# CS 5633: Analysis of Algorithms

### Homework 2

### **Explanation**:

Divide and Conquer algorithm can be divided into three parts: Divide, Conquer and Combine.

**Divide:** This involves dividing the problem into smaller sub-problems. In this problem, the unsorted array has n elements. So, we will divide the array into two arrays each with n/2 elements. We will keep dividing the array till the base case hit i.e. each sub-array has one or two elements.

**Conquer:** This involves solving sub-problems by calling recursively. Here, the base case is the sub-problem with elements one or two. When this situation arises, the sub-problem is considered to be solved.

**Combine:** At this stage, solutions from different sub-problems is merged. Here, we will compare solutions from two sub-problems and will return the index of the smaller value.

**English explanation:**

The divide-and-conquer algorithm for finding the index of the smallest number is an unsorted array of n distinct numbers works as follows:

1. This algorithm returns the index of that element when the input array has only one element.
2. At the time of more than one element, this algorithm divides them into two equal-sized subarrays.
3. Find the index of the smallest number using the same algorithm in each subarrays.
4. Finally, return the index of the smaller number which is found between the subarrays in step 3 using the divide-and-conquer algorithm.

### **Pseudocode:**

This program takes an unsorted array with n elements and returns the index of the lowest element using the divide-and-conquer approach.

function MinIndex(arr, left, right) {

if (left == right) return left

else if (right-left == 1) {

if (arr[left] > arr[right]) return right

else return left

}

else {

min\_1 = MinIndex(arr, left, (left+right)/2);

min\_2 = MinIndex(arr, ((left+right)/2)+1, right);

if(arr[min\_1] > arr[min\_2]) return min\_2

else return min\_1

}

}

### **Recurrence relation:**

At each step, the array is divided into two halves and compared once.

So, The recurrence relation for the program is:

T(n) = 2T(n/2) + 1

To find the Θ bound we have to find O and Ω bounds.

Assume, T(k) <= c.k where c is a constant and c > 0

Now, T(n) = 2T(n/2) + 1

<=2.c.n/2 + 1

=c.n+1

= c(n+1) - c

<= c(n+1)

So, T(n) ε O(n) where c is a constant and c > 0

Assume, T(k) >= c.k where c is a constant and c > 0

Now, T(n) = 2T(n/2) + 1

>=2.c.n/2 + 1

=c.n+1

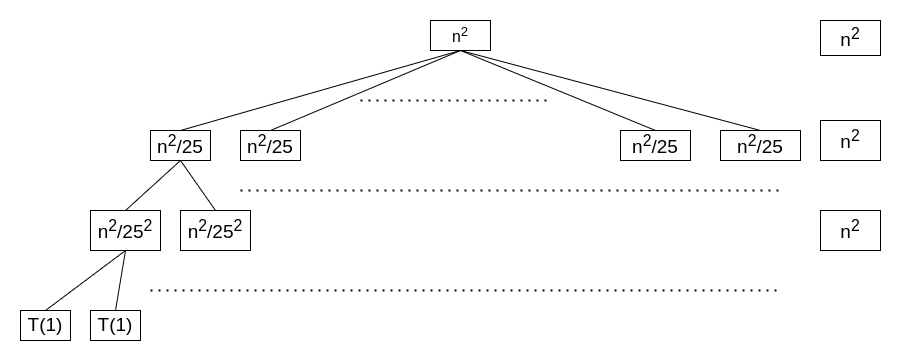
>= cn

So, T(n) ε Ω(n) where c is a constant and c > 0

As,T(n) ε O(n) and T(n) ε Ω(n),

So, T(n) ε Θ(n)

1. a)



Assumption: Assume that n is the exact power of 5

T(n) = Cost of Leaf Node(Lc) + Cost of Internal Node(Ic)

Here, n / 5k = 1

=> n = 5k

=> log5n = log55k

=> k = log5n

So, the height of the tree k = log5n

Now, Lc = 5k

= 5log5n

= nlog55

= n

Now, Ic = n2 + 25. n2/25+ 252. n2/252 + 253. n2/253  + ….

= n2 + n2 + n2 + n2 + …

= n2 . k [here k is the height of the tree]

= log5n.n2

So, T(n) = Lc + Ic

= n + log5n.n2

So, the run time complexity is O(log5n.n2)

**Proof by Induction**:

Assume, T(k) <= ck2log5k for k < n

The recurrence relation is, T(n) = 25T(n/5)+n2

We have to prove a base case. For simplicity let’s prove for n = 5

T(5) = 25.T(1)+52

=> 25.1+25

=> 50 <= c. 25. log55

=> 50 <= c. 25

=> c >= 2

So, this recurrence relation is true for c >= 2

Now, T(n) = 25T(n/5)+n2

<= 25.c.(n/5)2.log5n/5+n2

= 25.c.n2/52 (log5n-log55)+n2

= c.n2.log5n - (cn2 - n2 )

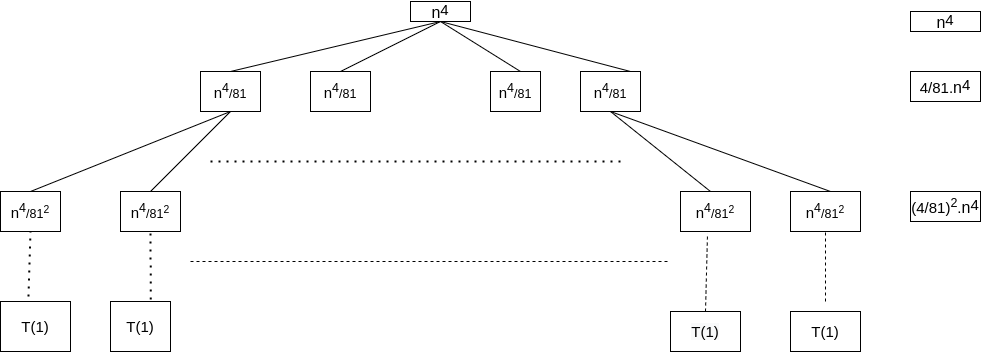
<= c.n2.log5n for cn2 - n2 >= 0

cn2 - n2 >= 0

=> c >= 1

So, T(n) = O(n2.log5n)

b)



Assumption: Assume that n is exact power of 3

T(n) = Cost of Leaf Node(Lc) + Cost of Internal Node(Ic)

n / 3k = 1

=> n = 3k

=> log3n = log33k

=> k = log3n

So, the height of the tree = log3n

Now, Lc = 4k

= 4log3n

= nlog34

Now, Ic = n4 + 4/81. n4 + (4/81)2. n4 + (4/81)3. n4 + ….

= (4/81)0.n4 + (4/81)1. n4 + (4/81)2. n4 + (4/81)3. n4 + ….

= n4 . [(4/81)0 + 4/81)1 + (4/81)2 + (4/81)3 +….]

= n4 . 1/(1-4/81)

= 81.n4 /77

So, T(n) = Lc + Ic

= nlog34 + 81.n4 /77

log34 is less than 2. So, nlog34 is less than n2

So, the run time complexity is O(n4)

**Proof by Induction**:

The recurrence relation is, T(n) = 4T(n/3)+n4

Base Step:

We have to prove a base case. For simplicity let’s prove for n = 3

T(3) = 4T(1)+34

=> 4.1+81

=> 85 <= c. 34

=> 85 <= c. 81

=> c >= 85/81

So, this recurrence relation is true for c >= 85/81

Hypothesis:

Assume, T(k) <= cn4 for all k < n

Induction Step:

Now, T(n) = 4T(n/3)+n4

<= 4.c.(n/3)4+n4

= 4.c.n4/ 81+n4

= cn4 + (4.c.n4/ 81+n4 - cn4)

= cn4 + (-77cn4/ 81+n4)

<= cn4 for -77cn4/ 81+n4 <= 0

So, -77cn4/ 81+n4 <= 0

=> n4(1 - 77c/81) <= 0

=> 1 - 77c/81 <= 0

=> 81 - 77c <= 0

=> c >= 81/77

So, T(n) = O(n4)