Aayush Shah
19BCE245 Tut-4 27/9/21

Date
Page

1.) Heap sort is not a divide and conquer alogorithm.

2. • We know that the above alogorithm is calculating the gcd of the two numbers x and y.
• It is based on the fact that when smaller number is divided from larger number, gcd b/w the two numbers remain same.
• Now, we can take a simple example to get time complexity of above code.
let $x=10$ $y=1$
we know their gcd is 1.
So, we need to subtract y from x 9 times.
which is $O(n)$.
Hence, time complexity of above alogrithm is $O(n)$

3) Now, $p(x) = a_0 + x(a_1 + a_2 + a_3 x(x))$

from above representation of the polynomial it is clear that there are 3 multiplication steps involved.

Hence, minimum number of multiplication steps = 3.

4. Maximum subarray sum problem using divide and conquer has a worst case time complexity of $O(n \log n)$

5. We can create a power function with the best time compexity of $O(\log n)$

6. Since b_1 has all array elements sorted, for t_1 we have a worst case time complexity as compared to an average time complexity for t_2.

7.
8.
9.

19BCE245

Tut - 4

Date
Page

$t_1 \rightarrow O(n^2)$, $t_2 \rightarrow O(n\log n)$.

Hence, $t_1 > t_2$

7. Time complexity will remain $O(n^2)$ because of swaps required to to be carried out.

8. Randomized quick sort worst case = $O(n^2)$ because we may randomly pick corner element each time.

9. Pseudo code of binary search is as follows:

```
Procedure binary search:
    A ← Sorted array
    n ← Size of array
    x ← Value to be searched

    Set lower bound = 1
    Set upper bound = n

  while x not found
      if upper bound < lower bound
          Exit : x doesn't exists

      Set mid point = (Lower bound + Upper bound) / 2

      If A[mid Point] < x
          Set lower bound = mid point + 1

      If A[mid Point] > x
          Set upper bound = mid point - 1

      If A[mid Point] = x
          Exit : X found at location mid Point
  end while
end procedure.
```

19BCE245
Tut-4

Date
Page

Now, since we are dividing the array in half and trying to obtain a solution binary search is a divide and conquere algorith with recurrence relation as follows:

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

using master method,

$a = 1 \quad b = 2 \quad d = 0 \quad \Rightarrow b = 1$

$a = b^d$

$\therefore T(n) = O(n^d \log n)$

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

Hence binary search has a time complexity of $\Theta(\log n)$

Best case : mid point is the solution
$T(n) = O(1)$, Eg search 2 in {1, 2, 3}

Average / worst case : mid point is not the solution
$T(n) = O(\log n)$
Eg. → search 2 in {1, 2, 3}
~~A = 10~~

10.) Quick sort algo. is as follows:

Algo for partition:

Step 1 : Choose the highest index value as pivot.
Step 2 : Take 2 variables to point left and right of the list excluding pivot
Step 3 : left points to ~~for~~ low index.
Step 4 : Right points to high index.
Step 5 : While value at left is less than pivot move right
Step 6 : While value at right is higher than pivot move left
Step 7 : If both step 5 and 6 don't matches, swap left and right.

19BCE245
Tut-4

Date
Page

Step 8: If left ≥ right, the point where they met is new pivot

▷ Algo for quick Sort:

1> Make the right most index value pivot
2> Partition the array using pivot value.
3> Quicksort left partition recursively
4> Quicksort right partition recursively

Now, since we are recursively calling quicksort, by dividing the array in 2 parts i.e. left right and trying to obtain a solution, quick sort is a D&C algo with recurrence relation as follows

In best case middle element is always taken as pivot:
$T(n) = 2T(n/2) + O(n)$ → best case

Using Master's ~~Algo~~ Theorem,
$a = 2$, $b = 2$, $d = 1$ ⇒ $b^2 = 2$
$\therefore a = b^d$
$\therefore T(n) = \Theta(n^d \log n)$
$\therefore T(n) = \Theta(n \log n)$

For worst case, smallest or largest element is always chosen as pivot (i.e. generally when array is already sorted)

$\therefore$ for worst case, $T(n) = O(n^2)$
eg $\{1, 2, 3, 4, 5\}$ because $T(n) = T(n-1) + O(n)$

11.) Merge sort algo is as follows:

Algo for merge sort:

19BCE245

Tut-4

Date

Page

1) If size of array passed to function, is 1 then return array element
2.) Divide array into two points as left array and right array
3.) Recursively call mergesort for left array and right-array
4.) Merge left array and right arry

> Algo for merge:

1.) Create a new array merged array having size equal to size of left array and right-array
2) While both arrays are not iterated execute step-3
3.) Give current element of merged array as the max value b/w the current element values of left array and right array and increase current element value of merged array and the max (left array, right-array)
4.) Empty left-array, remaining elements into merged array.
5.) Empty right-array remaining elements into meged array
6.) Return merged-array

Now since, at each step merge-sort divides the array into 2 parts and merges those two parts in linear time to give sorted array it is a D&C algo with the following recurrence relation

$$T(n) = 2T(n/2) + \Theta(n)$$

This soln is same as for best, worst & average cases because merge sort doesn't discriminate b/w the array elements & runs a complete algorithm every step.

From master's theorm,

$a = 2$, $b = 2$, $d = 1$

$\therefore a = b^d$

19BCE245

Tut-4

Date
Page

$\therefore T(n) = O(n^d \log n)$

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

12) Yes, we can improve the time complexity of multiplying large integer using Karatsuba Algorithm (Divide & Conquer)

Karatsuba Algorithm says that if we represent a binary string of input numbers then we can divide the binary strings into 2 parts and obtain a solution for multiplication. Eg. Assume X and Y for 2 numbers binary representation.

So, X_l = leftmost $n/2$ bits
X_r = Rightmost $n/2$ bits
Y_l = leftmost $n/2$ bits
Y_r = Rightmost $n/2$ bits

So, $X = (X_l)2^{n/2} + X_r$ $Y = (l)2^{n/2} + Y_r$

$\therefore XY = 2^n(X_l Y_l) + 2^{n/2}(X_l Y_r + X_r Y_l) + X_r Y_r$

$\therefore XY = 2^n(X_l)(Y_l) + 2^{n/2}((X_l+X_r)(Y_l+Y_r)) - X_l Y_l - X_r Y_r) + X_r Y_r$

$\therefore$ Since, n may be odd,

$XY = 2^{2ceil(n/2)} (X_l)(Y_l) + 2^{ceil(n/2)} [(X_l + X_r)(Y_l + Y_r) - X_l Y_l - X_r Y_r) + X_r Y_r$

This is Karatsuba Algorithm.

Eg Multiplicity 981 and 1234.

$\therefore 981 = 0981$

$\therefore a = 09 \times 10^2 + 81$

19BCE245

Tut-4

$b = 12 \times 10^2 + 34$

$\therefore a_H = 09 \qquad a_L = 81$
$b_H = 12 \qquad b_L = 34$

$\therefore a_L \times b_L = 81 \times 34 = 2754 = C_0$
$\therefore a_H \times b_H = 09 \times 12 = 108 = C_1$
$\therefore (a_L + a_H)(b_L + b_H) = a_L \times b_L - a_L \times b_L =$
$(90)(46) - 2754 - 108 = 1278 = C_2$

$\therefore 981 \times 1234 = C_1 \times 10^4 + C_2 \times 10^2 + C_0$
$= 1210554$

13. The highest upper bound for the worst case performance of quicksort : $O(n^2)$