CS 3510 Section B: Design & Analysis of Algorithms August 21th 2024

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

YOUR NAME HERE Due: August 30th 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, and the pseudo-code is optional. Pseudo-code, without sufficient explanation, 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 (i.e., f and g are in the same group if and only if $f = \Theta(g)$ into one group, and then rank the groups in increasing order of growth. You can assume that the logs have a base of 2 unless specified otherwise. You do not have to justify your answer.

(a.) $n^{2\pi}$

(b.) $n^{\log \log n} + 3$

(c.) $n \log(2n) + 0.01n!$

(d.) $12n^{2^{2 \log 10}} + n^2$ (Note: $2 \log 10$ is an exponent to 2)

(e.) $2^{(\log(n))^2}$ (Note: $(\log(n))^2$ is an exponent to 2)

(f.) $1024^{3+\log n}$

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

(h.) 2^{4096}

(i.) 2^{n+102}

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

Solution:

(h.) is O(1).

(j.) is $O(n \log(n))$.

(a.) is slower than (d.) and are both $O(n^2)$.

(g.) is slower than (b.) and are both quasi-polynomial.

(e.) is slower than (f.) is slower than (i.) and are all $O(2^n)$.

(c.) is $O(n!)$.

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).

Solution:

- *Algorithm:*

```
def findNumElements ( A, n, l, u) :
if A[1] >= l & A[n] <= u: return n
    L = binarySearchChanged (A, 1, n, l] + 1
    U = binarySearchChanged ( A, 1, n, u]
    return U - L

def binarySearchChanged(arr, low, high, x):
while low <= high:
    mid = low + (high - low) / 2
    if arr[mid] == x:
        return mid
    else if arr[mid] > x:
        return binarySearchChanged(arr, low, mid-1, x)
    else:
        return binarySearchChanged(arr, mid + 1, high, x)
else:
return mid
```

- *Correctness:* the findNumElements function first checks if the entire array is in the bounds and responds accordingly, then it uses 2 binarySearchChanged functions to find the lowest value above the lower bound and the highest value before the upper bound. And subtract the two index for the solution. The binarySearchChanged function is almost identical to a binary search function except it returns the midpoint value if the index isn't found.
- *Time Complexity Analysis:* Since time complexity of binarysearch is O(logn), the findNumElements function's time complexity would be T(n) = 2O(logn).

3.) (20 points) Assume n is a power of 4. Assume we are given an algorithm $f(n)$ as follows:

```
function f(n):
   if n>1:
      for i in range(20):
        f(n/4)
      for i in range(n*n):
        print("Banana")
      f(n/4)
      f(n/4)
   else:
      print("Monkey")
```

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

Solution:

$T(n) = 22T(n/4) + n^2$

$T(n) = aT(n/b) + n^d$

$a = 22, b = 4d = 2$

$logb(a) = log4(22)$ and b/c log4(22) is larger than 2, $O(n^{\log b(a)})$

$T(n) = O(n^{\log 4(22)})$

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

Solution: Since is a power of 4, we can express n as 4^k for some integer k. The recursion depth is because each recursive call reduces n by a factor of 4. There are 22^k leaves, and k=log4(n), thus $22^{\log 4(n)}$ "Monkey" is called. Simplifying would give $n^{\log 4(22)}$.

4.) (20 points) Kartik and Richard are trying to come up with Divide & Conquer approaches to a problem with input size n. Kartik comes up with a solution that utilizes 32 subproblems, each of size $n/16$ with time $n\sqrt[4]{n} + 2n \log n$ to combine the subproblems. Meanwhile, Richard comes up with a solution that utilizes 24 subproblems, each of size $n/5$ with time $3n^2 + n \log^2 n$ to combine.

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

Solution: Kartik: $n\sqrt[4]{n} + 2n \log n, 2n \log n$ is the dominating term. $a = 32, b = 16, d = 1$. $logb(a)$ is greater than 1, thus $O(n^{\log b(a)})$

$T(n) = O(n^{\log 16(32)})$

Richard: $3n^2 + n \log^2 n$, $3n^2$ is the dominating term. $a = 24, b = 5, d = 2$. $logb(a)$ is greater than 2, thus $O(n^{\log b(a)})$

$T(n) = O(n^{\log 5(24)})$

Overall Kartik's runs faster.

(b.) Let's say Shresha also tries to solve the same problem using an algorithm of her own. She utilizes 2 sub-problem of size $\sqrt{n}$ with constant time to combine the subproblems. She defines a base case where $n = 2$. What is the runtime of this algorithm? Is this algorithm faster than the one you chose in part (a)? (Hint: you cannot simply plug in the Master Theorem. Think about how the Master Theorem is derived for this question.)

Solution: T(n) = $2T(\sqrt{n}) + O(1)$ let there be k levels, At level k, the problem size is $n^{1/2^k}$ since we stop when its equal to 2, $n^{1/2^k} = 2$, $1/2^k * logn = 1, 2^k = logn$, and $k = log(log(n))$

$O(1) * k = O(log(logn))$, which is way faster than the one chosen in problem 1.

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

Solution:

- *Algorithm:*

```
def hadamardMatrix (n, v) :
if n=1: return v
else:
    L = hadamardMatrix (A, n/2, v[1, .. n/2]]
    R = hadamardMatrix (A, n/2, v[(n/2)+1, .. n]]
    for i in range(n/2):
        v[i] = L[i] + R[i]
        v[(n/2) + i] = L[i] - R[i]
    return v
```

- *Correctness:* in the hadamardMatrix function, we first set the base case of when length of the vector is 1, just return the vector because Hv = v if n = 1. Then, I recursively call the method again twice spliting them an half and we can calculate Hv1 and Hv2 of length n/2 recursively. That would be 2 x T(n/2). We then combine the two back into a vector length n where the first half is the sum of L and R while the second is the difference between L and R. Assuming adding two vectors of order n takes O(n) time, the time complexity would be T(n) = 2 T(n/2)+O(n). And then return the new vector.
- *Time Complexity Analysis:* T(n) = 2 T(n/2)+O(n)
- $T(n) = aT(n/b) + n^d$
- $a = 2, b = 2d = 1$
- $logb(a) = log2(2)$ and b/c log4(22) is equal to 1, $O(n^d \log n)$
- $T(n) = O(n \log n)$