,the initial cost is always greater than the cost at another levels.
So the initial cost is the greatest.
Therefore it gives us :

$$\boxed{T(n) = \Theta\left(2^n\right)}$$

(e) $T(n) = T(2n/5) + T(3n/5) + \Theta(n)$

Here we have,



The height of the tree for the maximum time for the loop to end is when it takes the rightmost path so we have
So the
$h = log_{\frac{5}{3}} n$ At each levels there are equal n cost we have:
$T(n) = \Theta(log_{\frac{5}{3}} n)$

$$\boxed{T(n) = \Theta\left(n \log n\right)}$$

Problem 2.1

Variant of Merge Sort

!! NOTE !!
Run file named mergesort-variant.py from the zip file.
In the zip file within the folder **Graph Gifs** are the testcases graph.

(a) Variant of Merge Sort

```python
# InsertionSort
def insertionSort(array, l, r):
    for i in range(l, r+1):
        key = array[i]
        j = i-1
        while j >= l and key < array[j]:
            array[j+1] = array[j]
            j -= 1
        array[j+1] = key

# MergeSort
def mergeSort(arr, l, r, k):
    # print("Working Array Length is : ", (r-l+1))
    if ((r-l+1) <= k):
        insertionSort(arr, l, r)
```

```
    else:
        m = (l+(r-1))//2
        mergeSort(arr, l, m, k)
        mergeSort(arr, m+1, r, k)
        return merge(arr, l, m, r)

# Merge
def merge(arr, l, m, r):
    # Size of the two splitted arrays
    sizel = m-l+1
    sizer = r-m

    # Temporary Arrays
    left = [None]*sizel
    right = [None]*sizer

    # Copying values to temporary arrays from arr
    for i in range(0, sizel):
        left[i] = arr[l + i]

    for j in range(0, sizer):
        right[j] = arr[m + 1 + j]

    # Indexes for the two arrays
    i = 0
    j = 0
    # Going throught the array and arranging
    for d in range(l, r+1, 1):
        if i < sizel and j < sizer:
            if left[i] <= right[j]:
                arr[d] = left[i]
                i += 1
            else:
                arr[d] = right[j]
                j += 1
        elif i == sizel:
            arr[d] = right[j]
            j += 1
        elif j == sizer:
            arr[d] = left[i]
            i += 1
```

(b) Here,
In the zip file within the folder **Graph Gifs**.
There are 3 test cases GIF with different **n** for which **k** is increasing :
"**graph-case-1.gif**","**graph-case-2.gif**","**graph-case-3.gif**".
The gifs of the graph with changing **k** were generated combining individual cases of values of k via matplot in python.

(c) Here,
Best Case Observation :
As seen from the generated GIFs of the plots as the k increased from 1 to n where n is the maximum number of elements if they are in best case arrangement they tend to take less time if the value of k is increased.
This is because the the list is already sorted and if k is high the list is divided into just few levels and insertion sort is applied.
And as insertion sort in best case has the time complexity of $\Theta(n)$ it reduces the total time in general

Average Case Observation :
In case of average case we can devise the conclusion that
$height = log(\frac{n}{k})$
So we have the complexity of this variant merge sort as:
$F(k) = nlog(\frac{n}{k}) + (\frac{n}{k})k^2$
So as observed from the GIFs as the k increased for the average case, the time increased for the values of n.
Worst Case Observation :
So as observed from the GIFs as the k increased for the worst case, the time increased drastically for the values of n this is because as k increases we have the insertion sort sorting more number of elements in the worst case array which has the worst case time complexity of n^2. Thus as $n^2 \ > \ n\log(n)$.The sorting time increases as k increases.

(d) From **b)** and **c)** we can observe that if the array is pre sorted then it's best if the k has maximum value.
if the array is random or unsorted its best if the value of k is near around k¿=1 or a more until certain limit be (l) that correspond with the ratio with the total number of elements n.There is a case until k reaches (l) a value such that the average and worst case of is decreasing than k=1 util (l).
Clearly k != n as it generates the most time.