ITCS-6114/8114

Homework 1

Name: Ankit Vaity
Student ID: 801203693

1. For the following decide whether $T(n) = \Theta(f(n))$, $T(n) = \Omega(f(n))$, $T(n) = O(f(n))$ or none of the above. Justify your answer

(a) $T(n) = n^2 + 3n + 4$, $f(n) = 6n + \log n$

$T'(n) = 2n + 3$ $\quad f'(n) = 6 + \frac{1}{n}$

By using L'Hospital's rule

$$\lim_{n\to\infty} \frac{T(n)}{f(n)} = \lim_{n\to\infty} \frac{T'(n)}{f'(n)}$$

$$= \lim_{n\to\infty} \frac{2n+3}{6 + \frac{1}{n}}$$

$$= \frac{2(\infty)+3}{6 + \frac{1}{\infty}}$$

$$= \frac{\infty}{6}$$

$$= \infty$$

$$\left(\text{if } \lim_{n\to\infty} \frac{T(n)}{f(n)} = \infty \Rightarrow T(n) = \Omega(f(n)) \right.$$

Answer: $T(n) = \Omega(f(n))$

(b) $T(n) = n$ $\quad f(n) = (\log n)^{10}$

$T'(n) = 1$ $\quad f'(n) = \frac{10(\log n)^9}{n}$

By L'Hospital's Rule

$$\lim_{n\to\infty} \frac{T(n)}{f(n)} = \lim_{n\to\infty} \frac{T'(n)}{f'(n)} = \lim_{n\to\infty} \frac{1}{\frac{10(\log n)^9}{n}} = \lim_{n\to\infty} \frac{n}{10(\log n)^9} = \frac{\infty}{\infty}$$

As L'Hospital's ~~Rule~~ conditions are satisfied we can take a derivative again

$$\lim_{n\to\infty} \frac{\frac{d}{dn}(n)}{\frac{d}{dn}10(\log n)^9} = \lim_{n\to\infty} \frac{1}{\frac{10\times 9\times(\log n)^8}{n}} = \lim_{n\to\infty} \frac{n}{(10\times 9)(\log n)^8} = \frac{\infty}{\infty}$$

Now after recursively taking derivatives we will get the following:

$$\lim_{n\to\infty} \frac{\frac{d}{dn}(n)}{(10\times 9\times 8\times 7\times \cdots \times 2)\frac{d}{dn}\log n} \quad \ldots\ldots (\text{lets consider } C = 10!)$$

$$= \lim_{n\to\infty} \frac{1}{\frac{C}{n}}$$

$$= \lim_{n\to\infty} \frac{n}{C}$$

$$= \infty$$

$$\left(\text{if } \lim_{n\to\infty} \frac{T(n)}{f(n)} = \infty \implies F(n) = \Omega(f(n))\right)$$

ANSWER: $T(n) = \Omega(f(n))$

(c) $T(n) = \sum_{i=1}^{n} 3^i$, $f(n) = 3^{n-1}$

$$= \frac{3^{n+1} - 1}{3-1} - 1 \quad \cdots \left(\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a-1} \right)$$

$$= \frac{3^{n+1} - 1}{2} - 1$$

$$= \frac{3^{n+1} - 1 - 2}{2}$$

$$= \frac{3^{n+1}}{2} - \frac{3}{2}$$

$$\lim_{n \to \infty} \frac{T(n)}{f(n)} = \lim_{n \to \infty} \frac{\frac{3^{n+1}}{2} - \frac{3}{2}}{3^{n-1}}$$

$$= \lim_{n \to \infty} \frac{1}{2} \left(\frac{3^{n+1}}{3^{n-1}} - \frac{3}{3^{n-1}} \right)$$

$$= \lim_{n \to \infty} \frac{1}{2} \left(3^{n+1-(n-1)} - \frac{1}{3^{n-1-1}} \right)$$

$$= \lim_{n \to \infty} \frac{1}{2} \left(3^2 - \frac{1}{3^{n-2}} \right)$$

$$= \frac{1}{2} \left(3^2 - \frac{1}{3^{\infty - 2}} \right)$$

$$= \frac{1}{2} \left(3^2 - \frac{1}{\infty} \right)$$

$$= \frac{3^2}{2}$$

$$= \frac{9}{2}$$

$$\left(\text{if } \lim_{n \to \infty} \frac{T(n)}{f(n)} < \infty \Rightarrow T(n) = \Theta(f(n)) \right)$$

ANSWER : $T(n) = \Theta(f(n))$

(d) $T(n) = n^{\log c}$ $\quad f(n) = c^{\log n}$

$$\lim_{n \to \infty} \frac{T(n)}{f(n)} = \lim_{n \to \infty} \frac{n^{\log c}}{c^{\log n}}$$

$$= \lim_{n \to \infty} \frac{\log (n^{\log c})}{\log (c^{\log n})}$$

$$= \lim_{n \to \infty} \frac{\log c \times \log n}{\log n \times \log c}$$

$$= \lim_{n \to \infty} 1$$

$$= 1$$

$$\left(\text{if } \lim_{n \to \infty} \frac{T(n)}{f(n)} < \infty \implies T(n) = \Theta(f(n)) \right)$$

ANSWER : $T(n) = \Theta(f(n))$

2. Let A[] be an array of n distinct numbers. If index $i<j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A. In other words, two elements of the array are considered to form an inversion if they are "out of order".

(a) List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$

ANSWER: (2,1) (3,1) (8,6) (8,1) (6,1)

(b) What array with elements from the set $\{1, 2, 3, \cdots n\}$ has the most inversions? How many does it have? Justify your answer

Array sorted in descending order will have most inversions

$[n, n-1, n-2, \cdots, 2, 1]$

Array element	no. of inversions
n	(n-1)
n-1	(n-2)
n-2	(n-3)
⋮	⋮
2	1
1	0
	$\Sigma = \frac{n(n-1)}{2} \cdots \left(\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \right)$

We subtract n from this result

ANSWER: $\frac{n(n-1)}{2}$

(c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

Insertion sort	Cost	times
for $j = 2$ to A.length	c_1	n
key = A[j]	c_2	$n-1$
sequence A[1...j-1]	0	$n-1$
i = j-1	c_4	$n-1$
while i > 0 and A[j] > key	c_5	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i = i-1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i+1] = key	c_8	$n-1$

running time $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$

Best Case : $O(n)$ ... when the array is ~~sorted~~ in ascending order ie. with no inversions

Worst Case : $O(n^2)$ ... when the array is in descending order ie with most inversions

~~Therefo~~ Because in Best case for each value of j the while loop will make only one comparison. and in Worst case for each value of j the while loop will make j-1 comparisons.

ANSWER : Greater the number of inversions in ~~insertion sor~~ input array, greater will be the running time of insertion sort

(d) Give an algorithm that counts the number of inversions in any permutation on n elements in $\Theta(n \log n)$ worst case time. (Hint: Consider which of the sorting algorithms may be modified to solve this problem)

We will modify the merge sort algorithm to count the number of inversions.
In the first recursive call we will count the number of inversions in left subarray
In the second recursive call we will count the number of inversions in right subarray
Finally we will count inversions in the merge step
In the merge step both the left and right sub arrays will have sorted elements. So if we find an element in left sub array that is greater than an element in the right sub array, then all the remaining elements in the left subarray will be greater than the element in right sub array that is under consideration.

Pseudo code:

```
MergeSort (arr)
    if arr.length ≤ 1
        return 0
    else
        mid = arr.length / 2
        left = arr [: mid]
        right = arr [mid:]

    Count_Left = MergeSort (left)
    Count_right = MergeSort (right)
    count_merge = MergeCount (left, right, arr)
    return count_left + count_right + count_merge
```

P.T.O.

```
MergeCount (left, right, arr):
    ~~inversions = 0~~
    while i < left.length and j < right.length
        ~~if arr[i] <= a~~
        if left[i] ≤ right[j]
            arr[k] = left[i]
            i++
        else
            arr[k] = right[j]
            j++
            inversions = left.length - i
        k++
    return inversions
```

ANSWER:
(Python code implementations)

```
def mergeSort(arr):
    inv_count = 0

    if len(arr) <= 1:        // Base condition
        return 0

    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]

    inv_count += mergeSort(left)
    inv_count += mergeSort(right)
    inv_count += ~~mergeArrays~~ mergeCount(left, right, arr)

    return inv_count
```

P.T.O

```
def mergeCount(left, right, arr):
    i = j = k = 0
    inv_count = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            arr[k] = left[i]
            i += 1
        else:
            arr[k] = right[j]
            inv_count += (len(left) - i)  // increase the count
            j += 1
        k += 1

    // check if any element was left in two halves
    while i < len(left):
        arr[k] = left[i]
        i += 1
        k += 1
    while j < len(right):
        arr[k] = right[j]
        j += 1
        k += 1

    return inv_count
```

3. Given a set S of n integers and another integer x, describe an efficient algorithm that determines whether or not there exist two elements in S whose sum is exactly x. Justify the running time of your algorithm.

Using hash table to find sum pair

ANSWER:

Pseudo Code:

```
findSumHash (arr, val)
    // define hash table
    hashTable = {}
  for i = 0 to (arr.length - 1)
    find = val - arr[i]
    if (hashTable.search(find))
        return True
    hashTable.insert(arr[i])

  return False
```

Python Implementation:

```
def findSumHash(arr, val):
    hashTable = {}
    for i in range(len(arr)):
        find = val - arr[i]
        if find in hashTable:
            return True
        hashTable[arr[i]] = arr[i]
    return False
```

Running Time:

Worst case running time is when scan the whole array and didn't find any pair with with sum equal to x

Time complexity = Time complexity of inserting n elements + Time Complexity of x, n times = $n + n = O(n)$

4.

(a) Prove every polynomial $p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0$ with $a_k > 0$ belongs to $\Theta(n^k)$

Lets check if $p(n) \leq c n^k$
if $c = a_k + a_{k-1} + a_{k-2} + \cdots + a_0$

then,

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 \leq (a_k + a_{k-1} + a_{k-2} \cdots + a_0) n^k$$

$$p(n) \leq a_k n^k + a_{k-1} n^k + \cdots + a_0 n^k$$

$$\therefore p(n) = O(n^k)$$

Lets check if $p(n) > c n^k$
if $c = a_k$

then,

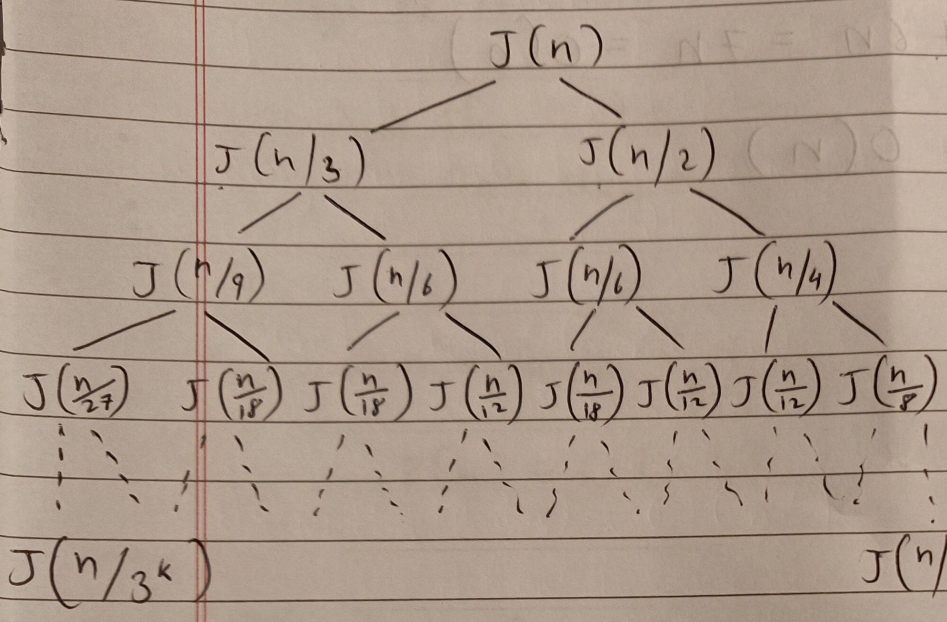
$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 > a_k n^k$$

$$p(n) > a_k n^k$$

$$\therefore p(n) = \Omega(n^k)$$

Answer: Since $p(n) = O(n^k)$ and $p(n) = \Omega(n^k)$
we can say $p(n) = \Theta(n^k)$

(b) Use a recursion tree to compute an asymptotic solution for the recurrence $J(n) = J(n/2) + J(n/3) + n$



Level	#nodes	time per level
0	1	n
1	2	$5/6\ n$
2	2^2	$(5/6)^2 n$
3	2^3	$(5/6)^3 n$
⋮	⋮	⋮
k	2^k	$(5/6)^k n$

Height of left subtree:

$\frac{n}{3^k} = 1$

$n = 3^k$

$\log_3 n = k$

Height of right subtree:

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 n = k$

right subtree will take longer time.

lower bound: $(5/6)^{\log_3 n} n$

$\sum_{k=0}^{\log_3 n} (5/6)^k n$

upper bound: $(5/6)^{\log_2 n} n$

$\sum_{k=0}^{\log_2 n} (5/6)^k n$

Total cost = Cost of leaf nodes + Cost of inside nodes

Cost of leaf nodes $= 2^k = 2^{\log_2 n} = n$

Cost of inside node $= n + \frac{5}{6}n + \left(\frac{5}{6}\right)^2 n + \left(\frac{5}{6}\right)^3 n + \cdots$

$= n\left[1 + \left(\frac{5}{6}\right) + \left(\frac{5}{6}\right)^2 + \left(\frac{5}{6}\right)^3 + \cdots\right] = n\left[\frac{1}{1-5/6}\right] \cdots \left(G.P = \frac{a}{1-r}\right)$

Cost of initial nodes $= n \begin{bmatrix} 1 \\ 1/6 \end{bmatrix} = 6n$

Total cost $= n + 6n = 7n = O(n)$

ANSWER : $J(n) = O(n)$

5. Give asymptotic bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

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

Using Master's theorem
$a = 5$, $b = 2$, $\log_2^5 = 2.32$, $c = 2$

$\log_b^a > c$
$\therefore$ Case 1 of Master's theorem applies, so

ANSWER: $T(n) = \Theta(n^{\log_2^5})$

(b) $T(n) = 3T(n/27) + n^{0.25}$

Using Master's theorem
$a = 3$, $b = 27$, $\log_{27}^3 = 0.33$, $c = 0.25$

$\log_b^a > c$
$\therefore$ Case 1 of Master's theorem applies, so

ANSWER: $T(n) = \Theta(n^{\log_{27}^3})$

(c) $T(n) = 16T(n/4) + n^2 + n\log n$

Using master theorem
$a = 16$, $b = 4$, $\log_4^{16} = 2$ $f(n) = n^2 + n\log n = \Theta(n^2)$, $l = 2$, $k = 0$

$l = \log_b^a$
Case 2 of master's theorem is applied, so

ANSWER: $T(n) = \Theta(n^2 \log n)$

(d) $T(n) = 3T(n-1) + 1$

By masters theorem for decreasing function
$a = 3$, $b = 1$ $f(n) = n^0$

$a > 1$
Case 3 is applied so,

ANSWER: $T(n) = O(3^n)$