Foundation of Algorithm

Priyanka Dwivedi
Srikant Lakshminarayan

February 13, 2018

Problem - 1

1. The recurrence function which captured the running time is:
   T = O(1) if (left==right)
   T = 2T(n/2) + O (1) if (left ! = right)

2. $T(n) <= 2T(n/2) + d$
   $T(n) <= 2 * (2T(n/4) + d) + d$
   $T(n) <= 4T(n/4) + 3.d$
   $T(n)4 * (2T(n/8) + d) + 3.d$
   $T(n) = 8T(n/8) + 7.d$
   stop when $n = 2^k$
   $T(n) = n.T(1) + (2^k - 1).d$
   $T(n) = n + (n - 1)$
   $T(n) <= n.d$
   $T(n) = O(n)$

3. The code gives us the longest sequence in an array along with the lengths of left longest sequence and right longest sequence. Suppose if an array given is[6 6 6 8 8 9 9 10 11 12 12 12 12 12] So values returned are (3, 5,5)

Problem - 2

1. $T(n) = 4T(n/2) + n^2$
   $a = 4, b = 2$
   $f(n) = n^2$
   $log_b a = log_2 4 = 2$
   $n^{log_b a} \quad vs \quad f(n)$
   $n^2 \quad vs \quad n^2$
   $n^{log_b a} = f(n)$
   Using Master's Theorem Case 2, $T(n) = \Theta(n^{log_b a} logn)$we get,

$$T(n) = \Theta(n^2 logn)$$

2. $T(n) = 2T(n/2) + \sqrt{n}$
   $a = 2, \quad b = 2$
   $f(n) = \sqrt{n}$
   $log_b a = log_2 2 = 1$
   $n^{log_b a} \quad vs \quad f(n)$

$n^1 \quad vs \quad \sqrt{n}$
$n^1 \quad > \quad \sqrt{n}$
Using Master's Theorem Case 1, $\quad T(n) = \Theta(n^{log_b a})$ we get,

$$T(n) = \Theta(n)$$

3. $T(n) = 7T(n/3) + n^2$
   $a = 7, \quad b = 3$
   $f(n) = n^2$
   $log_b a = log_7 3$
   $n^{log_b a} \quad vs \quad f(n)$
   $n^{log_7 3} \quad vs \quad n^2$
   $n^{log_7 3} \quad < \quad n^2$
   Using Master's Theorem Case 3, $\quad T(n) = \Theta(f(n)$ we get,

$$T(n) = \Theta(n^2)$$

4. $T(n) = 2T(n/8) + n^{1/3}$
   $a = 2, \quad b = 8$
   $f(n) = n^{1/3}$
   $log_b a = log_8 2 = 1/3$
   $n^{log_b a} \quad vs \quad f(n)$
   $n^{1/3} \quad vs \quad n^{1/3}$
   $n^{log_8 2} \quad < \quad n^{1/3}$
   Using Master's Theorem Case 2 $T(n) = \Theta(n^{log_b a} logn)$we get,

$$T(n) = \Theta(n^{1/3} logn)$$

Problem - 3

Algorithm Used : Merged sort with counting Inversion to calculate swap counts.

Algorithm:

1. Take User I/p and store in list
2. Divide entire list by recursively splitting list into two equal half.

3. Merge two halves according to condition given by photographer as follows Condition for left and right pointer for merging two halves
   (a) If both students 7 years old. Compare their height and place in ascending order of height.
   (b) If both student 8 years old. Compare their height and place in descending order of height.
   (c) If one student and other Professor
      i. If student 7 year then place student before Professor
      ii. If student 8 year then place Professor before student
   (d) If both student with different age, compare age and place in ascending order of age.
4. While Merging two halves according to photographer we calculate number of swaps to place person on current position.To calculate the swap count we use Counting inversion algorithm which is as follows: All the points are with reference to left and right pointer in two halves.
   (a) If the element in right part is to be placed then we need to calculate swap count as length of left - current left pointer.
   (b) If the element in left part is to be placed then we don't need to calculate swap count.
5. The swap count is combined and summation of each swap count will give us entire swaps count performed for placing everyone in right place.

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

(a) Access element at particular index : O(1)
(b) Append Element in list:O(1)
(c) Extend k Element in list : O(k)
(d) Length of list len(n) : O(1)

Overall Complexity of Algorithm

(a)	Append n i/p from user	O(n)
(b)	Divide list recursively in two equal halves	O(logn)
(c)	Merge list and calculate swap count for every merge	O(n)
(d)	Merge List and calculate swap count logn times	O(nlogn)
(e)	Overall complexity for algorithm is	O(nlogn)

Correctness :
To Solve the Problem we have used **Divide and Conquer** approach. The problem is to find number of swap to be performed to place students and Professor according to photographers instruction i.e Place 7 years students first in ascending order of height followed by Professor and 8 years student in descending order of height to take photograph. We divide the problem and independently solve and merge each problem. Here we divide problem the same way we divide in merge sort and merge data following the condition by photographer and compute the swap count by not actually swapping but calculating the number of swap counts that will be performed to place element at correct position.Since we divide the problem into half we recursively divide data therefore complexity is O(logn) and to merge the element and calculate the swap count it take O(n) but since it is done logn times the overall complexity turns out to be O(nlogn).

Problem - 4

Algorithm:

(a) Take n points.

(b) With these n points or coordinates make or create lines.

(c) Calculate slope and midpoints of all these lines.

(d) Creating a perpendicular bisector at the midpoint, calculate the value of the line and find the y-intercept.

(e) For calculation of y-intercept, if the value of the slope is infinity then the x value from midpoint is taken as intercept. if the slope is 0 then intercept is equal to y value of midpoint.

(f) The values are then sorted according to slopes and then y-intercept.

(g) Then the maximum number of occurrences of slope and intercept are formed.

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

(a) Access element at particular index : O(1)

(b) Append Element in list:O(1)

(c) Length of list len(n) : O(1)

Overall Complexity of Algorithm

(a) Take n input coordinates and make a list	O(n)
(b) Create lines with each input point with each other (getting all possible lines)	$O(n^2)$
(c) Calculate midpoints of lines	O(n)
(d) Calculate slope and intercept	O(n)
(e) Sort the values using merge sort	O(nlogn)
(f) Count the number of Same slopes and y-intercept	O(n)

Correctness
The algorithm always makes a line with the coordinates given and calculates the midpoints of those lines. Suppose there are many points who are getting aligned at a fold. The fold is also a line which is passing through all the midpoints. Therefore, if we get the equation of that fold or the intercept and its slopes then we can find all those lines from which that fold (line) is passing.

Problem - 5

(a) Algorithm Used : Select Random to compute any element present in array which occurs more than n/2 times

Algorithm:
Select(numbers,k)

i. Take n elements from user.
ii. Randomly select any value from numbers
iii. Iterate through the values and partition in three buckets Less,Equal and Greater
   A. Less : This bucket will have all values less than pivot.
   B. Equal : This bucket will have all values equal to pivot.
   C. Greater : This bucket will have all values greater than pivot.
iv. Compare searching index 'k' as follows:
   if k is less than length(Less) then element is within Less Bucket and call select with Less and k.
   if k is greater length(Less) and k is less length(Less)+length(Equal) then we have found the element i.e pivot.
   else element is in greater bucket and select only greater bucket as data and new k value i.e k-length(Less)-1
v. The result of Select function is length of Equal bucket. The length of equal is checked compared to n/2 to find whether there exist element which occurs more than n/2.
vi. if length is greater than n/2 then o/p is Yes.

Algorithm Used : Select Random to compute any element present in array which occurs more than n/3

Algorithm:

Select(numbers,k)

(a) Take n elements from user.
(b) Randomly select any value from numbers
(c) Iterate through the values and partition in three buckets Less,Equal and Greater
   i. Less : This bucket will have all values less than pivot.
   ii. Equal : This bucket will have all values equal to pivot.
   iii. Greater : This bucket will have all values greater than pivot.
(d) Compare searching index 'k' as follows:
   if k is less than length(Less) then element is within Less Bucket and call select with Less and k.
   if k is greater length(Less) and k is less length(Less)+length(Equal) then we have found the element i.e pivot.

else element is in greater bucket and select only greater bucket as data and new k value i.e k-length(Less)-1

(e) The result of Select function is length of Equal bucket. The length of equal is checked compared to n/2 to find whether there exist element which occurs more than n/3.

(f) if length is greater than n/3 then o/p is Yes.

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

(a) Access element at particular index : O(1)

(b) Append Element in list:O(1)

(c) Randomly generate pivot value : O(n)

(d) Length of list len(n) : O(1)

Overall Complexity of Algorithm

(a)	Append n i/p from user	O(n)
(b)	Randomly find a pivot value	O(n)
(c)	Calculate Less,Equal,Greater bucket with respect to pivot	O(n)
(d)	Compare and check if element at index k achieved	O(1)
(e)	Check the count greater than n/2	O(1)
(f)	Complexity for computing element greater n/2 and n/3 is 2	O(n)
(g)	Overall complexity	O(n)

Correctness :
Assumption: The random pivot value is good. The algorithm used to solve problem is **Select-Random**. In the problem we need to find whether there exist a element which occurs more than n/2 time and also if there is element which exist more than n/3 times. The algorithm randomly select value from given i/p as pivot and compute the less,equal,greater bucket with respect to that pivot.Check we have got element at n/2 and n/3.If yes return length of equal bucket else check according to the index if we need to calculate less or greater bucket.Since we need to find element which occurs more than n/2

times we need to check which element is present at n/2 index because it has to appear in the array more than n/2 times it has to pass n/2 index.Same is for element which appears more than n/3 times.It should pass the element at index n/3 so that it appears more than n/3 times.The Complexity to randomly select pivot value and compute bucket is O(n). Other operation can be performed in constant time so Overall complexity is O(n).