CS 3510: Design & Analysis of Algorithms August 25st 2024

Problem Set 1: Big-$\mathcal{O}$ and Divide & Conquer

Jinseo Lee Due: September 1th 2024

- Please type your solutions using LaTeX or any other software. Handwritten solutions will not be accepted.
- Your algorithms must be in plain English & mathematical expressions. Pseudo-code will receive no credit.
- Unless otherwise stated, all logarithms are to base two.
- If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists.

1.) (20 points) For the following list of functions, cluster the functions of the same order into one group (i.e., f and g are in the same group if and only if $f = O(g)$ and $g = O(f)$), and then rank the groups in increasing order of growth. You do not have to justify your answer.

(a.) $(\log n)^{\log n}$

(b.) n^{π}

(c.) 3^{2187}

(d.) $50n^{2^{\log 100}} + n$ (Note: $\log 100$ is an exponent to 2)

(e.) $9^{3\log_3 n}$ (Note its 9 to the power of $3\log_3 n$)

(f.) $2048^{\log n}$

(g.) $n^{\log\log n}$

(h.) $n\log(n^{15}) + 7n$

(i.) $n\sqrt{n^7}$

(j.) $\log(n!)$

Solution: c, (j=h), b, i, e, f, d, (a=g)

2.) (20 points) Suppose you are given the following inputs: a sorted array A, containing n distinct integers, a lower bound l, and an upper bound u, where l and u might not be in the array A.

Design a divide & conquer algorithm in $\mathcal{O}(\log n)$ to find the number of elements in A between l and u, inclusive. Make sure to justify the algorithm's correctness and the time complexity (using Master's Theorem).

(a.) Describe your algorithm in plain english.

Solution:

I will use Binary Search to find the indices of the two numbers that best represents the range and subtract those two to find the number of elements in A within the range. Specifically, binary search divides A in half and compare l to the mid, the middle element. If mid is smaller than l, the search proceeds to the right half, and if it is greater than l, it goes to the left half. It continues this process until the index of the smallest element that is greater than or equal to l is found. For u, similar algorithm is applied: Divide in half, compare it with u. It continues this process until the index of the greatest element that is smaller than or equal to u is found. After that, subtract the indices and add 1.

(b.) Argue that your algorithm correctly transforms the input to the correct outputs.

Solution:

A, for example, is an sorted array of [2,4,5,6,7,8,9] and (l,u) = (3,8). Then the mid element is: $n/2 = 3 -> 6$. Since 6 is greater than 3, go to the left half, which is [2,4,5]. The middle index, 4, is greater than 3. Therefore, the smallest element that is greater than or equal to l is 4, whose index is 1. Similarly, the very first middle number, 6, is smaller than 8. Therefore, proceed to the right half, [7,8,9]. The middle point is 8, which is equal to u. The second index is that of 8, which is 5. 5 -(1) + 1 = 5. There are 5 numbers, [4,5,6,7,8] within the range in the array A.

This algorithm generalizes to the input range of 'n' as long as the array is sorted. Regardless of how big or small the size of n is, the array is divided in half and the search is done on each side to find the corresponding element that best represents the range.

(c.) Give the runtime of your algorithm and explain the reasoning.

Solution:

The runtime of binary search is $O(log(n))$ because the search is done on only on the half of array each time (1 sub-problem) with the sub-problem size of n/2. Therefore, the recurrence equation could be expressed as following: $T(n) = T(n/2) + O(1)$, with a=1, b=2, d=0. Since $log_2(1) = 0 =$ d, case 2 of Master Theorem applied. $O(n^d log(n)) = O(log(n))$.

3.) (20 points) Assume n is a power of 6. Assume we are given an algorithm $f(n)$ as follows:

```
function f(n):
   if n>1:
      for i in range(34):
        f(n/6)
      for i in range(n*n):
        print("Banana")
      f(n/6)
   else:
      print("Monkey")
```

(a.) What is the running time for this function $f(n)$? Justify your answer. (Hint: Recurrences)

Solution:

The function calls the loops for 35 times (34 + 1) with the sub-problems size of $n/6$. It also includes extra loop that does $O(n^2)$ work. Therefore, $T(n) = 35T(n/6) + O(n^2)$, with a = 35, b = 6, and d = 2. Since $log_6(35)$ is smaller than 2, case 1 of Master Theorem is applied. Hence, the running time is $O(n^d)$, which is $O(n^2)$.

(b.) How many times will this function print "Monkey"? Please provide the exact number in terms of the input n. Justify your answer.

Solution:

Monkey is printed when the function reaches the base case (n == 1). Since the function divides n by 6 each time, the function reaches the base case after $log_6(n)$. Therefore, in terms of n, the function above prints Monkey $35^{log_6(n)}$ times, which could be expressed as $n^{log_6(35)}$ times.

4.) (20 points) Rohit and Saigautam are trying to come up with Divide & Conquer approaches to a problem with input size n. Rohit comes up with a solution that utilizes 27 subproblems, each of size $n/9$ with time $n\sqrt{n} + n\log n$ to combine the subproblems. Meanwhile, Saigautam comes up with a solution that utilizes 17 subproblems, each of size $n/4$ with time $n^2 + n\log^2 n$ to combine.

(a.) What is the runtime of both algorithms? Which one runs faster, if either?

Solution:

Rohit: a = 27, b = 9, d = 3/2 (polynomial growth $> nlog(n)$). Therefore, $T(n) = 27T(n/9) + O(n^{3/2})$. Case 1 applied as $log_9(27) = 3/2$. Runtime is $O(n^{3/2}log(n))$.

Saigautam: a = 17, b = 4, d = 2 (polynomial growth $> nlog^2(n)$). Therefore, $T(n) = 17T(n/4) + O(n^2)$. Case 3 applied as $log_4(17) > 2$. Runtime is $O(n^{log_4(17)})$

Robit's algorithm is faster than that of Saigautam.

(b.) Let's say Diksha also tries to solve the same problem using an algorithm of her own. She utilizes 11 sub-problems of size $n/6$ with time $\log n$ to combine subproblems. Is this algorithm faster than the one you chose in part (a)? Why or why not?

Solution:

a = 11, b = 6, and suppose d equals 1 (n^1) since there is a difficulty expressing $log(n)$ in terms of n. Then, $log_6(11) > 1$, so case 3 applied. The runtime is $O(n^{log_6(11)})$, which is faster than Robit's from part (a).

5.) (20 points) Assume that n is a power of two. The Hadamard matrix H_n is defined as follows:

$$H_1 = \begin{bmatrix}1\end{bmatrix}$$
$$H_2 = \begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$
$$H_n = \begin{bmatrix}H_{n/2} & H_{n/2}\\ H_{n/2} & -H_{n/2}\end{bmatrix}$$

Design an algorithm that calculates the vector $H_n v$, where n is a power of 2 and v is a vector of length n. Justify the runtime of your algorithm by providing a recurrence relation and solving it. (Hint: you may assume adding two vectors of order n takes $\mathcal{O}(n)$ time.)

(a.) Describe your algorithm in plain english.

Solution:

The algorithm I have came up with will recursively calculate the elements of the matrix until I acquire the numerical value, which could be done by the numerical value of H_1. For example, first divide a vector v in half and now I have V_i and V_j. Next, multiply H_n by V_i and V_j:

$$\begin{bmatrix}H_{n/2} & H_{n/2}\\ H_{n/2} & -H_{n/2}\end{bmatrix}\begin{bmatrix}V_i\\ V_j\end{bmatrix} = \begin{bmatrix}H_{n/2}V_i + H_{n/2}V_j\\ H_{n/2}V_i - H_{n/2}V_j\end{bmatrix}$$

Now I have to recursively conquer the two sub-problems, which are $H_{n/2}V_i$ and $H_{n/2}V_j$ in this case, until the multiplication of H_1 and the output becomes possible. (Base case = H_1).

(b.) Argue that your algorithm correctly transforms the input to the correct outputs.

Solution:

The algorithm utilizes divide and conquer method and splits it into sub-problems. The Hadamard matrix incorporates the recursive process where the vector is divided in half and multiplied by H. This process is continued until it hits the base case, n = 1 (H_1v = vector size of 1). Each step in the recursion matches the construction of H_n, and the process generalizes to all n that is a power of 2.

For example, when n = 4, v is divided into $V_i = [v_1, v_2]$ and $V_j = [v_3, v_4]$. Then it calculates H_2V_i and H_2V_j to produce H_4V, which is [H_2V_i + H_2V_j, H_2V_i - H_2V_j].

(c.) Give the runtime of your algorithm and explain the reasoning.

Solution:

Since there are two sub-problems, each input size of 1/2, and extra $O(n)$ time, the recurrence equation for my algorithm is: $T(n) = 2T(n/2) + O(n)$. $log_2(2) = 1 = d$. Case 2 of the Master Theorem is applied, then the runtime is $O(n^d log(n))$, which is $O(nlog(n))$.